DESIGN AND ANALYSIS OF DYNAMIC THERMAL MANAGEMENT IN CHIP MULTIPROCESSORS (CMPS)

A Dissertation

by

IN CHOON YEO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2009

Major Subject: Computer Science

DESIGN AND ANALYSIS OF DYNAMIC THERMAL MANAGEMENT IN CHIP MULTIPROCESSORS (CMPS)

A Dissertation

by

IN CHOON YEO

December 2009

Major Subject: Computer Science

ABSTRACT

Design and Analysis of
Dynamic Thermal Management in Chip Multiprocessors (CMPs). (December 2009)
In Choon Yeo, B.S., Dongguk University;
M.S., Dongguk University
Chair of Advisory Committee: Dr. Eun Jung Kim

Chip Multiprocessors (CMPs) have been prevailing in the modern microprocessor market. As the significant heat is converted by the ever-increasing power density and current leakage, the raised operating temperature in a chip has already threatened the system's reliability and led the thermal control to be one of the most important issues needed to be addressed immediately in chip designs. Due to the cost and complexity of designing thermal packaging, many Dynamic Thermal Management (DTM) schemes have been widely adopted in modern processors.

In this study, we focus on developing a simple and accurate thermal model, which provides a scheduling decision for running tasks. And we show how to design an efficient DTM scheme with negligible performance overhead. First, we propose an efficient DTM scheme for multimedia applications that tackles the thermal control problem in a unified manner. A DTM scheme for multimedia applications makes soft realtime scheduling decisions based on statistical characteristics of multimedia applications. Specifically, we model application execution characteristics as the probability distribution of the number of cycles required to decode frames. Our DTM scheme for multimedia applications has been implemented on Linux in two mobile processors providing variable clock frequencies in an Intel Pentium-M processor and an Intel

Atom processor. In order to evaluate the performance of the proposed DTM scheme, we exploit two major codecs, MPEG-4 and H.264/AVC based on various frame resolutions. Our results show that our DTM scheme for multimedia applications lowers the overall temperature by 4 °C and the peak temperature by 6 °C (up to 10 °C), while maintaining frame drop ratio under 5% compared to existing DTM schemes for multimedia applications. Second, we propose a lightweight online workload estimation using the cumulative distribution function and architectural information via Performance Monitoring Counters (PMC) to observe the processes dynamic workload behaviors. We also present an accurate thermal model for CMP architectures to analyze the thermal correlation effects by profiling the thermal impacts from neighboring cores under the specific workload. Hence, according to the estimated workload characteristics and thermal correlation effects, we can estimate the future temperature of each core more accurately.

We implement a DTM scheme considering workload characteristics and thermal correlation effects on real machines, an Intel Quad-Core Q6600 system and Dell PowerEdge 2950 (dual Intel Xeon E5310 Quad-Core) system, running applications ranging from multimedia applications to several benchmarks. Experiments results show that our DTM scheme reduces the peak temperature by 8% with 0.54% performance overhead compared to Linux Standard Scheduler, while existing DTM schemes reduce peak temperature by 4% with up to 50% performance overhead.

To my family

ACKNOWLEDGMENTS

I am sincerely grateful to my advisor, Dr. Eun Jung Kim, for allowing me to conduct research with her. I am constantly amazed by her extraordinary ability to transform seeming unsolvable problems into a tractable form her infinite knowledge on subject matters, and her relentless attention to detail. Her exceptional commitment to research and strong demand for excellence have guided me this far. I am truly grateful to her insightful advice, her encouragement and constant motivation throughout this work.

I would also like to thank professors Valerie E. Taylor, Hank Walker, and Peng Li for their service on my advisory committee. Their insightful comments and constructive criticism helped me improve my research. In addition, I am deeply grateful to Dr. Kihwan Yum for giving me powerful advising during this study.

Furthermore, I would like to thank my friends and fellow students at Texas A&M University for numerous discussions about various issues related to research and lives. I sincerely thank current and former members of the High Performance Computer Lab for being supportive of me during this work. I also want to thank Seung-Ryong Kang, Young-Woo Ahn, Seung-Jin Sul for being great friends and for always being available whenever I need their assistance and help. I also thank to Chih-Chun Liu for being a great friend and for always being available whenever I need his assistance and help. All my friends at Texas A&M University have helped me in various ways during the years of my Ph.D. program. I thank them all, especially Sun-Hwan Jang, Jae-Woo Seo, Yoon-Jin Kim, Gun-Hee Jo, Ja-Ryeong Koo, Baik-Song Ahn, Heung-Ki Lee, and Ju-Young Jung.

Last, but not least, I would like to thank my parents and my family members for their continuous support and encouragement. I am especially grateful to my wife for her endless support and love. Without her dedication and belief in me, this work would have been impossible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

Chip Multiprocessors (CMPs) have become the main trend in the design of new generation processors. CMP architectures include several cores within one single die area to increase their performance. Instead of pushing the limits of a processor's frequency, the demand for more capable microprocessors must be satisfied by other methods. However, decreased chip size and increased power-density produces a significant amount of heat, threatens system performance and reliability, and even increases power leakage. This heat dissipation is pushing the limits of current packaging technology and cooling solutions. Packages are designed for the worst typical behaviors and rely on Dynamic Thermal Management (DTM) techniques to control temperature at runtime. Therefore, over the past decade, chip design trends have shift to providing more effective thermal managements.

Brooks and Martonosi [1] state the key goals of DTM as: (1) providing inexpensive hardware or software responses, (2) reliably reducing power, and (3) impacting performance as little as possible. Although many hardware-based temperature control techniques, such as Dynamic Frequency and Voltage Scaling (DVFS) and clock gating, have been proposed and applied in modern processors, the demand for more efficient techniques is prevailing in modern microprocessors. [1, 2, 3, 4, 5, 6].

Although modern microprocessors can meet the computation requirement for multimedia data, the high definition multimedia data requires high computations, which converts into a huge amount of heat in modern embedded systems. Therefore, it is critical to keep the temperature of microprocessors under safe limits at runtime.

The journal model is *IEEE Transactions on Networking.*

In DTM schemes for modern processors, DVFS is a common method to control temperature. However, due to the nature of multimedia applications with different frame sizes and types in data, it is not easy to match their QoS requirements while temperature is under control. There have been a handful of studies on thermal managements for multimedia systems [7, 8, 9, 10, 4, 11]. However, according to our observations, these schemes tend to overestimate or underestimate multimedia application requirements. Their results inevitably lead to manage operating temperature high or degrade their performance. In order to compensate the tradeoff between QoS and temperature control, we derive application execution characteristics in various multimedia codecs such as MPEG-4 and H.264/AVC. The application execution characteristics can be represented by the probability distribution of cycle demand that is the number of cycles required to decode a frame. Using this representation, we estimate an adequate processor speed to execute multimedia applications for decoding frames at runtime. Then, we develop Thermal-Aware Scheduler (TAS) that takes optimal frequencies to avoid thermal emergencies while minimizing performance degradations in real environments. We experiment on Intels Pentium-M and Atom processors using various multimedia data encoded by MPEG-4 and H.264/AVC. Compared to feedback control scheme [10], Frame-based scheme [12] and cycle counter-based scheduler [13], TAS lowers average temperature by 6 °C and peak temperature by 10 °C or more, with maximum 5% frame drop ratio.

In DTM schemes for CMP architectures, a thread migration has been proposed to achieve thermal balance among cores without throttling the computation performance in CMP architectures. However, several studies are reactive to the increased chip temperature, while others, such as [4, 5], are proactive based on the predicted future temperature. The proactive DTMs are more effective in temperature control and preventing thermal emergencies, for they trigger the control schemes before the core temperature reach the desired threshold. Since applications have used different functional units that can affect operating temperature, the temperature difference among applications can be up to 9 °C [2, 3]. In fact, the temperature difference between on-chip components can be as much as $10 \sim 15$ °C [3]. Also, all applications do not result in the same heat dissipation pattern. In other words, there are significant variations in the thermal characteristics of different applications [3, 4] and different cores in the same chip.

Also, according to our observations, the temperature of a core varies by 2 °C to 16 °C depending on different levels of thermal correlation in a 4-core CMP system. The temperature of a component is highly correlated with those of other components in the same chip [14, 15, 16, 3]. The temperature model, capturing correlation effects in a uniprocessor, cannot be directly applied to that of a CMP, due to their potential heterogeneity where each core has an independent task to run. Furthermore, [14, 4] have already reported the significant variations in the thermal behaviors of different applications. Although there have been a handful of studies using simple workload models, such as average workload and Instructions Per Cycle (IPC), these studies measured the workload information offline for temperature managements [17, 18]. We believe that it is critical to develop an efficient and online application-based thermal model for DTM which is applicable to real world applications that have dynamic workload behaviors and distinct thermal contributions to the chip temperature. To demonstrate the proposed DTM scheme's scalability and efficiency, especially to satisfy the demand of thermal control in the recent server environments, we implement and evaluate our DTM scheme on real machines, an Intel Quad-Core Q6600 system and Dell PowerEdge 2950 (dual Intel Xeon E5310 Quad-Core) system, running applications ranging from multimedia applications to several benchmarks. For several applications with dynamic workload behaviors, experiments results show that our

DTM scheme reduces the peak temperature by 8% with 0.54% performance overhead compared to Linux Standard Scheduler, while existing DTM schemes reduce peak temperature by 4% with up to 50% performance overhead.

In summary, this thesis focuses on dynamic thermal managements in CMP architectures with negligible performance overhead. Specifically, the contributions of this thesis are follows:

- *A near-optimal thermal management in various multimedia systems.* We estimate multimedia applications' thermal characteristics using statistical approaches to be suitable for various multimedia codecs. A thermal-aware scheduler for multimedia systems provides soft realtime performance guarantees with statistical processor allocations.

- *A better understanding of how the workload for running applications affects operating temperature in CMP architectures.* We define the dynamic workload for running applications as a statistical function (cdf) and a workload characteristics function $w(t)$ at a given interval time $[t-1, t]$. Also, the workload characteristics function consists of a positive factor and negative factors obtained by Performance Monitoring Counters (PMC). These analytical results provide a statistical approach to understand the temperature variations influenced by applications.

- *Thermal correlation effects can explain how the heat transfer works in real CMP products.* Thermal models for CMP architectures should consider the heat transfer among cores, which is defined as the thermal correlation effects. The thermal model, capturing thermal correlation effects in a uniprocessor, cannot be directly applied to the thermal model of CMP architectures due to the potential heterogeneity where each core has an independent task to run. We provide an extended thermal model based on thermal correlation effects in CMP architectures.

- *A measurement of temperature via Digital Thermal Sensor (DTS).* In order to estimate temperature through Digital Thermal Sensor (DTS) in CMP architecture, we develop a specific device driver to access them at runtime. In a silicon die of CMP, each core has a unique thermal sensor that triggers independently. The trigger point of these thermal sensors is not programmable by software since it is set during the fabrication of the processor [19].

- *Application-Based Thermal Management.* Basically, an Application-Based Thermal Management (ABTM) consists of three major components: an application-based thermal model, a future temperature estimation, and a thermal-aware scheduler. We used a specific device driver for Linux to access the Digital Thermal Sensor (DTS) and measure each core's real temperature, and then used the temperature information in the future temperature estimation. Also, we used an application-based thermal model to exploit the thermal model according to each application's execution behavior. At the same time, the future temperature estimation utilizes the workload characteristics and thermal correlation effects to estimate the future temperature and the time left until the temperature reaches the migration threshold. Hence, the thermal-aware scheduler is able to react to the thermal emergency appropriately using the estimated information.

The rest of the thesis is organized as follows. In Chapter II, we describe background and related work of this thesis. In Chapter III, we propose an efficient thermal management for multimedia applications. We consider how the performance of a multimedia system is affected by the complexity of scenes, and then we find an appropriate frequency based on the information on scene complexity. In Chapter IV, we first derive the applications' characteristics in various multimedia applications by transmitting MPEG-4 and H.264/AVC encoded by two different frame resolutions. By using those applications' characteristics, we estimate a frequency to execute multimedia applications which decode frames at runtime. In Chapter V, we present an advanced future temperature prediction model for each core. This allows us to estimate the thermal behavior considering both core temperature and applications temperature variations, and to take appropriate measures to avoid thermal emergencies. In Chapter VI, the proposed thermal-aware scheduler scheme utilizes an advanced future temperature prediction model for each core to estimate different thermal behaviors and measures the amount of time it takes for each core to reach the desired temperature threshold. In Chapter VII, we further model thermal correlation effects by profiling the thermal impacts from neighboring cores under the specific workload. Finally, in Chapter VIII, we propose a thermal model based on thermal correlation effects and online workload estimation using architectural information. In Chapter IX, we conclude this thesis and discuss future work.

CHAPTER II

BACKGROUND AND RELATED WORK

In this chapter, we review prior work in the areas of dynamic thermal managements.

A. Dynamic Thermal Management in Single Core Architecture

Several schemes using architecture adaptation provide Dynamic Thermal Management (DTM) solutions [1, 2]. Brooks and Martonosi suggested the fetch toggling to avoid thermal limit using the stall of instruction fetching [1]. Heo, *et al.* transformed the fetched computation into other duplicated unit during cooling down the overheated unit [2]. However, these schemes cannot satisfy the workload deadline in real-time. Especially, missed deadlines result in low performance in multimedia systems.

Skadron, *et al.* suggested several thermal management schemes including temperature-tracking [16], hybrid scheme [3], and feedback control [20, 10]. [16] used the temperature-tracking scheme to manage temperature based on frequency scaling, localized toggling, and computation migrations. [3] proposed a hybrid scheme combining fetch gating and DVFS. Also, a feedback control configures temperature based on feedback information [20, 10]. Since the approach introduced by Skadron, *et al.* does not take into account the complexity of scenes for multimedia, it cannot avoid the degradation of performance in multimedia applications with radical picture changes.

In [20], Mircea, *et al.* designed a thermal model using the thermal behavior, thermal resistances, and thermal capacitances within functional blocks at the architectural level. Many temperature researches have adopted this model. Althoug thermal behavior can be detected at runtime, this requires specific hardware, such as the built-in Performance Monitoring Counter (PMC) [21].

DTM schemes can be roughly grouped into two approaches: proactive schemes [4, 22, 8, 9, 23, 7] and reactive schemes [20, 24, 10]. In proactive schemes, the results of the previous task determine the speed of the multimedia system. Pouwelse, *et al.* estimated the decoding time per frame based on the offline information on decoding time and frame size [22, 8]. In [9], the Frame Data Computation Aware (FDCA) estimated the decoding time for incoming frames based on the information on decoding macro blocks. However, their method did not require any pre-profile information. [4] proposed predictive thermal management using profiled information, which showed the maximum performance under the thermal constraints. [23] presented an offline scheduling algorithm to save power with quality degradation.

In contrast, reactive schemes determine the speed of the system based on historical information. [24] designed a user-level power management, in which daemons configure the speed setting of the CPU using the characteristics of applications, such as soft real-time, interactive, and batch program. Also, [20, 10] designed feedback control in multimedia systems. Control modules change the level of frequency based on the occupancy of the display buffer. In [7], Son, *et al.* suggested two schemes including proactive and reactive. Their reactive scheme configured the frequency based on delay and drop frame rate, while the proactive scheme determined the frequency using the predicted decoding time based on the future GOP size. However, this scheme needs a profiling process before decoding a GOP.

B. Dynamic Thermal Management in Multicore Architecture

Nowadays, several thermal control techniques [1, 3] via hardware-based mechanisms, such as Dynamic Frequency Scaling (DFS), Dynamic Voltage Scaling (DVS), and clock gating, have been proposed and applied in modern processors. However, these

DTM mechanisms belong to temporal thermal control and bring high execution performance overhead. Therefore, as multicore systems become more popular, some software-based spatial thermal control mechanisms, such as [6, 17, 5], have been studied in a CMP system.

In [17], the proposed mechanism, called heat-and-run, has two key components: SMT thread assignment and CMP thread migration. Within heat-and-run the SMT thread assignment attempts to increase processor-resource utilization by co-scheduling threads which use complementary resources. The CMP thread migration moves threads away from overheated cores and assigns them to free SMT contexts on alternate cores. This maintains throughput while cooling the overheated cores. They evaluated their experiments in an extended Wattch simulator by running five threads within four cores. Heat-and-run thread assignment (HRTA) and heat-and-run thread migration (HRTM) achieve 9% higher average throughput than stop-go and 6% higher average throughput than DVS. Moreover, [25] confirmed that when performance is constrained by temperature, the performance gains brought by thread migration and the importance of limiting the migration frequency to reduce performance overhead. [17] proposed new migration method for temperature-constrained multicore in order to exchange threads whenever the simultaneous occurrence of a cold and a hot core is detected. The authors demonstrate that their method yields the same throughput as HRTM, but requires much fewer migrations. However, the performance overhead migrations cause is not further considered according to the application memory usage.

Mulas *et al.* show that the thread with less memory usage tends to migrate easier than other threads, thus reducing the performance overhead caused by migrations [6]. However, this study ignores the application workload behaviors are ignored. That implies that sets of running threads will migrate without considering the different thermal effects caused by various threads, while the core temperature reaches the upper/lower temperature threshold. Furthermore, these studies are based on simulated results, and neglect thermal correlation among cores. The power dissipated by the rest of the chip is assumed to be negligible. Moreover, in this studies the migration action is triggered by the current temperature (when temperature is higher than the maximum allowed temperature).

Kumar *et al.* provide using HybDTM, a methodology for fine-grained, coordinated thermal management using both software (priority scheduling) and hardware (clock gating) techniques. HybDTM estimates temperature by using a regression-based thermal model based on Performance Monitoring Counters [26]. However, HybDTM cannot effectively reduce overheat temperature without noticeable performance overhead (9.9% performance overhead compared to cases without any DTM). This is why Performance Monitoring Counters cannot solely estimate real temperature. Also, both priority scheduling and clock gating generated high performance overhead. Most importantly, these proposed thermal management mechanisms make impractical assumptions and have only been evaluated by running benchmarks, which have stable application workloads and thermal behaviors. For example, HRTA cannot co-schedule threads without knowing the thread characteristics, such as Instructions Per Cycle (IPC) information and execution resources.

Liu *et al.* propose an application level power management, called Chameleon, for real-world user applications [24]. Chameleon consists of three components: (1) an OS interface that can be used by power-aware applications to measure their CPU usage, (2) a CPU scheduler that supports per-process CPU power settings and application isolation, and (3) a speed adapter that maps the CPU speed settings to the nearest speed supported by the hardware. However, the need for power-aware applications is impractical, since each application's source codes need to be modified. Otherwise, LongRun is used for the legacy applications. Even though this work deals with power management, it inspired us to develop a scalable DTM for the real-world case and more specifically, to satisfy the demand for thermal control in the recent server environment.

Michaud *et al.* confirm that when performance is constrained by temperature, the performance gains from thread migrations [25]. It also demonstrates the importance of limiting the migration frequency to reduce performance overhead. Hence, several advanced DTM studies [17, 18, 6] advocate providing thermal fairness and reducing the peak temperature through temperature-aware thread migration schemes. However, as presented in [3], an accurate and practical dynamic model of temperature is needed to characterize accurately current and future thermal stress and application dependent behavior, as well as to evaluate architectural techniques for managing thermal effects. Moreover, estimating thermal behavior from the average of power dissipation is unreliable. Therefore, we propose to each application's thermal behavior by characterizing its workload through the statistic probability distribution and online workload estimation based on architectural information through Performance Monitoring Counters (PMC). Most importantly, Shang *et al.* introduce the thermal correlation effects in the on-chip networks [27]. We are motivated to model the thermal correlation effects from neighboring cores in the CMP systems from the architecture level and further present an adaptive and scalable DTM based on the thermal correlation effects for the CMP system.

CHAPTER III

EFFECTIVE DYNAMIC THERMAL MANAGEMENT FOR MPEG-4 DECODING

In this work, we present Dynamic Thermal Management (DTM) based on a Dynamic Voltage and Frequency Scaling (DVFS) technique for MPEG-4 decoding to guarantee thermal safety while maintaining a quality of service (QoS) constraint. Although many low-power and low-temperature multimedia playback techniques have been proposed, most of them are impractical in real-time and have several restricting assumptions. Multimedia data consists of several frames requiring different decoding efforts. Since both temperature and performance of a multimedia system are affected by the complexity of scenes, our main idea is to use the information on scene complexity to find an appropriate frequency. In order to predict the complexity of the current scene, we extract information from the previous group of pictures (GOP) using feedback control with a display buffer. Experimental results with twelve movies show that our DTM scheme guarantees the threshold of temperature (70 °C) while maintaining 0% frame miss ratio. Also, the proposed DTM scheme decreases the average temperature by up to 13% without any additional hardware and playback latency.

The main contributions of this research are summarized as follows:

- Our DTM scheme estimates multimedia application's characteristics according to the complexity of the scene using GOP information and frame drop ratio.
- DTM with the advanced feedback controller provides soft realtime performance guarantees under thermal safety.
- Compared to the prior DTMs for multimedia applications such as feedback

control [10], Frame-based [12], the proposed DTM lowers temperature by 13% on average when running MPEG-4 data under 5% frame drop ratio.

A. Thermal Issues in Multimedia Applications

Thermal issues are becoming critical in multimedia systems to achieve high reliability. Although the speed of a modern microprocessor supports processing of the multimedia data in real-time, a multimedia system consumes lots of power for computation and cooling. General purpose computer systems consume over 25% of the power for energy management such as air conditioning, backup cooling and power delivery systems [1]. However, portable battery-operated devices cannot afford such high cooling power.

Without sufficient cooling, embedded systems suffer from long-time overheating and eventually cause the system to crash. However, reducing the voltage level causes the overall performance to slow down. Therefore, the best solution to reduce energy dissipation with dynamic voltage and frequency scaling (DVFS) techniques is to dynamically adjust the voltage scales, while maintaining the minimum required circuitry to accommodate workloads within appropriate computation time and throughput constraints [12]. Multimedia data consist of different frames with different deadlines to be displayed. MPEG frames are classified into three different coding types including intra (I), predictive (P), and bi-directional (B) that consume various power/energy, which leads to raise a different amount of temperature during decoding frames. In addition, a wide variety of frame sizes make it difficult to predict power consumption and to control temperature. Furthermore, since DVFS reduces the overall computation speed, it is likely to have some frames missing their deadlines. Therefore, it is challenging to find a right speed to control system temperature without quality degradation. In [10, 20], the authors suggested a feedback control from a display buffer to find an adequate speed without quality degradation and to reduce the power consumption of MPEG decoding. However, they did not concern thermal problems and exploited only the buffer occupancy information that is not sufficient to control the speed for both performance and temperature. We observe that the required decoding time depends on the complexity of scenes that can be measured with the number of frames in a group of pictures (GOP), and those frames in a GOP require similar computation time for decoding. Therefore, the previous GOP information can be used to predict the computation power of a current frame. We propose an efficient Dynamic Thermal Management (DTM) scheme for a multimedia system to find an appropriate frequency for the available decoding and display buffer based on an advanced feedback control. Our scheme exploits the previous GOP information considering the trade-offs between the quality of data and thermal safety using a frequency efficient factor.

B. Overview of a Feedback Control

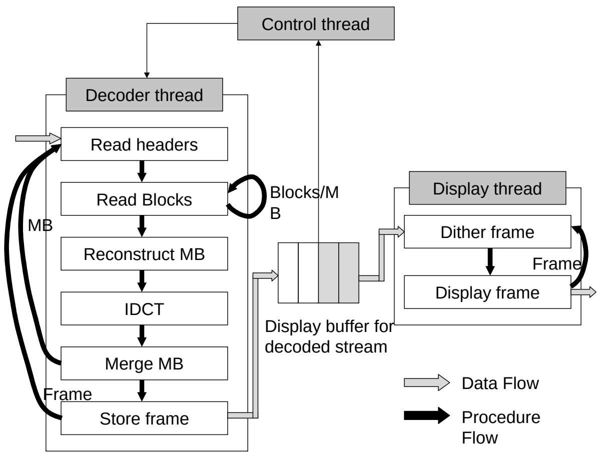


Fig. 1. MPEG process with display buffers

Fig. 1 shows the details of decoding procedures used in [20, 10]. After reading a stream in 'Read headers' and 'Read Blocks' steps, a decoder thread decodes macro blocks at 'Reconstruct MD', 'IDCT', and 'Merge MB.' Finally, the decoder thread puts the decoded frame into a buffer for display. Then a display thread performs the steps of 'Dither' and 'Display' with the frames in the display buffer. The decoder thread executes CPU-intensive operations, while the display thread just displays the decoded frames. To obtain an adequate frequency for the decoding stream, a control thread checks the state of the display buffer. With the high occupancy in the display buffer, the control thread decreases the frequency, since the decoding elapsed time with the current frequency is too much faster than the display elapsed time. On the contrary, the low occupancy lets the control thread increase the frequency to meet the deadline of each frame. Therefore, the performance (i.e., the deadline of frames) should be considered in the low occupancy, while energy efficiency and thermal safety also should be considered in the high occupancy of the display buffer. To fulfill these two considerations, the control thread has to determine an optimal frequency without QoS degradation. Frames should be decoded sequentially and displayed on the display device with a constant playback interval denoted by $t_{interval}$. Although each frame can be decoded at a different elapsed time due to computation variations, each decoded frame in the display buffer are displayed at a uniform speed. To support the QoS requirement, the adjusted frequency should satisfy the following Equation (3.1)[10]:

$$\frac{\sum_{k=1}^{i} D_k}{i} \leq t_{interval}, 1 \leq i \leq n, \tag{3.1}$$

where D_k is the decoding time for frame k, i means the number of frames in the display buffer, and n is the size of the display buffer. Note that for the consecutive frame, we do not have any information about the required decoding time. Without the display buffer, it is difficult to estimate the optimal frequency for decoding the frame. Also, the display buffer with enough space for several frames can make a system determine the optimal frequency without frame misses. Lu *et al.* uses a feedback controller to adjust the frequency with the number of decoded frames in the display buffer within a region specified by $\{B_l, B_h\}$, where B_l is the lower threshold for the number of frames in the buffer and B_h is the higher threshold [10]. Using the feedback controller for decoding frames in the display buffer assumes that the decoder speed is adequate for decoding the frames as long as the number of the frames in the display buffer is within the specified region. However, there are two serious problems in the feedback controller with the display buffer. The first problem is that the feedback controller using the display buffer does not satisfy the deadline of all frames. The frequency is adjusted by the value based on the number of the frames in the display buffer within a region specified by $\{B_l, B_h\}$. This problem occurs with the movies containing several complicated scenes, such as Star Wars 3 and Terminator 3. For example, a high frequency may be required even though the occupancy of the display buffer seems to be sufficient to decode upcoming frames. In such cases, frames will be dropped if the the optimal frequency is only derived from the display buffer occupancy. Before decoding the next frame, we do not know how much time will be required for it. Without a buffer, we must use the most conservative estimate to set the decoding speed. But if there are some decoded frames already in the buffer between the decoder and the display device, we can apply a predictive operating frequency to be close to the optimum. Although the actual decoding time of the frames may vary greatly, its effect on the real-time constraint is hidden by the display buffer. In our approach, we always try to control the number of decoded frames in the buffer within a region specified by $\{B_l, B_h\}$, where B_l is the lower threshold for the number of frames in the buffer and B_h is the higher threshold. As long as the number of frames in the buffer is within the specified region, we assume that the current decoder speed is the right choice for decoding the frames; i.e. the average decode rate is equal to the display rate. However, if the actual number of frames in the buffer becomes higher or lower than the respective threshold, this means the current decoding speed is too fast or too slow, respectively. We apply a formal feedback controller to pull the number of frames back into the specified region by adjusting the decoding speed. The second problem is that the feedback controller using the display buffer cannot control the temperature to guarantee thermal safety. Without considering temperature constraints, the display buffer decides the optimal frequency using only its occupancy. This is a very critical problem in embedded systems, since most embedded systems do not have cooling systems such as a fan.

C. Advanced Feedback Controller Using GOP Information

The previous feedback control scheme uses the occupancy of display buffer between the decoder thread and the display thread to adjust the frequency of a processor to avoid the buffer underflow and overflow. However, although the occupancy of the display buffer is high, the frame may miss when the decoding time of the current frame is longer than the total display time the decoded frames in the display buffer. The relationship among the decoding time for each frame, the display interval and the occupancy of the display buffer is defined in Equation (3.2). Let D_i be a decoding time of $frame_i$. The decoding time D_i should be finished before all previous frames in the display buffer will be presented. Otherwise, $frame_i$ will miss its deadline. Therefore,

$$D_i < n \cdot t_{interval}, \tag{3.2}$$

where $t_{interval}$ is the periodic display time in the display buffer and n is the occupancy of the display buffer.

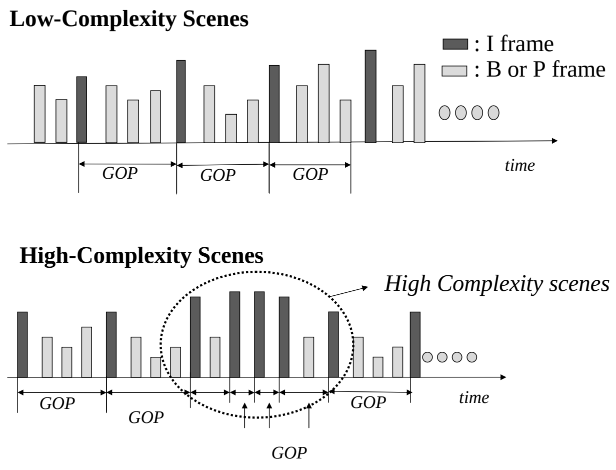


Fig. 2. Low complexity vs. high complexity scenes

As shown in Fig. 2, the number of frames in a GOP decreases when pictures of scenes change rapidly. A single GOP has several frames which consists of I, B, and P frames. And for more complex scenes, the number of B and P frames decreases while only a single I frame is allowed for any GOP. Since the display interval time depends on frames per second (fps) and has a value between 25 *msec* and 30 *msec*, we calculate D_i of the frame that exceeds the display interval time, $t_{interval}$, in this situation. Therefore, the scene complexity can be estimated by adding these D_i values in a GOP.

For example, let's assume that frame f_n should be displayed at time t_n and f_{n+1} should be displayed at t_{n+1}. When frame f_n is ready to be displayed, there are four frames from f_{n-3} to f_n in the display buffer at time t_n. Also assume that a decoding time, D_{n+1} of f_{n+1} and D_{n+2} of f_{n+2} takes three times more than $t_{interval}$. Under these circumstances, all frames ($f_{n-3} \sim f_{n+1}$) in the display buffer are displayed and the buffer will be empty since the next frame (f_{n+2}) has not been decoded. In this case, the frame f_{n+2} is dropped. In order to avoid the future frames drop, the optimal frequency of f_{n+2} should be determined at time time t_n. However, it is difficult to estimate the frequency for a future frame.

We propose a prediction scheme to find the adequate frequency using the information on decoding the previous GOP. According to our experiments, the frame decoding time depends on the complexity of scenes, which continues to exist in several consecutive frames. It means that frames in each GOP have similar complexity of scenes. Therefore, since the GOP consists of several frames, we can predict the optimal frequency of the current GOP using the information on the complexity of scenes in the previous GOP. If the complexity of the previous GOP is high, the complexity of the current GOP will be also high. Therefore, the complexity of the previous GOP can be used as a weight factor to determine the frequency of the current GOP. The weight factor (α) is calculated as follows:

$$\alpha = \frac{\sum_{i=1}^{k} X_i D_i}{\sum_{i=1}^{k} D_i}, 1 \leq i \leq k, \tag{3.3}$$

where D_i is the decoding time of the $frame_i$ and X_i is the indicator which is 1 or 0. With α, the new frequency can be calculated as in Equation (3.4). Let $freq_i$ be the frequency of the decoding time of the $frame_i$. The frequency of the current frame should be configured based on the number of previous frames which have taken less time than the threshold time for displaying that frame.

$$freq_i = (1 - \alpha) \cdot freq_{buf(i-1)} + \alpha \cdot freq_{\max}, \tag{3.4}$$

where $freq_{buf(i-1)}$ is the frequency value to be calculated by the feedback control based on the occupancy of the display buffer. $freq_{max}$ is the maximum frequency of the processor. Hence, the complexity of scenes can be estimated by the weighting factor, α. For example, if the previous GOP has twelve frames and three frames among them have larger decoding time than the selected threshold value, α is calculated as 0.25. It implies that 75% of decoding time of frames in the previous GOP is decoded within threshold and 25% decoding time of frames exceeds the threshold. According to Equation (3.4), the frequency of the current frame is adjusted to handle the frames with higher complexity based on the occupancy of the display buffer. Therefore, the frequency for decoding the current frame is selected by the information of the occupancy of the display buffer and the information of previous GOP. When the higher complexity of the previous GOP is, the higher frequency of the current frame is needed. Using this scheme, we can avoid the missed frames when the complexity of scenes is increased suddenly.

D. Thermal Control Using GOP Information

Although many studies has been focused on the relationship between frequency and power consumption, the relationship between frequency and temperature has to be formulated to find out the optimal frequency within thermal safety. Therefore, we consider a simple thermal model of the processor [28, 29] in that the relationship between processor's frequency and temperature is the basis for any frequency scaling scheme. By modeling the power dissipation or by increasing the input power, more precise models can be derived from this simple model [30].

We analyze Fourier's Law of heat conduction where the rate of heating or cooling is proportional to the difference in temperature between the object and the environment [30]. We define $T(t)$ and $P(t)$ as the temperature and the power at time t, respectively. Then we can use the Fourier's Law as the following [28, 29]:

$$T'(t) = P(t) - bT(t), \tag{3.5}$$

where b is a positive constant representing the power dissipation rate. Now, we define $freq(t)$ as the processor frequency at time t. The power consumption of a processor is an increasing convex function of the frequency [28]. Most work assume that power and processor frequency are relevant as follows [28]:

$$P(t) = a(freq^{\gamma}(t)) \tag{3.6}$$

for some constants a and $\gamma > 1$. With an assumption that $T_0 = 0$ (The initial temperature is the ambient one), through Equation (3.5), the solution to Equation (3.6) can be presented as [30]:

$$T(t) \;=\; \int_{t_0}^{t} a(freq^{\gamma}(\tau)e^{-b(t-\tau)})d\tau + T_0 e^{-b(t-t_0)}. \tag{3.7}$$

Then, for the variation of the temperature, we deal with two cases of the variation at any point t [30]. The first case, when temperature is non-decreasing, by Equation (3.5) and Equation (3.6), can be derived like the following.

$$freq(t) \geq (\frac{b(T(t))}{a})^{\frac{1}{\gamma}}. \tag{3.8}$$

The other case, when temperature is non-increasing, can be expressed as follows:

$$freq(t) \leq (\frac{b(T(t))}{a})^{\frac{1}{\gamma}}. \tag{3.9}$$

Therefore, we observe that scaling the frequency to change the temperature can be performed for the desired direction. Finally, we derive the following equation if we maintain the frequency constant at $freq(t) = freqC$ during time interval $[t_0, t]$.

$$T(t) = \frac{a(freq_c^{\gamma})}{b} + (T(t_0) - \frac{a(freq_c^{\gamma})}{b})e^{-b(t-t_0)}. \tag{3.10}$$

$$dT/dt = -b(T(t) - \frac{a(freq_c^{\gamma})}{b}) \tag{3.11}$$

In addition, the temperature variation by the frequency is based on the equations above (The initial temperature is 0. During [0, t], workload is executed at some frequency level and there is no workload to be executed in the interval [t, t']). Assuming γ=3.0, we can obtain the thermal parameter values for a and b. The values of a and b, are processor-specific but application-independent constants. Also, we can determine the thermal parameters, while observing the heating and cooling curves when we run an application which fully occupies the processor. After a long-time execution of the application, the infinite steady-state temperature value $T(\infty) = T_s$ can be observed. Setting $T(t) = T$ and $a(freq_c^{\gamma})/b = T_s$, Equation (3.5) and (3.6) is transformed as follows:

$$T = T_s + (T_{init} - T_s)e^{-bt}, \tag{3.12}$$

where T_{init} is the initial temperature. Using T_s and sampling the temperature every millisecond, the rate of increase is plotted against (T - T_s) at each point. The resulting set of points is fitted to a straight line using least mean square error fitting. From Equation (3.12), the slope of this straight line represents the value of b. We obtain $b = 0.016$. By applying this value b to the relation, the value a is also obtained as $a = 3.0E - 28$. We used the thermal model above for the simulation. With the plotted temperature variations, we see the effects and decrease in temperature by the suggested DTM for MPEG-4 decoding. All the plots are based on real experimental data including the decoding times of each frame. There can be overestimation of temperature computation, because there can be many short processor idle durations even in the interval between the start and the finish of the decoding. We assume that there is no processor idle duration in the decoding period for a frame. However, even with this assumption, the relative temperature comparisons among four cases shows that the GOP-based DTM can decrease the processor temperature overall. Although the temperature variation by the 1.0 GHz static frequency decoding is shown for the comparison, it is no good in terms of frame miss rate. We show the comparison of the frame miss rate by each method in later section.

E. DTM with the Advanced Feedback Controller

To maintain the temperature under the thermal safety, we should use a DTM scheme for the multimedia decoder. The new DTM scheme uses the accurate frequency from the previous GOP frequency. In our scheme, we decide the threshold of temperature to control the overall temperature during a decoder running. In order to decide the temperature threshold, we need the occupancy of the display buffer which can indicate the efficiency of the frequency for decoding the previous GOP.

$$e = \frac{\sum_{i=1}^{n} D_i}{n \cdot t_{\text{interval}}}, 0 < e \leq 1, \tag{3.13}$$

where $T_{emergency}$ is the maximum allowable temperature and is defined as 80 °C in our experiments. And $T_{threshold}$ is the software threshold of temperature during decoding MPEG-4 stream. Therefore, ΔT can be defined as the difference between an emergency temperature and software temperature threshold. The software temperature threshold is the factor that guarantees thermal safety. n is the total number of frames in the display buffer and $t_{interval}$ is the period of displaying the frames. The e is the frequency efficient factor for decoding frames in the previous GOP only when D_i is equal to or less than $t_{interval}$. With the factor e, we decide the new software threshold of temperature as shown in Algorithm (1). If $T_{current}$ exceeds $T_{threshold}$, $T'_{threshold}$ replaces $T_{threshold}$, and $freq_{max}$ is also replaced by $freq[T'_{threshold}]$. Therefore, $freq_i$ is determined to maintain the temperature under the thermal safety with Equation (3.4) because the determined $freq_{max}$ is smaller than the previous $freq_{max}$.

Algorithm 1 DTM algorithm

```
Require: Define $Table[\,]$ for frequency according to threshold temperature
 1: Determine a threshold temperature($T_{threshold}$).
 2: $i \leftarrow GOP_i$
 3: for $i = 1$ to $GOP_{max}$ do
 4:     Calculate $e$ in $GOP_{i-1}$
 5:     Estimate a current temperature($T_{current}$).
 6:     if $T_{current} > T_{threshold}$ then
 7:         $\Delta T \leftarrow T_{emergency}$ - $T_{threshold}$
 8:         $T'_{threshold} \leftarrow T_{threshold}$ + (1-$e$)$\cdot\Delta T$
 9:         $index \leftarrow index + 1$
10:         $freq_{max} \leftarrow Table[index]$
11:         $freq_i \leftarrow$ (1-$\alpha$)$\cdot freq_{i-1} + \alpha\cdot freq_{max}$
12:     else if $T_{current} <$ ($T_{threshold}$ - $MIN$) then
13:         $index \leftarrow index$ - 1
14:         $freq_{max} \leftarrow Table[index]$
15:         $freq_i \leftarrow$ (1-$\alpha$)$\cdot freq_{i-1} + \alpha\cdot freq_{max}$
16:     end if
17: end for
```

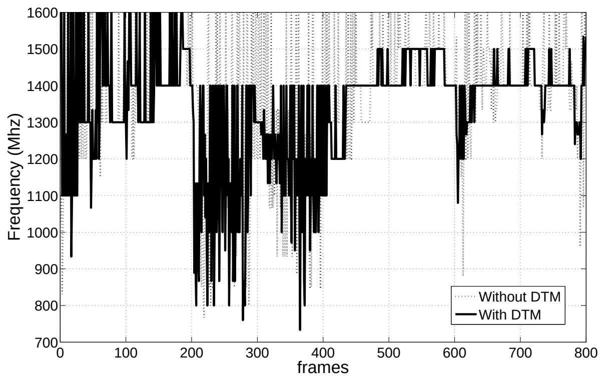


Fig. 3. Comparison with DTM and without DTM for frequency

For example, assuming $T_{emergency}$ to be 80 °C and $T_{threshold}$ to be 60 °C , ΔT is calculated as 20 °C . If e is 0.75 and the current temperature is over the current software temperature threshold, the new software temperature threshold can be adjusted to 64 °C . As a result, $freq_{max}$ is decreased to the next low frequency by $T_{threshold}$. In this example, $freq_{max}$ is adjusted from 1600 Mhz to 1400 Mhz when $T_{threshold}$ is changed. This new software temperature threshold makes $freq_{max}$ decrease the overall temperature. With this scheme, the adjusting temperature threshold can guarantee to maintain the overall system temperature. Also, the processor frequency can be efficiently adjusted at runtime, while taking into account the current thermal condition and the previous frequency. Fig. 3 shows the difference of $freq_{max}$ of the case with DTM or without DTM. The DTM scheme can determine the lower frequency than other DVFS schemes without managing temperature because $freq_{max}$ is changed through the temperature threshold. Fig. 4 shows the effect on the temperature in comparison with DTM and without DTM. The proposed scheme prevents the system temperature from reaching a dangerous level by controlling $freq_{max}$ and maintaining the temperature within the steady state.

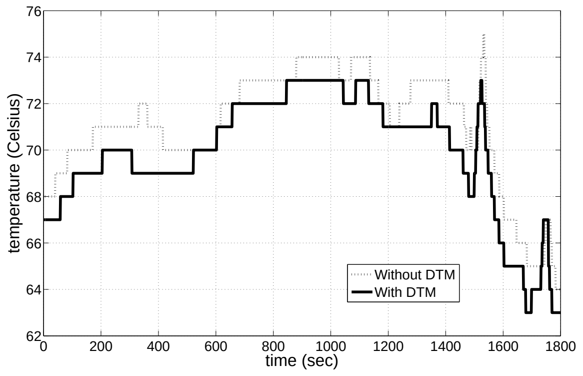


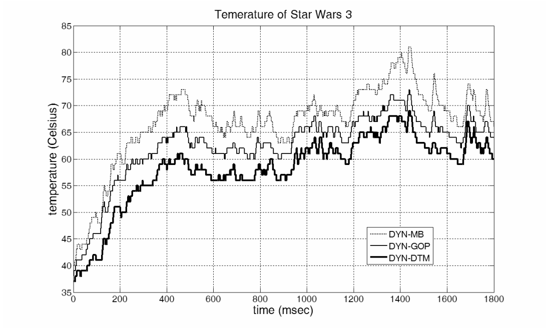
Fig. 4. Temperature comparison with and without DTM

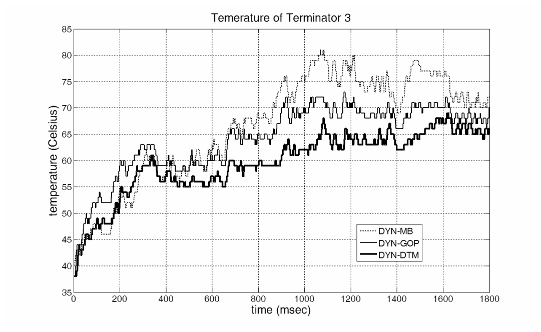
F. Experimental Results and Analysis

To demonstrate the benefits of our control algorithm, we compare three schemes in terms of the number of missing frame, frames per second (fps) and the variance of temperature. `DYN-MB` scheme stands for the feedback controller with the display buffer, `DYN-GOP` scheme is the feedback controller with the display buffer and the information of GOP. Finally, `DYN-DTM` is the feedback controller based on the previous GOP information with DTM. Although `DYN-GOP` and `DYN-DTM` use the information of the previous GOP, only `DYN-DTM` supports the dynamic thermal management. Fig. 5(a) and 5(b) describes the temperature variance in two movies, Star Wars 3 and Terminator 3, which have the higher-complexity data than any other movies. It is observed that the `DYN-DTM` scheme controls the temperature more precisely than the other two schemes. This is a reason why the thermal control in `DYN-DTM` uses the efficiency of frequency and temperature threshold. Another noticeable result is that `DYN-DTM` maintains the peak temperature to be at least 12% lower than other benchmark schemes. As shown in Fig. 5(a), there are the high-complexity scenes in the first part and the last part of this movie, while the middle part has relatively lower complexity. Therefore, the `DYN-DTM` performs efficiently in the first and the last parts while maintaining the software temperature threshold at 70 °C. Fig. 5(b) also shows that `DYN-DTM` outperforms other two schemes even in multiple high-complexity scenes that are located at the middle of the movie. As a result, the proposed `DYN-DTM` scheme reduces the overall temperature up to 13% by using information from the previously decoded GOP and with dynamic thermal management. The most noticeable merit from this scheme is that it prevents all frames from exceeding the threshold temperature without dropping any frame at all.

G. Conclusions

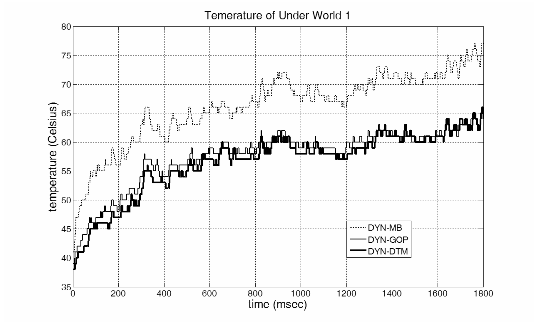
In this work, we proposed a method to find a proper frequency using an advanced feedback controller for the available decoding and display buffer based on the information of the previous GOP. Also, our scheme efficiently adjusts the frequency using a frequency efficient factor, while keeping all frames from being dropped and maintaining thermal safety. We have implemented the proposed scheme on Linux and conducted benchmark testings. Experimental results prove that the proposed method does not drop any frames while the temperature is kept under the threshold. In other words, the proposed scheme suggests a solution for thermal constraints without any quality degradation for MPEG-4 decoding.

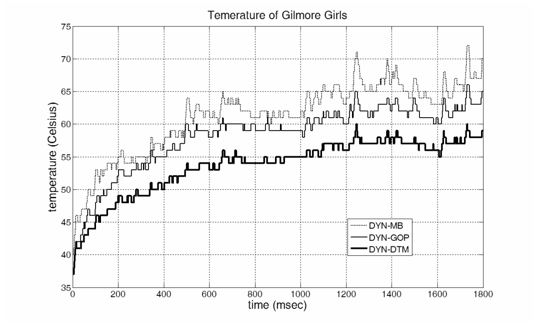




(a) *Star Sars* 3 (b) *Terminator* 3

Fig. 5. Variance of temperature of high-complexity movies





(a) *Under World* 1 (b) *Gilmore Girls*

Fig. 6. Variance of temperature of mid- and low-complexity movies

CHAPTER IV

THERMAL-AWARE SCHEDULING BASED ON STATISTICAL CHARACTERISTICS OF MULTIMEDIA APPLICATIONS

Dynamic Voltage and Frequency Scaling (DVFS) is a common method to control temperature in microprocessors. However, due to the nature of multimedia applications with different frame sizes and types in data, it is not easy to match their QoS requirements while temperature is under control. There have been handful studies on temperature management for multimedia applications [7, 8, 9, 10, 4, 11]. However, according to our observations, these schemes tend to overestimate or underestimate multimedia application requirements, which could result in false, inevitably leading to high operation temperature or performance degradation. In order to compensate the tradeoff between QoS and thermal control, in this work, we first derive application characteristics in various multimedia applications by transmitting MPEG-4 and H.264/AVC encoded by two different frame resolutions. The application characteristics can be represented by *cycle demand*, which is the number of cycles required to decode a frame. Using this representation, we estimate an adequate processor speed to execute multimedia application for decoding frames at runtime. Then, we propose Thermal-Aware Scheduler (TAS) that takes optimal frequencies to avoid thermal emergency while minimizing performance degradation in the embedded environments. To achieve this goal, TAS integrates DVFS feature into the traditional soft real-time scheduler.

Also, TAS can be classified into hybrid schemes to be integrated proactive and reactive approaches together. In the viewpoint of proactive approaches, TAS estimates temperature parameters to get accurate information for temperature according to applications' workload before running multimedia applications. Since those temperature parameters are related to processor and work as architecture specific factors, future temperature can be predicted more accurately using those temperature parameters. In the viewpoint of reactive approaches, TAS utilizes cycle demand distribution as application characteristics for multimedia applications, and those historical information helps the optimal frequency be decided for decoding next frames. As a result, TAS provides better solution to find the adequate frequency based on cycle demand distribution and predict the future temperature using profiled thermal parameters according to workload rather than using only one scheme between reactive and proactive schemes.

We experimented on an Intel's Pentium-M processor and Atom processor using various multimedia data encoded by MPEG-4 and H.264/AVC. Compared to feedback control DTM [10], Frame-based DTM [12] and cycle counter-based scheduler [31], TAS lowers average temperature by 6 °C and peak temperature by 10 °C or more, with maximum 5% frame drop ratio. Moreover, we also compare the predicted temperature by application thermal behavior to the estimated temperature by thermal sensors in Linux while playing movies.

The main contributions of this research are summarized as follows:

- We estimate multimedia applications' thermal characteristics using statistical approaches to be suitable for various multimedia codecs with only 2.5% error on average.
- TAS provides soft realtime performance guarantees with statistical processor allocations. Almost all deadlines of decoding and displaying frames in a lightly loaded real environments, and bounds the deadline miss ratio under the application-specific performance requirement (e.g., meeting 95% of deadlines) in a heavily loaded environment.

- Compared to the previous DTMs such as feedback control [10], Frame-based [12], and cycle counter scheduler [31], our proposed TAS lowers temperature by 10 °C on average when decoding and displaying MPEG-4 and H.264/AVC data under 5% frame drop ratio.
- Although other statistical DVFS algorithm [32, 33, 23] assume that the task decoding frames requires high computation power for multimedia application and the task for displaying decoded frames can be negligible, TAS exploits all tasks related to multimedia applications at runtime and further reduces operating temperature and the frame drops by considering both task-based and system-based characteristics.
- We provide a hybrid estimated scheme with a reactive approach using statistical cycle demand information and a proactive approach for system temperature behavior with a certain workload.

A. The Problems of Multimedia Applications Processing

The display buffer with enough space for several frames can make a system determine the optimal frequency without frame misses. Lu *et al.* uses a feedback controller to adjust the frequency with the number of decoded frames in the display buffer within a region specified by $\{B_l,\ B_h\}$, where B_l is the lower threshold for the number of frames in the buffer and B_h is the higher threshold. Using the feedback controller for decoding frames in the display buffer assumes that the decoder speed is adequate for decoding the frames as long as the number of the frames in the display buffer is within the specified region.

However, there are two serious problems in the feedback controller with the display buffer. The first problem is that the feedback controller using the display

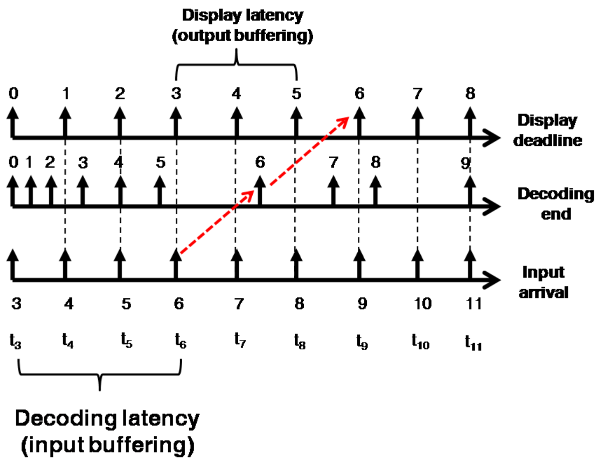


Fig. 7. The timing gap between decoding and displaying data using the buffer management

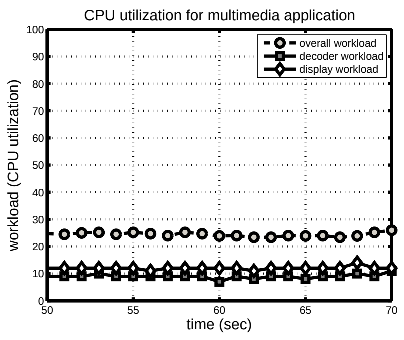
buffer does not satisfy the deadline of all frames. The frequency is adjusted by the value based on the number of the frames in the display buffer within a region specified by $\{B_l, B_h\}$. This problem occurs with the movies containing high complex scenes in succession, such as Star Wars 3 and Terminator 3. For example, a high frequency may be required even though the occupancy of the display buffer seems to be sufficient to decode upcoming frames. In such cases, frames will be dropped if the the optimal frequency is only derived from the display buffer occupancy. Also, since the occupancy of the display buffer is not managed by accurate information such as the executed cycle demand or workload, the feedback control scheme cannot provide a solution to decode and display frames under thermal control. With the feedback control, an optimal frequency can be adjusted by the buffer management, but it may overestimate the necessary frequency estimation. As a result, the operating temperature can be raised than what we expect.

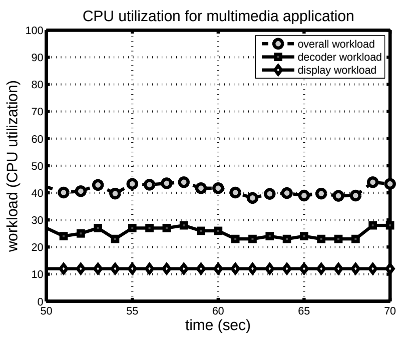
As shown in Fig. 7, even if input for $frame_6$ arrives at time, t_6, display time for $frame_6$ is t_9. Hence, the decoding time for $frame_6$ is larger than the difference between t_6 and t_7. It means that the decoding task for $frame_6$ needs high computation power from the time, t_6. Using the buffer management, however, the high frequency for $frame_6$ can be determined at time t_8, due to consideration of buffer occupancy. Therefore, it is infeasible for adjusting the frequency using only the buffer management scheme. In order to determine the frequency more precisely for multimedia applications, we use the cycle demand, C, to denote the minimum requirement cycle to meet a frame deadline (it is determined by frames per second). The parameter C represents application characteristics and can be explained by workload at runtime. In this work, we can estimate the parameter C using statistical information by Instructions Per Cycle (IPC) and the number of instructions for each frame execution.

The second problem is that the feedback controller using the display buffer cannot control the temperature to guarantee thermal safety. Without considering temperature constraints, the display buffer decides the optimal frequency using only its occupancy. This is a very critical problem in embedded systems, since the most embedded systems do not have cooling systems such as a fan. Therefore, a new approach is required to control temperature without the quality degradation. Although an optimal frequency can be adjusted by the buffer management, it may overestimate the required frequency. Since the response time for decoding task in the buffer management is too long, it cannot provide an immediate solution for adjusting frequency.

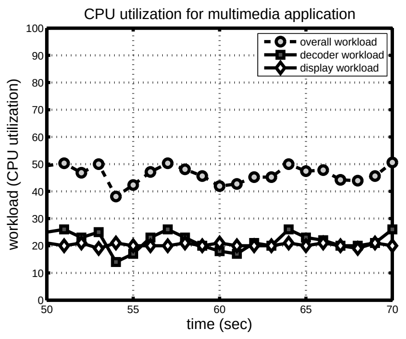
B. The Workload of Multimedia Applications

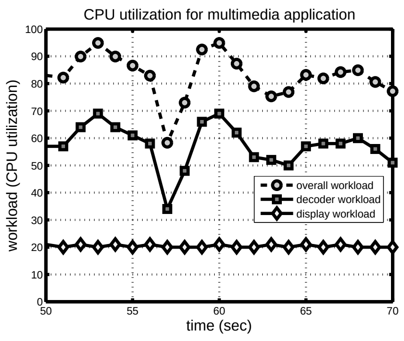
Statistical approaches using DVFS have been proposed to deal with demand variations by considering the probability distribution of CPU demands of individual task. Each task makes a scheduling decision to change CPU frequency using statistical information when a new task starts, and the the frequency is maintained into the same





(a) MPEG-4 data (800X600) (b) H.264/AVC data (800X600)





(c) MPEG-4 data (1280X720) (d) H.264/AVC data (1280X720)

Fig. 8. The workload of decoding and displaying multimedia data according to several codecs and frame resolutions

speed for the whole job [33, 32, 23]. The main approach of TAS is based on these statistical DVFS schemes, but differs from them for three reasons. First, TAS exploits a simple calculation based online profiling to estimate the demand distribution from Performance Monitoring Counters (PMC), while the other approaches use complex estimation approaches or cycle counters to measure the cycle demand for each task. Second, TAS estimates the overall frequency of multiple tasks related to multimedia applications which consist of at least three tasks and each task requires their own the cycle demand. Therefore, all related tasks in multimedia applications should be considered for estimating proper frequency at runtime. In contrast, PACE assumes a single task or treats all concurrent tasks as a joint workload [32]. Also, the estimation based on the cycle counter in process control block (PCB) cannot provide accurate characteristics of multimedia applications.

Finally, TAS supports the latest multimedia data format according to various codecs and frame resolutions. Although the other researches have focused on MPEG-4 codec and small frame resolutions such as 320 X 240 pixels or 640 X 272 pixels, the latest multimedia data format encoded by H.264/AVC codec and high-definition (HD) video frame resolutions supported by HDTV and blue-ray technology requires much more complex computations for decoding frames, and their QoS should be guaranteed in higher standards. The workload of decoding MPEG-4 multimedia data with small frame resolution as shown in Fig. 8(a) requires small CPU computations, but the workload of MPEG-4 data with large frame resolution as shown in Fig. 8(c) requires relatively huge CPU computations. Also, much more CPU computations should be provided for the multimedia data encoded by H.264/AVC, as shown in Fig. 8(b) and 8(d). Since huge CPU computations mean that their works may raise operating temperature and happen large frame drops, more accurate estimation of the demanded cycles for multimedia applications according to various codecs and frame resolutions should be required to guarantee QoS under thermal safety.

C. Thermal-Aware Scheduling for Multimedia Applications

Although the most thermal management schemes have been based on a coarse-grained approach using feedback control of the display buffer, a fine-grained approach using more accurate information of frame and GOP should be considered to find the optimal frequency under thermal safety. Since the reactive schemes have used history information, they cannot provide an immediate solution to avoid critical thermal conditions. Also, the proactive schemes make some overhead to profile workload of multimedia applications before their execution even if the future temperature can be predicted. Moreover, unless temperature cannot be managed even in low temperature degree, it is too late to be controlled when the overheat happens. Therefore, we need more effective scheme which consists of both reactive and proactive schemes together.

In this work, we propose Thermal-Aware Scheduler (TAS) to integrate both proactive and reactive schemes. With the proactive scheme, TAS estimates system thermal characteristics according to workload before running multimedia applications. Since the system thermal characteristics using thermal parameters are dependent on a specific processor or architecture, the thermal characteristics can be measured by the thermal model added the effect of workload. Moreover, since these temperature parameters are determined by processor and architecture specific factors, the future temperature can be predicted more accurately using the temperature parameters. With the reactive scheme, we obtain the probability distribution of cycle demand at runtime, which is the number of cycle required to decode a frame. The probability distribution helps to make a decision of accurate frequency for decoding and displaying frames in multimedia applications.

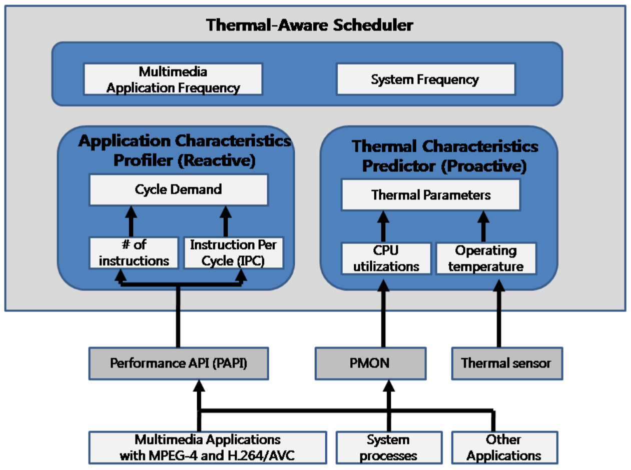

Thermal-Aware Scheduler
Multimedia Application Frequency
System Frequency
Application Characteristics Profiler (Reactive)
Cycle Demand
# of instructions
Instruction Per Cycle (IPC)
Thermal Characteristics Predictor (Proactive)
Thermal Parameters
CPU utilizations
Operating temperature
Performance API (PAPI)
PMON
Thermal sensor
Multimedia Applications with MPEG-4 and H.264/AVC
System processes
Other Applications

Fig. 9. TAS overview

As a result, these proactive and reactive schemes are used to determine an optimal frequency for multimedia applications with negligible performance overhead while controlling temperature.

As shown in Fig. 9, TAS is comprised of three components: an application characteristics profiler as the reactive scheme, the thermal characteristics predictor as the proactive scheme, and the optimal frequency adaptor. The application characteristics profiler exploits Instruction Per Cycle (IPC) and the number of instructions for each frame, and automatically derives the probability distribution of their cycle demands. The temperature predictor based on application workload determines the temperature parameters to predict the future temperature dynamically. The optimal frequency adaptor adjusts the frequency based on information of application characteristics and thermal parameters by workload. Our framework provides the efficient temperature management solution through an integration of application characteristics based on the cycle demand estimation, the thermal prediction based on the statistical characteristics, and DVFS, which are performed by the application characteristics profiler, the thermal characteristics predictor, and the frequency adaptor, respectively. We describe the operations of each component in the following sections.

D. Application Characteristics Profiler for Multimedia Applications

The application characteristics profiler estimates the probability distribution of cycle demands for decoding frames at runtime. We estimate the cycle demand distribution to obtain more accurate multimedia computation requirements. Therefore, compared to other thermal management schemes, we are able to guarantee the thermal safety under the desired temperature without overestimation. Moreover, the performance degradation can be minimized by avoiding underestimation. With these advantages, the cycle demand distribution provides statistical performance guarantees [31], which is sufficient for MPEG-4 and H.264/AVC with various frame resolutions under the thermal control.

To estimate the cycle demand distribution of decoding frames at runtime, we need two steps: the first step is to measure cycle usage measured by Instruction Per Cycle (IPC) and the number of instructions in a fixed window size, and the second step is to derive the probability distribution at runtime. Although the information and estimation through offline profiling can be more accurate, offline profiling makes the additional system overhead, and it is not feasible in multimedia applications which has dynamic workload in each frame. In order to measure the cycle usage for decoding frames at runtime, we exploit Performance Monitoring Counters (PMC) for Intel's Pentium-M and Atom processor, and implement a monitoring module for Instruction Counter and IPC measurement [34]. As a decoding step executes, the executed cycles are calculated by Equation (4.1).

$$C_i = \frac{I_i}{IPC_i}, \tag{4.1}$$

where C_i is the used cycles, I_i is the number of instruction for decoding a frame, and IPC_i is the value of IPC for decoding i_{th} frame obtained by PMC. Next, we can derive the probability distribution of cycle demands in a fixed window size, which is equal to frames per second (*fps*). To do this, we use a profiling window to keep track of the number of cycles consumed by n frames. Even though the parameter n can be specified by the application, we set n to the number of *fps*. Let C_{min} and C_{max} be the minimum and maximum numbers of cycles, respectively, in the window. In our environments, C_{min} and C_{max} are assumed to be 1 million cycles and 10 million cycles because the most multimedia applications requires meeting 96% of frame decoding demands no more than 9 million cycles, and then 9 million cycles per frame is the maximum requirement for decoding in multimedia applications [31]. We obtain a probability density function (*pdf*) and a cumulative distribution function (*cdf*) using following steps:

1. Let X be a continuous random variable and then a probability distribution or probability density function (pdf) of X is a function $f(x)$ such that for any two numbers a and b with $a \leq b$,

$$p(a \leq X \leq b) = \int_{a}^{b} f(x)dx. \tag{4.2}$$

   That is, the probability that X takes on a value in the interval $[a, b]$ is the area under the graph of the density function.

2. Using pdf $p(x)$ in Equation (4.2), the cumulative distribution function (cdf) $F(x)$ of a discrete random variable X with $P(x)$ is defined for every number x by

$$F(x) = P(X \leq x) = \sum_{y \leq x} p(y). \tag{4.3}$$

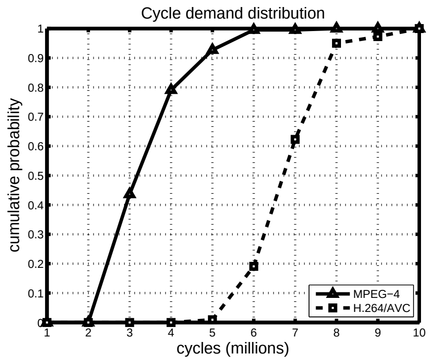


Fig. 10. The cumulative distribution function (cdf) of decoding frames in the multimedia application

3. We denote the *cdf* as $F(x)$ for a random variable X as the number of cycles for decoding a frame according to the pdf $f(x)$ and probability p using Equation (4.3) and (4.4),

$$P(C_{min} \leq X \leq C_{max}) = \int_{C_{min}}^{C_{max}} F(x)dx, \tag{4.4}$$

where X in the interval $[C_{min}, C_{max}]$ with the same sized groups and $F(x) = P(X \leq x) = \sum_{y \leq x} p(y)$. We refer to $c_0, c_1, \cdots, c_n$ with the same size, 1 million, as the group boundaries.

4. For decoding frames in multimedia application, we estimate the probability $P(C_{min} \leq X \leq C_{max})$ in MPEG-4 and H.264/AVC, as shown in Fig. 10.

According to multimedia applications and codecs such as MPEG-4, H.264/AVC with various frame resolutions, the decoding time and computational requirements are different respectively. To satisfy a various computational requirements, a frequency for decoding frames should be decided by the cycle demand based on the probability requirements for decoding frames in the window. Specifically, let ρ be the probability required for decoding frames in a window, and every decoding task for a frame needs to meet the probability ρ of deadlines. In other words, every frame of the window should meet its deadline with a probability ρ. To support this requirement, the C_k cycles should be allocated to all decoding tasks in the same window, i.e.,

$$F(C) = P[X \leq C_k] \geq \rho. \tag{4.5}$$

To determine this parameter C_ρ for a task, we find C_x whose cumulative distribution is at least ρ, i.e., $F(C_x) = P[X \leq C_x] \geq \rho$. Since we assume the probability ρ is 0.96, we determine this C_x as the parameter C_ρ. The demanded probability, ρ_b, is different to allocated probability, ρ_a. Even though ρ_b is required to decode frames on a window, allocated cycle should be split the range $[C_{min}, C_{max}]$ into equal-sized groups. Therefore, we determine a little more allocated cycle (C_a) and allocated probability (ρ_a) by Equation (4.5). In order to get a frequency for decoding frames at a given window, frequency, f_d, can be obtained by Equation (4.6).

$$f_d = \frac{C_\rho \times fps}{\Delta t}, \tag{4.6}$$

where f_d is a frequency for the cycle demand for decoding frames in the window, fps is frames per second, and the time interval Δt is 1 sec. The demanded number of instructions shows the requirement of instructions are different according to frames. By this observation, we can calculate the frequency (f_d) by Equation (4.5) and (4.6). We determine an optimal frequency by the number of instructions for decoding frames using a real multimedia data, and then calculate the allocated instructions based on cdf $F(x)$.

Although we find an optimal frequency for decoding frames, the additional system workload generated by the operating system such as scheduling overhead, file I/O handling, and network monitoring should be considered to guarantee the performance in real systems. The system workload occupies between 5% and 50% according to the assigned frequency. Therefore, the optimal frequency (f_d) for decoding frames should be adjusted by including the system workload.

E. Experimental Environments

Since the thermal management is difficult to be simulated, we have implemented and evaluated TAS in the real-world mobile products. To evaluate the scalability, we conduct our experiments in two different systems as shown in Table I. For our experiments, we have implemented Process Monitoring (PMON) and measured temperature using ACPI on Linux. We used 16 multimedia data encoded by MPEG-4 and H.264/AVC, respectively, with various frame resolutions as shown in Table II and Table III.

Also, we measured the number of instructions and IPC using the Performance API (PAPI) based on performance counter in most major microprocessors [34]. These counters exist as a small set of registers that count events, occurrences of specific signals related to the processor's function. Monitoring these events facilitates correlation between the structure of source/object code and the efficiency of the mapping of that code to the underlying architecture.

In order to demonstrate the flexibility of TAS, we exploit for the experiments

Table I. Experimental systems description

	System I	System II
Processor	Intel Pentium-M 730	Intel Atom N270
Memory Size	1 GB	1 GB
LCD resolution	1600 X 1200	1280 X 800
Maximum Frequency	1.6 Ghz	1.6 Ghz
Minimum Frequency	0.6 Ghz	0.8 Ghz
Scaling level	6 levels	4 levels
Operating System	SUSE 11.1 (Kernel Version: 2.6.27)	SUSE 11.1 (Kernel Version: 2.6.27)

Table II. The experimental multimedia data (Standard Definition)

Name	Encoded Codec	Title	Frame resolution	The number of frames
SD-M1	MPEG-4	Star Wars III	800 X 600	10,000
SD-M2	MPEG-4	Terminator 3	800 X 600	10,000
SD-M3	MPEG-4	24 hours	800 X 600	10,000
SD-M4	MPEG-4	Eragon	800 X 600	10,000
SD-H1	H.264/AVC	The heartbreak kid	800 X 600	4,000
SD-H2	H.264/AVC	300	800 X 600	4,000
SD-H3	H.264/AVC	Apocalypto	800 X 600	4,000
SD-H4	H.264/AVC	Beowulf	800 X 600	4,000

Table III. The experimental multimedia data (High Definition)

Name	Encoded Codec	Title	Frame resolution	The number of frames
HD-M1	MPEG-4	Star Wars III	1280 X 720	10,000
HD-M2	MPEG-4	Terminator 3	1280 X 720	10,000
HD-M3	MPEG-4	24 hours	1280 X 720	10,000
HD-M4	MPEG-4	Eragon	1280 X 720	10,000
HD-H1	H.264/AVC	The heartbreak kid	1280 X 720	4,000
HD-H2	H.264/AVC	300	1280 X 720	4,000
HD-H3	H.264/AVC	Apocalypto	1280 X 720	4,000
HD-H4	H.264/AVC	Beowulf	1280 X 720	4,000

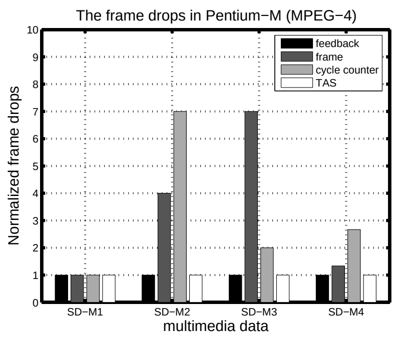
based on various multimedia data with different frame resolutions in two different platforms. For the application with fluctuant workload, we use Mplayer to execute the "Transformers" video clip. One should note that the Mplayer would generate two threads during execution: one is the X windows deamon, which maintains about 30% workload, and the other thread is decoding frames whose workload is fluctuant between 40% and 70%.

F. Experimental Results and Analysis

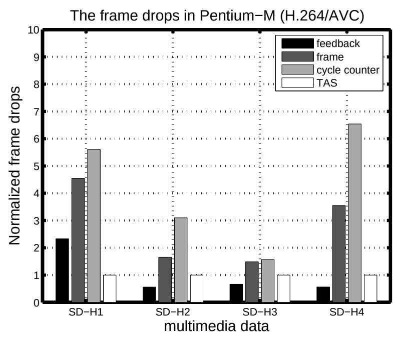
1. The effect on performance overhead and thermal managements in Intel Pentium-M processor

Fig. 11 shows that the frame drop ratio of multimedia data encoded by MPEG-4 and H.264/AVC using different thermal management schemes in Pentium-M processor. Feedback control and frame-based control schemes are indicated **feedback** and **frame**. Cycle counter-based scheduler and TAS are indicated **cycle counter** and **TAS**. The feedback control scheme uses PI controller based on monitoring the occupancy of the display buffer [10]. Also, the frequency is linearly subdivided into 40 discrete levels, which is not true in real systems. Due to using linearly frequency decisions, the actual frequency can be determined higher, and then operating temperature can be managed in higher degrees compared to other schemes. But the frame drop ratio can be reduced, as shown in Fig. 11, which implies that the frequency decision of feedback control scheme overestimates the required frequency compared to other schemes.

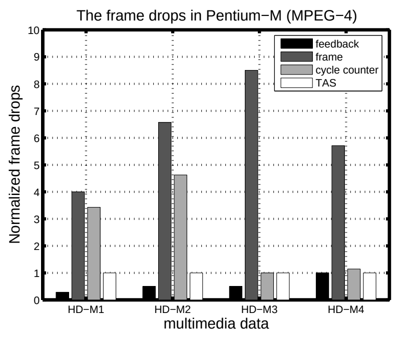
As compared to feedback control scheme, frame-based scheme makes a decision of the optimal frequency by considering the frame-dependent (FD) part of the decoding process whereas the frame-independent (FI) part of dithering and display steps



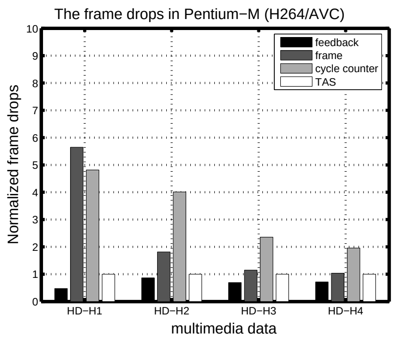
(a) MPEG-4 (800 X 600)



(b) H.264/AVC (800 X 600)



(c) MPEG-4 (1280 X 720)



(d) H.264/AVC (1280 X 720)

Fig. 11. The frame drop of standard definition multimedia data encoded by MPEG-4 and H.264/AVC in Intel Pentium-M processor

decoded frames [12]. Although the FD time varies considerably depending on the frame type, the FI time is nearly constant of the given frames. That implies that the FI time depends on the frame resolution of the given movie stream, which is obviously constant for the same movie. Therefore, frame-based control scheme has higher frame drop ratio in movies based on high definition frame resolutions (1280 X 720) than the standard definition frame resolution (800 X 600), as shown in Fig. 11(c) and Fig. 11(d). Also, they assume that there is no display buffer, i.e., a frame should be decoded and displayed in a given time, determined by a frame rate. However, they do not consider the system workload such as workload generated by operating system, file I/O, and several daemon processes, since the system workload should be one of factors to determine the optimal frequency in real systems. By the frequency decisions based on FD and FI time information, operating temperature can be controlled in lower levels compared to other schemes as shown in Fig. 12(e), Fig. 12(f), Fig. 12(g), Fig. 12(h), Fig. 13(e), and Fig. 13(g).

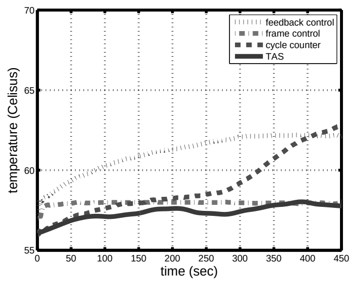
Cycle counter-based scheduler determines the proper frequency based on the statistical information of several frames [31]. Specially, the approach for thermal control based on statistical information overcome the disadvantages of feedback control and frame-based control scheme by estimating relatively the accurate cycle demand for decoding frames. Moreover, the cycle counter-based scheduler prevents potential overheads from frequent changes in the frequency unlike the frame-based control scheme. This is the reason why the cycle counter-based scheduler has superior thermal management compared to the feedback control scheme, but the cycle counter-based scheduler shows higher frame drop ratio than other schemes, as shown in Fig. 11(b) and Fig. 11(d). The reason is that the cycle counter scheme takes no account of the effect by the display task and the system workload. Therefore, their estimation for the optimal frequency is underestimated due to insufficient information through the cycle counter of only decoding task.

TAS is also based on the statistical information of several frames, but TAS takes into account all tasks related to multimedia applications. By this approach, TAS has an advantage of managing lower temperature while reducing frame drops in multimedia applications. Specially, the display task as well as the decoding task should be treated as an important factor for the frequency decision in multimedia data based on the high definition frame resolution (1280 X 720), as shown in Fig. 11(c) and Fig. 11(d). Therefore, TAS overcomes the disadvantage of cycle counter-based scheduler and keeps up better performance compared to other schemes. Also, since more accurate frequency can be determined in TAS, temperature can be lowered than other schemes in almost all results, as shown in Fig. 12 and Fig. 13.

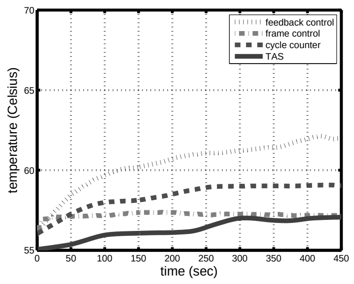
Compared to other schemes, TAS reduces the peak temperature by 6 °C with a reduction of frame drops by average 22.9%. Although temperature managed by TAS shows a little bit higher than by the frame-based control scheme, the frame drop ratio is reduced up to 82.6% compared to the frame-based control scheme.

2. The effect on performance overhead and thermal managements in Intel Atom processor

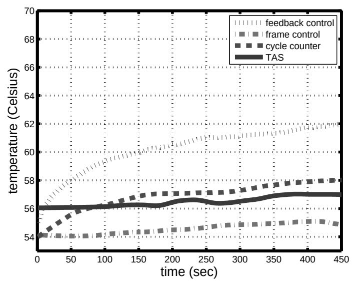
Intel Atom processor based on an entirely new microarchitecture was developed specifically targeted performance and low power for the embedded system [35]. Also, since the data transfer through low power optimized front side bus is faster than Intel's Pentium-M processor, operating temperature can lower compared to Intel Pentium-M processor. Although temperature in Atom processor can be managed lower than Pentium-M processor, the overall frame drop ratio of all experiments in Atom processor is higher than Intel Pentium-M processor, as a shown in Fig. 14. There are two reasons: first, the scaling range for DVFS in Atom processor is smaller than Pentium-



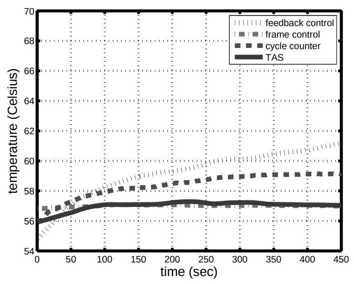
(a) Star Wars 3 (MPEG-4)



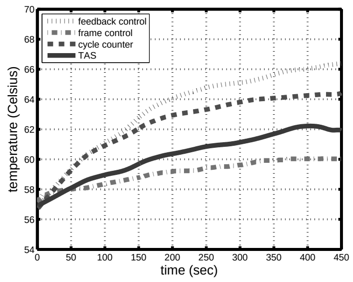
(b) Terminator 3 (MPEG-4)



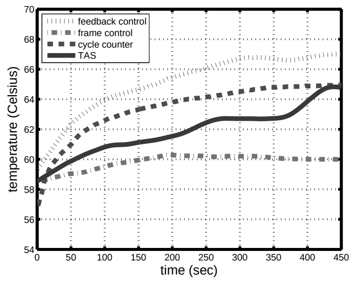
(c) 24 hours (MPEG-4)



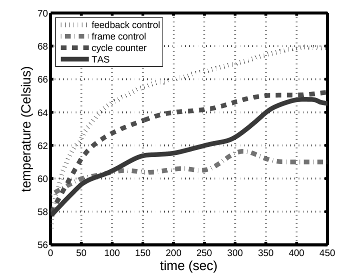
(d) Eragon (MPEG-4)



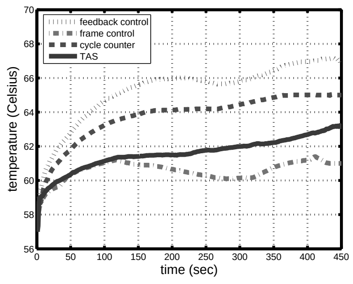
(e) Star Wars 3 (H.264/AVC)



(f) Terminator 3 (H.264/AVC)

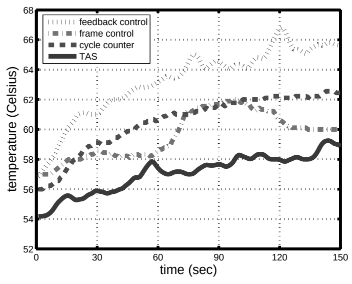


(g) 24 hours (H.264/AVC)

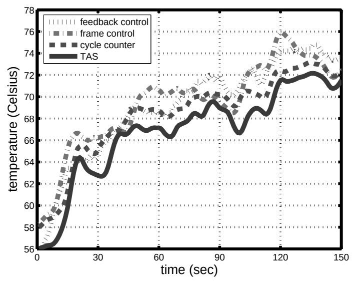


(h) Eragon (H.264/AVC)

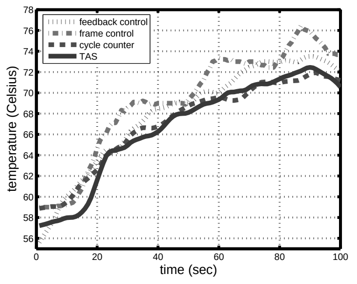
Fig. 12. Resulting temperatures with feedback, frame, cycle counter, and TAS in the standard definition multimedia data in Intel Pentium-M processor (frame resolution : 800 X 600)



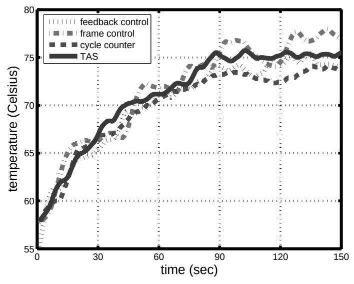
(a) The heartbreak kid (MPEG-4)



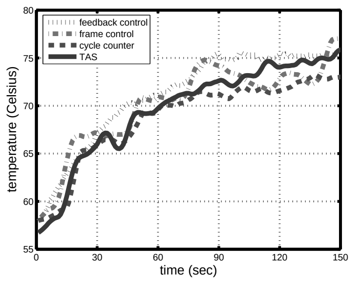
(b) Apocalypto (MPEG-4)



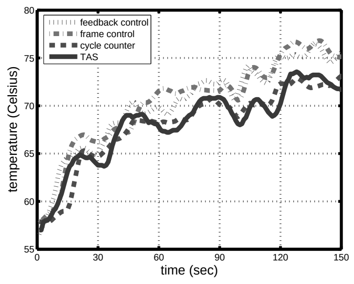
(c) 300 (MPEG-4)



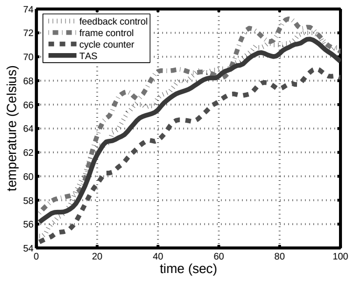
(d) Beowulf (MPEG-4)



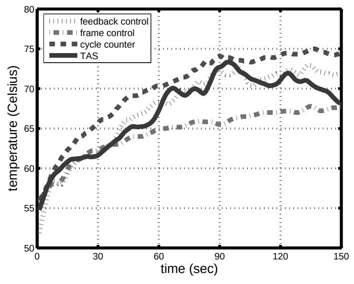
(e) The heartbreak kid (H.264/AVC)



(f) Apocalypto (H.264/AVC)

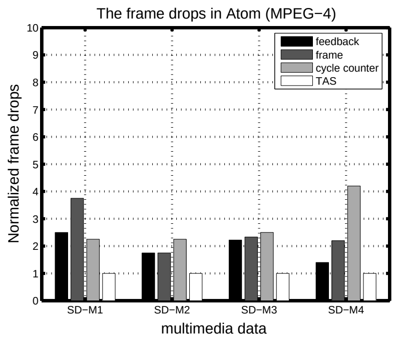


(g) 300 (H.264/AVC)

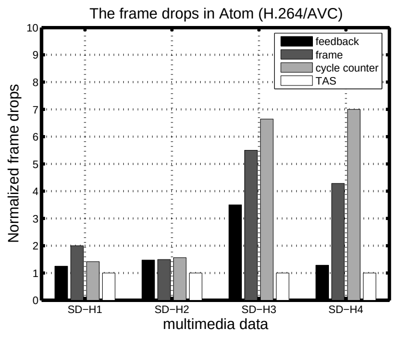


(h) Beowulf (H.264/AVC)

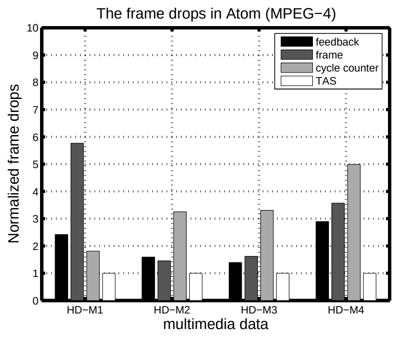
Fig. 13. Resulting temperatures with feedback, frame, cycle counter, and TAS in the high definition multimedia data in Intel Pentium-M processor (frame resolution : 1280 X 720)



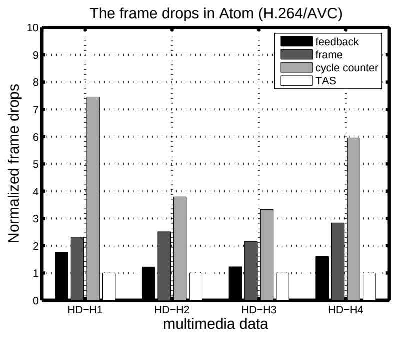
(a) MPEG-4 (800 X 600)



(b) H.264/AVC (800 X 600)



(c) MPEG-4 (1280 X 720)



(d) H.264/AVC (1280 X 720)

Fig. 14. The frame drop of standard definition multimedia data encoded by MPEG-4 and H.264/AVC in Intel Atom processor

M processor. Therefore, the overestimation or underestimation effects can be much more serious than Pentium-M processor. The second is that the overall performance of Atom processor is relatively lower than Pentium-M processor even with the same frequency.

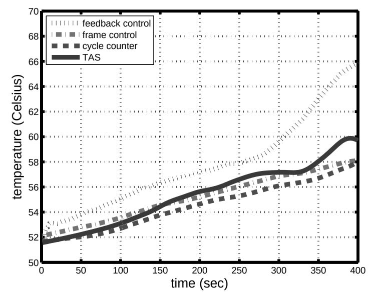
In the experiments for high definition multimedia data in Intel Pentium-M, temperature control using the cycle counter-based scheduler and TAS show the best in almost all results, as shown in Fig. 15 and Fig. 16. This is the reason why the cycle counter-based scheduler and TAS determines more accurate frequency based on the statistical information for the cycle demand to decode frames. However, cycle counter-based scheduler can underestimate the required frequency for multimedia applications whenever the huge decoding time is required in high definition multimedia data. Due to lack of overall estimation for decoding and display tasks, the underestimated frequency causes to increase performance overhead in the overall system. Therefore, the frame drop ratio of cycle counter-based scheme is higher than the other schemes, as shown in Fig. 14(c) and Fig. 14(d). Moreover, the inaccurate estimation of frequency without the workload of displaying frames causes the frame drop ratio bigger than other schemes. Also, since the cycle counter-based scheme depends on the decoding time without any information for temperature, it is unable to predict the future temperature or the thermal characteristics.

Compared to other schemes, TAS reduces the peak temperature by 6 °C with a reduction of frame drops by average 27.3%. Although temperature managed by TAS shows a little bit higher than by the frame-based control scheme, the frame drop ratio is reduced up to 72.2% compared to the frame-based control scheme.

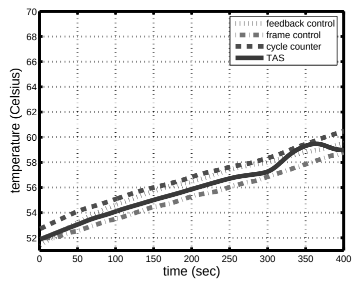
As a result, TAS derives statistical information by taking advantage of the cycle demand obtained by IPC and the number of instructions. Based on the statistical information for the previous frames, this scheme calculates the currently required frequency as well as looking ahead to see if the future frequency should be increased or decreased. This scheme can also adjust the frequency accordingly for movies with rapid changes. The appropriate frequencies in different decoding time can be precisely predicted with IPC and the number of instructions which depends on the processor. Also, this leads TAS to operate in lower temperature levels than other thermal management schemes. TAS based on the application thermal characteristics lowers temperature by about 4 °C in average and reduces about 6 °C in the peak temperature compared to other previous thermal management schemes. Since TAS meets up to 5% frame drop ratios in multimedia applications, TAS outperforms the previous thermal managements for multimedia applications such as the feedback control scheme, the frame-based scheme, and the cycle counter scheduler.

G. Conclusions

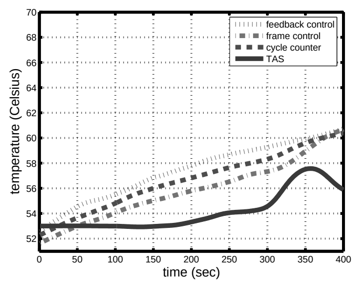
In this work, we propose Thermal-Aware Scheduler (TAS) which uses both application characteristics represented by the probability distribution of cycle demand to decode a frame and the system thermal model augmented by the effect of workload. Our experimental results show that the distribution of cycle demands in various codecs affect temperature directly as an application workload. This implies that the overall temperature can be predicted and controlled by the optimal frequency to decode frames for any type of multimedia data. Also, TAS scheme explores the application thermal characteristics based on statistical information of cycle demands, which can estimate the future temperature within 2.5% prediction error in average compared to the measured temperature by a thermal sensor. Therefore, TAS provides more accurate estimation and more efficient temperature management compared to other schemes such as the feedback control scheme, the frame-based scheme, and the cycle counter scheme.



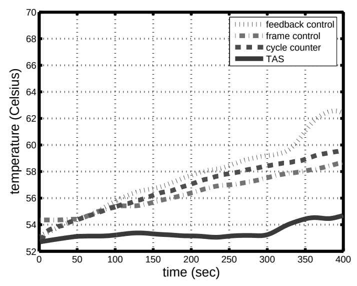
(a) Star Wars 3 (MPEG-4)



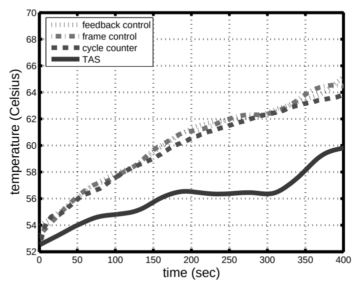
(b) Terminator 3 (MPEG-4)



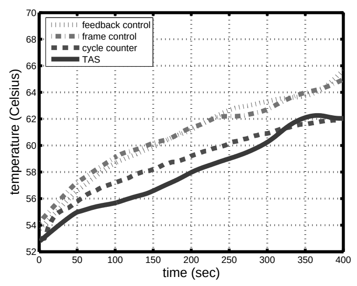
(c) 24 hours (MPEG-4)



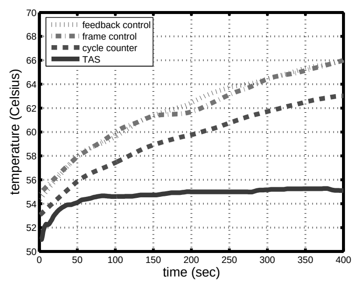
(d) Eragon (MPEG-4)



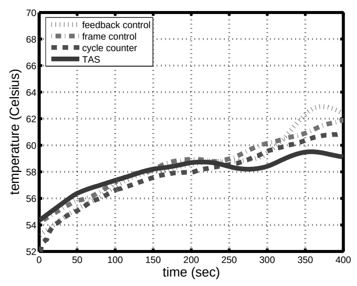
(e) Star Wars 3 (H.264/AVC)



(f) Terminator 3 (H.264/AVC)

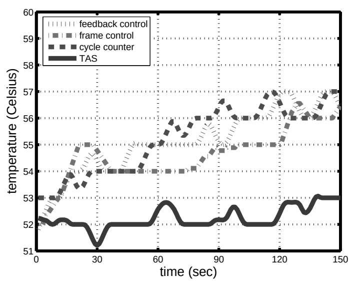


(g) 24 hours (H.264/AVC)

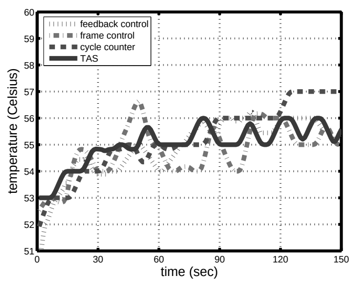


(h) Eragon (H.264/AVC)

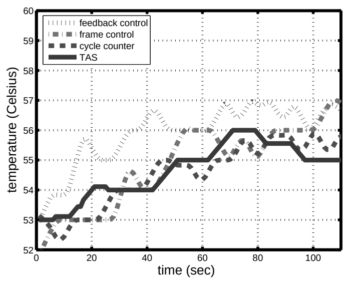
Fig. 15. Resulting temperatures with feedback, frame, cycle counter, and TAS in the standard definition multimedia data in Intel Atom processor (frame resolution : 800 X 600)



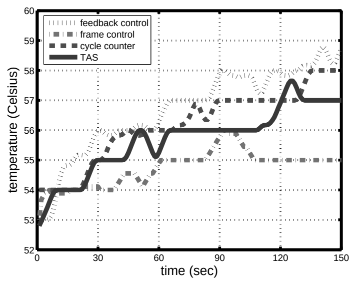
(a) The heartbreak kid (MPEG-4)



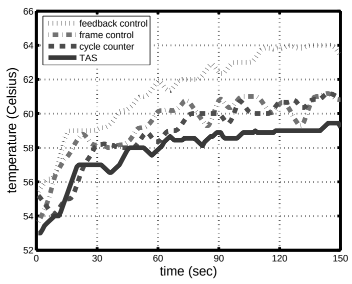
(b) Apocalypto (MPEG-4)



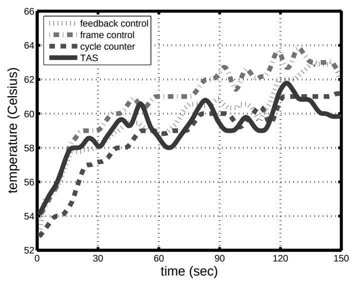
(c) 300 (MPEG-4)



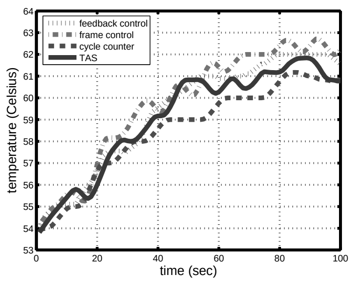
(d) Beowulf (MPEG-4)



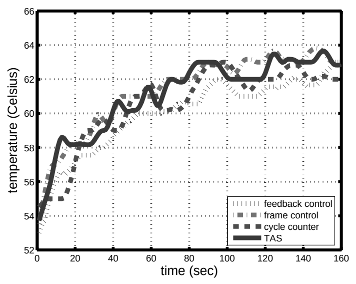
(e) The heartbreak kid (H.264/AVC)



(f) Apocalypto (H.264/AVC)



(g) 300 (H.264/AVC)



(h) Beowulf (H.264/AVC)

Fig. 16. Resulting temperatures with feedback, frame, cycle counter, and TAS in the high definition multimedia data in Intel Atom processor (frame resolution : 1280 X 720)

CHAPTER V

PREDICTIVE DYNAMIC THERMAL MANAGEMENT FOR MULTICORE SYSTEMS

From this chapter, we introduce how to manage temperature in Chip Multiprocessors (CMPs). CMPs have already been employed as the main trend in new generation processors. CMPs includes multiple cores within one single die area to increase the microprocessors' performance. However, the increased complexity and decreased feature sizes have caused very high power density in modern processors. The power dissipated is converted into heat and the processors are pushing the limits of packaging and cooling solutions. The increased operating temperature potentially affects the system reliability. Moreover, leakage power increases exponentially with operating temperature. Increasing leakage power can further raise the temperature resulting in a thermal runaway [1]. Hence, there is a need to control temperature at all levels of system design.

Recently, many hardware and software-based Dynamic Thermal Management (DTM) [1, 3] techniques have been proposed in sense of that they, except [4], start to control the temperature after the current temperature reaches the critical temperature threshold. DTM schemes can be characterized as temporal or spatial. Temporal management schemes, such as Dynamic Frequency Scaling (DFS), Dynamic Voltage Scaling (DVS), clock gating, slowdown the CPU computation to reduce heat dissipation. Although they could effectively reduce temperature, they incur significant performance overhead. On the other hand, spatial management schemes, such as thread migration, can reduce the temperature without throttling the computation [17]. However, neighboring thermal effect and the application thermal behavior are not considered in prior works. Due to packaging technology in CMPs, the temperature of each core will be affected by other cores. The temperature differential between cores can be as much as $10 \sim 15$ °C [3]. There are significant variations in the thermal behavior among different applications [3, 4].

Motivated by these facts, we propose a Predictive Dynamic Thermal Management (PDTM) in the context of multicore systems. Our PDTM scheme utilizes an advanced future temperature prediction model for each core to estimate the thermal behavior considering both core temperature and applications temperature variations, and then take appropriate measures to avoid thermal emergencies. To the authors' best knowledge, no prior attempt has been made to implement the temperature prediction model along with the thermal-aware scheduling on a real four-core product under Linux environment. The experimental results on Intel's Quad-Core system running two SPEC CPU 2006 benchmarks simultaneously show PDTM lowers temperature by about 5% in average and reduces up to 3 °C in peak temperature with only at most 8% performance overhead compared to Linux standard scheduler without DTM. Moreover, to validate the presented PDTM, we also rebuilt HRTM [17], and our PDTM outperforms HRTM in reducing average temperature by about 7%, performance overhead by 0.15%, and peak temperature by about 3 °C, while running single benchmark.

The main contributions of this work are summarized as follows:

- We propose an advanced future temperature prediction model for multicore systems with only 1.6% error in average.
- We demonstrate that our scheme outperforms the existing DTM schemes (HRTM and HybDTM) and provides thermal fairness among cores in a CMP system.
- The proposed PDTM incurs low performance overhead which is only 1% when running single benchmark, and 8% when running two benchmarks simultane-

ously.

- Most importantly, there is no additional hardware unit required for our prediction model and thermal-aware scheme. It means that our model and scheme is scalable for all the multicore systems and can be applied to real-world CMP products.

A. Predictive Thermal Model

In this section, we present a thermal model to predict the future temperature at any point during the execution of a specific application. The model is based on our observation that the rate of change in temperature during the execution of an application depends on the difference between the current temperature and the steady state temperature of the application[1]. Moreover, the thermal behavior is different among applications. Since the system temperature is affected by both each application's thermal behavior and each processors' thermal pattern, we define the application-based thermal model and the processor-based thermal model in this work.

1. The application-based thermal model in CMP systems

The Application-based Thermal Model (ABTM) accommodates the short-term thermal behavior in order to predict the future temperature in fine-grained. As shown in Fig. 17, there are rapid temperature changes even when the workload is statically 100%. Specifically, this model first derives the thermal behavior from local intervals (short term temperature reactions) and then predicts the future temperature by incorporating this behavior into a regression based approach that is known as the

[1]The steady state temperature of an application is defined as the temperature the system would reach if the application is executed infinitely.

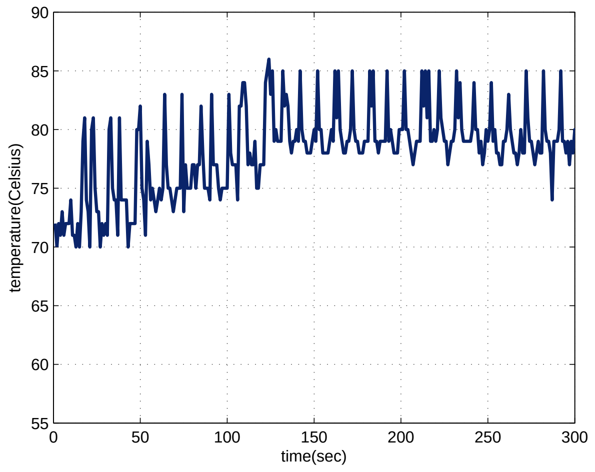


Fig. 17. Real temperature of one core on running `bzip2` benchmark

Recursive Least Square Method (RLSM). In the general least-squares problem, the output of a linear model y is given by the linear parameterized expression

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \cdots + \theta_n f_n(u), \tag{5.1}$$

where $u = [u_1 \,, u_2 \,, \cdots \,, u_n \,]$ is the model's input vector, f_1,...,f_n are known functions of u, and θ_1, θ_2,...,θ_n are unknown parameters to be estimated. In our study, let the input vector, u, and the output vector, y, be time units and working temperature respectively. To identify the unknown parameters θ_i, experiments usually have to be performed to obtain a training data set composed of data pairs $(u_i \,; y_i \,)$, $i =$ $1, \cdots, m\}$. Expressed in matrix notation, the following equation can be obtained: Y

$= X\theta$ where X is an $m \times n$ matrix:

$$X = \begin{bmatrix} f_1(u_1) & \cdots & f_n(u_1) \\ \vdots & \vdots & \vdots \\ f_1(u_m) & \cdots & f_n(u_m) \end{bmatrix} \tag{5.2}$$

θ is a $n \times 1$ unknown parameter vector:

$$\theta = [\theta_1, \theta_2, ..., \theta_n]^T \tag{5.3}$$

and Y is a $n \times 1$ output vector:

$$Y = [Y_1, Y_2, ..., Y_n]^T \tag{5.4}$$

If $X^T X$ is nonsingular, the least square estimator can be derived as

$$\theta = (X^T X)^{-1} X^T Y, \tag{5.5}$$

Denote the i_{th} row of the joint data matrix $[X : Y]$ by $[X_i^T : Y_i]$. Suppose that a new data pair $[X_{k+1}^T : Y_{k+1}]$ becomes available as the $(k+1)^{th}$ entry in the data set. To avoid recalculating the least squares estimator using all input and output data samples, let $P_k = (X^T X)^{-1}$ for the k^{th} in Equation (5.5). Likewise, the recursive least square method at $(k+1)^{th}$ can be developed as

$$P_{k+1} = P_k - \frac{P_k x_{k+1} x_{k+1}^T P_k}{1 + y_{k+1}^T P_k y_{k+1}}, \tag{5.6}$$

where y_{k+1} is the output vector and x_{k+1} is input vector of of f_{k+1}.

$$\theta_{k+1} = \theta_k + P_{k+1} x_{k+1} (y_{k+1} - x_{k+1}^T \theta_k) \tag{5.7}$$

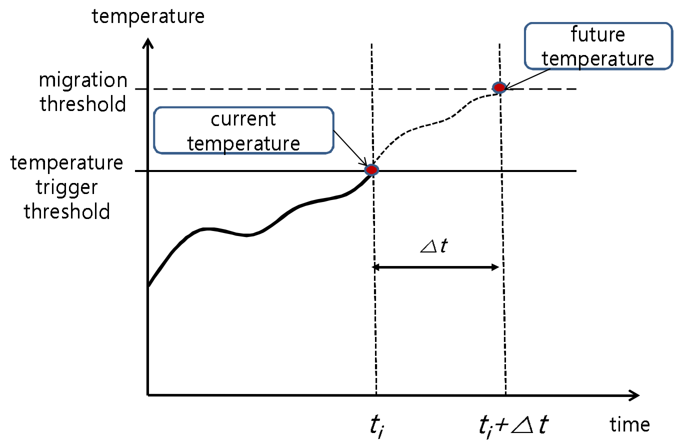


Fig. 18. The calculation of Δt (migration time) using ABTM

where matrix P is an intermediate variable in the algorithm. Eventually, we get future temperature, y_n, by an application thermal behavior using the current θ vector. Detailed descriptions of the Least Square Method and Recursive Least Square Method can be found in the literatures [36]. With Equation (5.1), ABTM can predict future temperature for an application as shown in Fig. 18.

2. The core-based thermal model in CMP systems

The heat transfer equations model the steady state temperature of systems with the heat sources [37]. It has been observed in those models that the temperature changes exponentially to the steady state starting from any initial temperature. In other words, the rate of temperature change is proportional to the difference between the current temperature and the steady state [37]. We initially assume that the steady state temperature of the application is known. Later we will relax this constraint. Let T_{ss} be the steady state temperature of an application. Let $T(t)$ represent the temperature at time t and let T_{init} be the temperature when an application starts execution ($T(0)$=T_{init}). The prediction model assumes that the rate of variations of temperature is proportional to the difference between the current temperature and the steady state temperature of the application [30]. Thus

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{5.8}$$

Solving Equation (5.8) with $T(0) = T_{init}$ and $T(\infty)$=T_{ss}, we get

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{5.9}$$

where b is a processor-specific constant. The value of b is determined using Equation (5.8) by observing heating and cooling curves corresponding to all SPEC CPU 2006 benchmarks on the core. Also, since the value of b is different to the amount of workload, b should be determined by the workload on each processor. Running several benchmarks, we obtained $b = 0.009$ when the workload is 100%. We precompute the steady state temperature of an application offline. By rearranging Equation (5.9), we get the steady state temperature T_{ss} of the application.

$$T_{ss} = \frac{T(t) - T_{init} \times e^{-bt}}{(1 - e^{-bt})} \tag{5.10}$$

Therefore, with Equation (5.9) and (5.10), we get the future temperature after time t and the steady state temperature, T_{ss}, of each core.

3. The predictive thermal model

Our approach, which towards characterizing the thermal contribution of individual processor, uses ABTM and CBTM at runtime as the input for the overall thermal model to directly estimate the future temperature. For each application, we exploit both short-term (ABTM) and long-term (CBTM) future temperature values to prevent Ping-Pong effect[2]. The application-based temperature, T_{app}, predicts the transient variations in application temperature which includes the temperature contribution at the running period on the core before being migrated into other core. On the other hand, the core-based temperature, T_{core}, is calculated with the aggregated temperature by workload. The overall predictive temperature is then given as:

$$T_{predict} = w_s T_{app} + w_l T_{core} \tag{5.11}$$

where $T_{predict}$ is determined as the overall predictive temperature, w_s is a weighting factor of ABTM, and w_l is a weighting factor of CBTM. Note that w_s and w_l should be adjusted according to the application workload. Since the benchmarks we used in this study maintain 100% workload in most time, we find that the optimal values for w_s and w_l are 0.7 and 0.3 ,respectively, based on our experimental results. Instead of iteratively calculating $T_{predict}$, a predefined temperature trigger threshold provokes the calculation. And after a certain core's temperature has exceeded the temperature trigger threshold, it detects processes that exceed the workload threshold and applies appropriate ABTM. This approach can reduce the overheads from the prediction calculations. In order to properly predict any future temperature, we need to know Δt beforehand. This value is the time interval that is required for the current ABTM to reach the temperature threshold of the next stage in Fig. 18.

B. PDTM Scheduler

The Linux standard scheduler is designed to compromise two opposing aspects: response time and throughput. Interactive processes such as shell programming are built to run in a satisfactory response time. On the other hand, CPU-intensive pro-

[2]A process can be migrated among several cores very frequently.

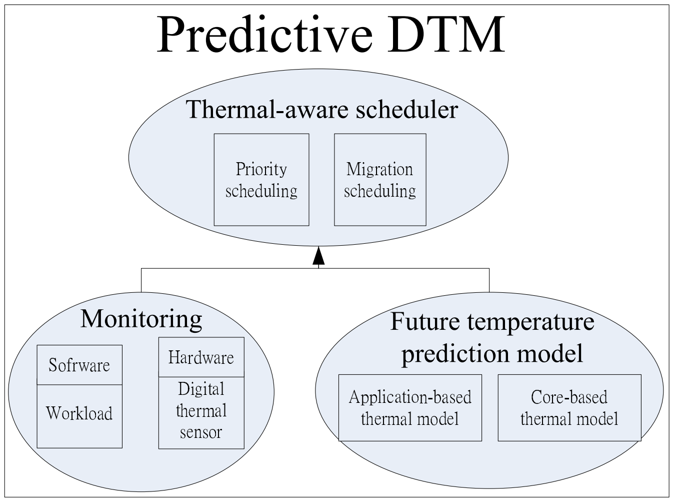


Fig. 19. System overview

grams needs to ensure throughput. To keep up with this corollary in multi-cores, a certain process is rarely migrated into another core in Linux standard scheduler. This is mainly because an active process uses running information like TLB for the process through cache memory [38]. However, when the workload is noticeably unbalanced, the Linux standard scheduler initiates process migrations despite migration overhead. However, the Linux standard scheduler does not take the temperature behavior into account. To resolve this issue, PDTM enables the scheduling policy to accommodate the temperature behavior as well as workloads in a multicore environment.

Our PDTM mainly composes of three components as shown in Fig. 19. In the monitoring part, the application workload (CPU utilization) is monitored for application's migration to balance workload by Linux standard scheduler. However, it is not aware of temperature. PDTM uses Digital Thermal Sensor (DTS) to detect temperature at runtime. The detected temperature information can be used in the future temperature prediction model.

As shown in Algorithm (2), PDTM determines that migration is necessary when

Algorithm 2 PDTM scheduler algorithm

```
1: $T_{cur} \leftarrow$ CalcT($process_i$)
2: for $T_{cur} \geq T_{ttt}$ do
3:     $\Delta t_m \leftarrow$ ABTM$^{-1}$($T_{tmt}$)
4:     for $j = 1$ to $MAX_{cores}$ do
5:         $T_{cbtm} \leftarrow$ CBTM($\Delta t_m$)
6:         $T_{abtm} \leftarrow$ ABTM($\Delta t_m$)
7:         $T[j] \leftarrow \omega_s \cdot T_{abtm} + \omega_l \cdot T_{cbtm}$
8:     end for
9:     Migrated_Core $\leftarrow$ MIN_CORE($T[]$)
10:    $T_{pred} \leftarrow$ MIN_TEMP($T[]$)
11:
12:    if Current_Core $\neq$ Migrated_Core then
13:        MIGRATION($process_i \rightarrow$ Migrated_Core)
14:    end if
15:
16:    if $T_{pred} \geq T_{pst}$ then
17:        Decrement priority($process_i$)
18:    else
19:        Increment priority($process_i$) until priority = 0
20:    end if
21: end for
```

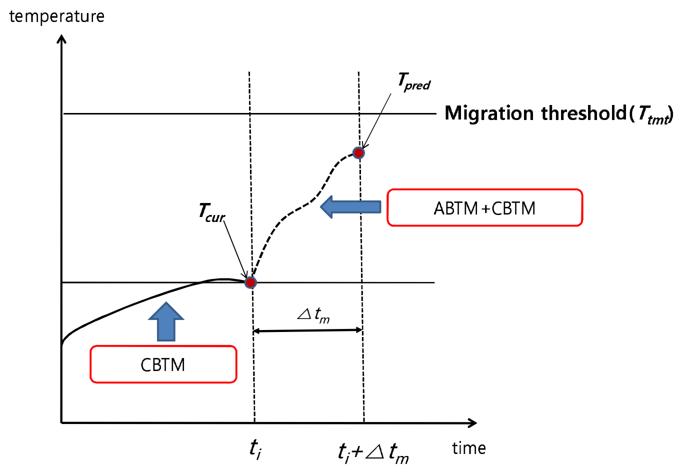


Fig. 20. PDTM utilizes ABTM and CBTM simultaneously to predict both short-term and long-term future temperature for multicore

the predicted temperature exceeds the migration threshold (T_{tmt}). When the current temperature (T_{cur}) reaches the temperature trigger threshold (T_{ttt}), Δt_m, the time to which the migration threshold, is calculated by ABTM. PDTM begins to calculate the future temperature via ABTM and CBTM for other cores after Δt_m. The core with minimum value among future temperature ($T[]$) is selected as new core for migration. As shown in Fig. 20, the goal is to find the future coolest core after Δt_m with our prediction. If the prediction temperature, T_{pred} is also larger than priority scheduling temperature(T_{pst}), the priority of application should be adjusted as well as migration.

ABTM is capable of predicting the future temperature within the short-term by tracking the application's thermal behavior, and is recognized as a fine-grained scheme. However, since frequent local temperature differences significantly affect temperature predictions, this impairs the ability to predict thermal behaviors in the long run. On the other hand, although CBTM lacks the ability to track each application's thermal behavior, this model is capable of foreseeing long-term thermal behaviors by facilitating the workload constant and the current working temperature of a core. This makes CBTM to be recognized as a coarse-grained scheme. In sum-

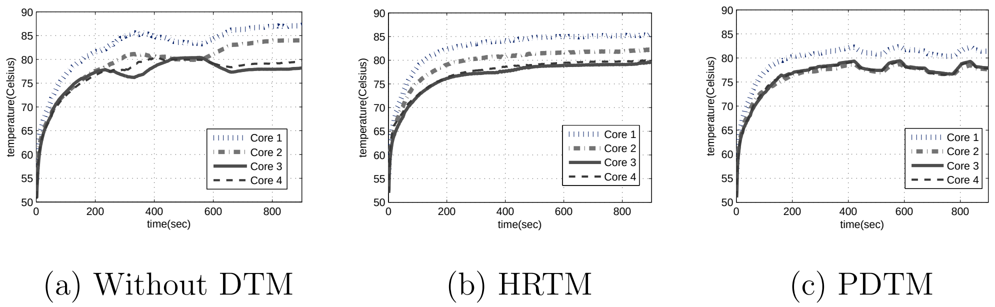


Fig. 21. Comparisons among without DTM, HRTM, and PDTM using `libquantum` benchmarks

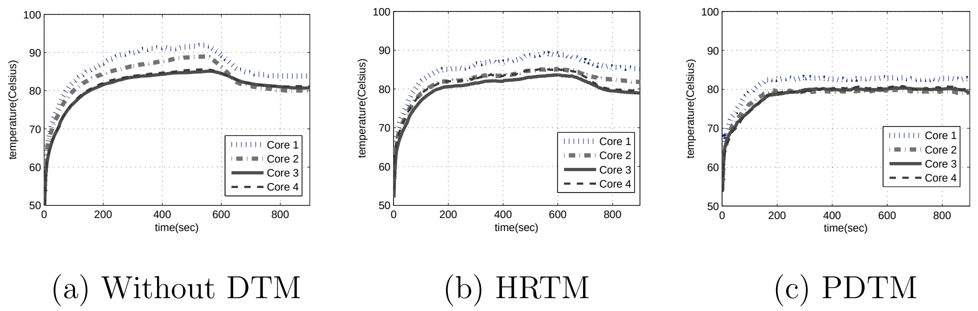


Fig. 22. Comparisons among without DTM, HRTM, and PDTM using `bzip2` and `libquantum` benchmarks

mary, PDTM takes the advantages of ABTM and CBTM in order to predict local application thermal behaviors as well as tracking the core behavior. The accuracy of our prediction model help determine the future coolest core for migration.

C. Experimental Results and Analysis

In order to estimate working temperature through Digital Thermal Sensor (DTS) for multicore systems, we develop a specific driver to access them in runtime. In CMPs silicon die, each core has a unique thermal sensor that triggers independently. The

Table IV. Environments parameters

Parameters	values (° Celsius)
Initial Temperature	54
Trigger Threshold	60
Migration Threshold	70
Priority Scheduling Threshold	82

Table V. A set of benchmarks list

Benchmarks	Temperature	Memory Usage
`perlbench+hmmer`	Low	Low
`perlbench+bzip2`	Low	High
`libquantum+hmmer`	High	Low
`libquantum+bzip2`	High	High

trigger point of these thermal sensors is not programmable by software since it is set during the fabrication of the processor [19]. In the experiments, we set temperature trigger threshold as 60°C to start PDTM, and the migration threshold as 70 °C to migrate applications when the predicted temperature exceeds the migration threshold. Also, priority scheduling threshold is 82 °C. When predicted temperature is reached at priority scheduling threshold, the priority of application can be adjusted as lower value. Our implementation parameters are provided in Table IV. All experiments are tested under ambient temperature control and fixed fan speed.

1. Digital thermal sensor for Intel quad-core

In Intel's Core Architecture, the DTS can be accessed by a Machine Specific Register (MSR). The value in the MSR is an unsigned number and the unit is Celsius (°C).

In MSR, we use `IA32_THERM_STATUS` register in order to get temperature of each core. Within the register, it uses 7 bits where the value of DTS is stored. We can get temperature for four cores by Equation (5.12).

$$T_{core} = T_{junction} - DTS_{value} \tag{5.12}$$

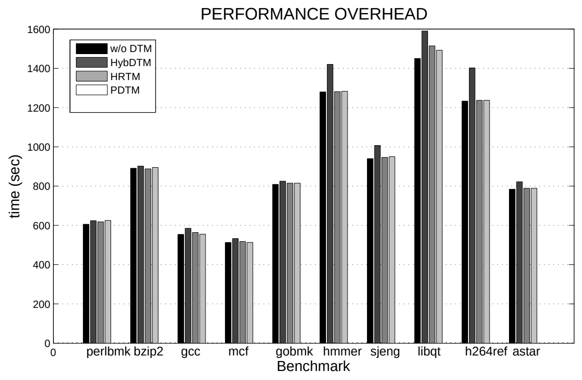


Fig. 23. Performance overhead : PDTM incurs only under 1% performance overhead in average while running single benchmark

$T_{junction}$ is a manufactural value by Intel.

2. Experimental results and analysis

To demonstrate PDTM, we conduct the experiments with a single SPEC2006 benchmark and a set of two SPEC2006 benchmarks as shown in Table V. Running the single benchmark, the presented PTDM can decrease 8% temperature in average (Fig. 21), and reduces up to 5 °C in the peak temperature with only under 1% performance overhead compared to Linux standard scheduler without DTM, as shown in Fig. 23. Running two benchmarks simultaneously, PDTM can even lower about 10% temperature in average and reduces up to 3 °C in peak temperature while running a set of benchmarks with only under 8% performance overhead compared to Linux standard scheduler without DTM, as shown in Fig. 22. It means PDTM can be more effective to control temperature than Linux standard scheduler when temperature and workload is higher.

In order to verify our scheme, we also rebuilt HybDTM [26] (the software scheme-changing priority) and HRTM [17] on Quad-Core system. HybDTM uses the priority-based scheme and HRTM uses the migration-based scheme. HybDTM scheme relies on the hardware performance counter, while HRTM relies on the current temperature information. The experimental results show that PDTM outperforms HRTM in reducing average temperature by about 7%, performance overhead by 0.15%, and the peak temperature by about 3 °C. In addition, the future temperature prediction model provides more accurate prediction with only less than 1.6% error; on the other hand, the estimation model, introduced in HybDTM, has at most 5% average error. The main reason of the accuracy in the prediction model is that we consider not only the core-based temperature at each core, but also the application thermal behavior. Therefore, PDTM is capable to manage the temperature fairness and control the overall temperature lower than other schemes even in the CPU-intensive situation.

D. Conclusions

In this work, we propose the Predictive Dynamic Thermal Management (PDTM) with an advanced future temperature prediction model for multicore systems, and implement PDTM on Intel Quad-Core with a specific device driver to access the Digital Thermal Sensor. We demonstrate that our scheme is able to reduce the overall temperature and provide thermal fairness among four cores. The proposed temperature prediction model can provide more accurate prediction and more efficient temperature management by using ABTM and CBTM with lower performance overhead compared to other schemes (HRTM and HybDTM). Most importantly, there is no additional hardware unit required for our prediction models and scheduler.

CHAPTER VI

TEMPERATURE-AWARE SCHEDULER BASED ON THERMAL BEHAVIOR GROUPING IN MULTICORE SYSTEMS

While manufacturing technology continues to improve reducing the size of packages, the physical limits of semiconductor-based microelectronics have become a major design concern. Due to the demand of more capable microprocessors, some methods, such as instruction-level parallelism (ILP) and thread level parallelism (TLP), have been proposed and employed in the modern processors. Moreover, multiple independent CPUs become a common solution to increase the system's overall TLP in the current market. A combination of increased available space due to the refined manufacturing processes and the demand for increased TLP is the logic behind the creation of Chip multiprocessors (CMPs).

Instead of pushing the limits of processor's frequency, the demand for more capable microprocessors must be satisfied by other methods. However, due to the decreased chip size and increased power-density, the power has been converted into significant heat and threaten the system performance, reliability, and even increased the power leakage. The great heat dissipation is pushing the limits of current packaging technology and cooling solution. Packages are designed for worst typical behavior and rely on Dynamic Thermal Management (DTM) techniques to control the temperature. Therefore, the chip design trend has been shifted to provide better power-efficiency, lower power-density, and more effective thermal management in the recent decade.

In this work, we propose a proactive thermal-aware scheduler (TAS) that exploits this variability in the context of multicore systems. TAS scheme utilizes an advanced future temperature prediction model for each core to estimate different thermal behaviors and measure the time duration before each core reaching the desired temperature threshold. Therefore, the appropriate measurements would be triggered to avoid thermal emergency based on the measured results. Although the proactive schemes have be proposed in [4, 5], the scheme in [4] is only applicable to the context of multimedia applications, since it predicts the temperature of the next frame based on the profile of the past frames; on the contrary, the scheme in [5] fails to consider the difference of temperature increasing pattern in different cores, because the steady state temperature and thermal parameter b are impractically assumed to be the same in each core within a single chip. Most importantly, in [5], the authors propose to migrate tasks from a potentially overheated core to the future coolest core based on the temperature prediction results. However, we believe that the target core of task migration should be determined by the core which needs longest time period to reach the predefined temperature threshold, because the temperature of the coolest can be increased faster than others due to the thermal correlation effects and its own thermal increasing pattern.

Therefore, we propose a simple and accurate prediction model to profile the application's thermal behavior and classify them into several groups offline, and then measure the time duration before reaching the desired temperature threshold for each core. The proposed temperature-aware scheduler is scalable to any current multicore model and architecture with on-chip thermal sensors that can be accessed at the software level. Eventually, we exploit and implement the advanced future temperature prediction model with the TAS strategy in the Intel Quad-Core Q6600 system. Experiments were conducted under CPU-intensive SPEC CPU 2006 benchmark, TAS maintains the system temperature below a given threshold by the proposed prediction model. Moreover, we demonstrate that TAS scheme based on simple parameters can control the tradeoffs between throughput and thermal fairness. Compared to tradi-

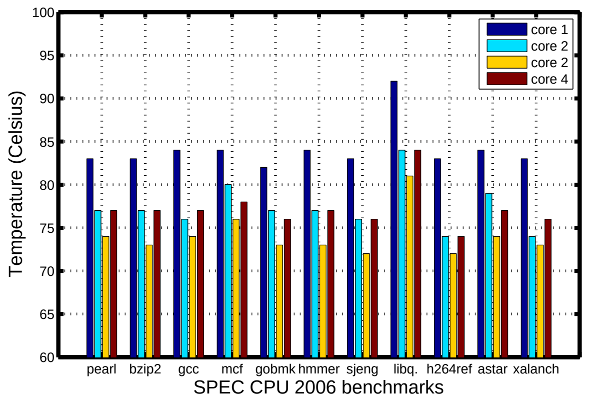


Fig. 24. T_{ss} according to SPEC CPU 2006 benchmark suite

tional schedulers employed in conjunction with DTM techniques, the temperature-aware scheduler can achieve higher throughput while maintaining QoS guarantees for soft real-time tasks with marginal loss in fairness among the best-effort tasks.

The main contributions of this work are summarized as follows:

- We classify the applications' thermal behavior groups using K-means clustering method with the steady state temperature.

- We propose an efficient temperature-aware scheduler in multicore systems and implement it in Intel Quad-Core Q6600 and two Quad-Core Intel Xeon E5310 processors systems. We demonstrate that our scheme is able to successfully reduce the overall temperature and provides the thermal fairness among cores.

- Most importantly, there is no additional hardware unit required for our temperature-aware scheduler. Our scheme is applicable to any multicore environment in real-world CMP products seamlessly.

A. Thermal Behavior Group

In this section, we propose how to classify the thermal behavior group by T_{ss}. Also, we introduce how to predict the future temperature and the time duration before reaching the predefined threshold using the thermal parameter b and the thermal behavior groups. we discuss about the advanced future temperature prediction model for each core to estimate the different application thermal behaviors and measure the time duration before each core reaching the desired temperature threshold. Based on the prediction results, the appropriate measures are triggered to avoid thermal emergency. Instead of being reactive to the current temperature, the temperature control techniques should be triggered if the core is predicted to be overheated in the near future to more effectively control temperature under the desired temperature threshold. Although the proactive schemes have been proposed in [4, 5], the scheme in [4] is only applicable to the context of multimedia applications, since it predicts the temperature of the next frame based on the profile of the past frames; on the contrary, the scheme in [5] fails to consider the difference of temperature increasing pattern in different cores, because the steady state temperature and the thermal parameter b are impractically assumed to be the same in each core within a single chip. Most importantly, in [5], the authors propose to migrate tasks from a potentially overheated core to the future coolest core based on the temperature prediction results. However, we believe that the target core of task migration should be determined by the core which needs longest time period to reach the predefined temperature threshold, because the temperature of the coolest can be increased faster than others due to the thermal correlation effects and its own thermal increasing pattern.

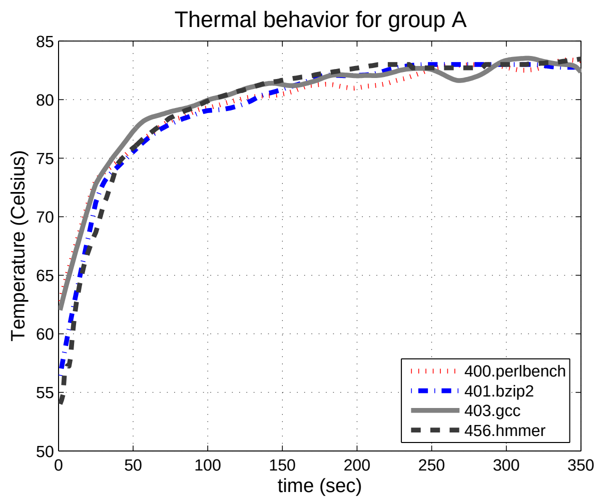
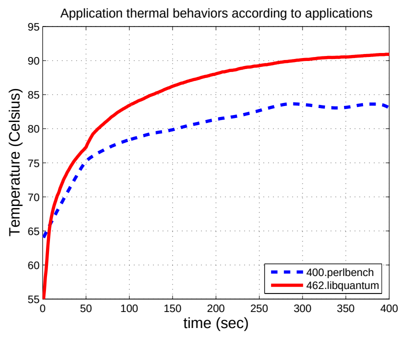


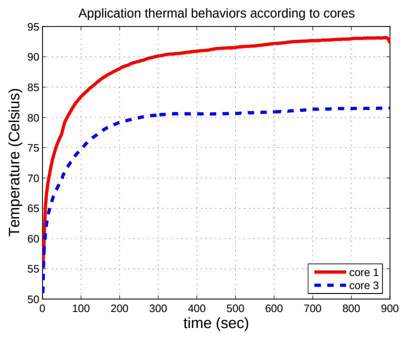
Fig. 25. Thermal behavior for Group A

1. Thermal behavior groups based on the applications' thermal pattern

As shown in Fig. 24, T_{ss} of each benchmark suite is different from each other, although all of their CPU utilizations are almost 100%. In order to manage temperature at runtime, the accurate applications' thermal behavior should be necessary. We observe T_{ss} and the thermal parameter b, T_{ss} value is more sensitive than the thermal parameter b to different thermal behaviors of applications. As shown in Fig. 25, the applications' thermal patterns are similar if their T_{ss} are analogous. In this research, we classify SPEC CPU 2006 benchmark applications with T_{ss} value as several thermal behavior groups using a K-means clustering method. The K-means clustering method is an algorithm to cluster n objects based on attributes into k partitions, $k < n$. It is similar to the expectation-maximization algorithm for mixtures of Gaussians in that they both attempt to find the centers of natural clusters in the data. It assumes that the object attributes form a vector space. The objective it tries to achieve is to



(a) Running two applications on the same core



(b) Running one application on the different cores

Fig. 26. The application thermal behavior according to applications and cores

minimize the total intra-cluster variance, or, the squared error function as follow:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2 \tag{6.1}$$

where there are k clusters S_i, $i = 1, 2, ..., k$, and μ_i is the centroid or mean point of all the points $x_j \in S_i$. As our preliminary experiments for eleven SPEC CPU 2006 benchmarks, $k = 5$ is the optimal value to classify applications as thermal behavior group as shown in Table VI.

For example, `400.perlbench`, `401.bzip2`, `403.gcc`, and `456.hmmer` applications can be classified as the same group (Group A) that has a similar thermal pattern and T_{ss}, as shown in Fig. 25. As our preliminary results, each T_{init} in Group A is different in four applications, but T_{init} cannot affect application's thermal pattern and their T_{ss}.

Table VI. The result of thermal behavior group using K-means clustering on 4-core system

SPEC CPU Applications	Core 1 T_{ss}	Core 2 T_{ss}	Core 3 T_{ss}	Core 4 T_{ss}	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	**GROUP**
400.*perlbench*	83 °C	77 °C	74 °C	77 °C	1	3	2	5	3	**A**
401.*bzip*2	83 °C	77 °C	73 °C	77 °C	1	1	2	5	3	**A**
403.*gcc*	84 °C	76 °C	74 °C	77 °C	1	1	2	5	3	**A**
429.*mcf*	84 °C	80 °C	76 °C	78 °C	1	1	1	3	4	**D**
445.*gobmk*	82 °C	77 °C	73 °C	76 °C	1	1	2	1	6	**C**
456.*hmmer*	84 °C	77 °C	73 °C	77 °C	1	1	2	5	3	**A**
458.*sjeng*	83 °C	76 °C	72 °C	76 °C	1	3	2	1	6	**C**
462.*libquantum*	92 °C	84 °C	81 °C	84 °C	2	3	2	4	1	**E**
464.*h*264*ref*	83 °C	74 °C	72 °C	74 °C	1	3	4	2	5	**B**
473.*astar*	84 °C	79 °C	74 °C	77 °C	1	1	1	3	4	**D**
483.*xalanchbmk*	83 °C	74 °C	73 °C	76 °C	1	4	2	2	2	**B**

2. The region of the thermal behavior group

By the previous thermal equations [30], we obtain

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \cdot e^{-bt} \tag{6.2}$$

Using Equation (6.2) and our measurements, we can obtain T_{ss} and b using the following steps:

1. We first run each SPEC CPU 2006 benchmark suite on each core until the temperature is not changed anymore to obtain the respective steady state temperature.
2. Then, we calculate the thermal parameter b by accessing the real temperature from the Digital Thermal Sensor (DTS) within a core using Equation (6.2).

Using Equation (6.2), we calculate thermal parameter, b, as shown in Equation (6.3).

$$b = -\frac{\log \frac{T_{ss}-T(t)}{T_{ss}-T_{init}}}{t}, \tag{6.3}$$

where T_{ss} is the steady state temperature, T_{init} is the initial temperature, and $T(t)$ is the current temperature at time t. As a result in Fig. 26(a), the thermal curve is different according to which application works. Moreover, even though the same application is running, the thermal pattern is also different according to which core is used as in Fig. 26(b).

While we exploit the steady state temperature value, T_{ss}, for clustering, we need to find another metrics for the classification of a new application at runtime. Since the T_{ss} value is not available before reaching the steady state, we use only the measured temperatures while applications running. In our observation, the thermal pattern can

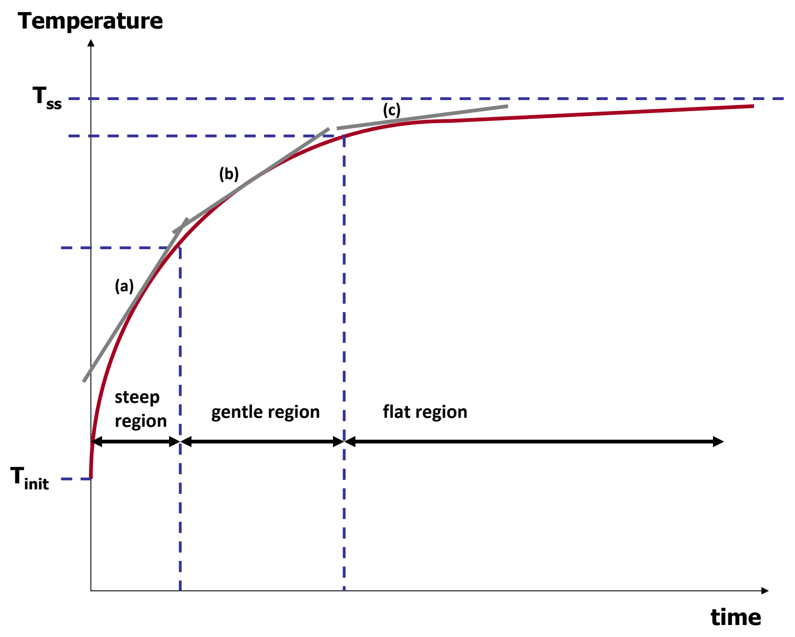


Fig. 27. Slopes for the thermal pattern at runtime

be divided by three regions as shown in Fig. 27. Each region has a different thermal slope that can affect the temperature increasing rate at a given time.

As shown in Fig. 27, the slope (a), (b), and (c) are different according to the time t. To calculate the slope for operating temperature, we can use a simple equations as follows:

$$S_i = \frac{T(i + \Delta t) - T(i)}{\Delta t}, \tag{6.4}$$

where S_i is the slope of the application's thermal pattern for ith region, $T(i)$ is the previous temperature, and $T(i + \Delta t)$ is the current temperature. Also Δt is a predefined time interval. Using the current temperature and the slope value, we can estimate an application's current region at runtime. As mentioned above, applications in the same thermal group have similar the thermal pattern, and the slope of regions in the same group is also similar. Based on those slope values and the thermal behavior groups, a temperature-aware scheduler estimates more accurate future temperature, and provide more effective dynamic thermal management.

B. Temperature-Aware Scheduler for Multicore Systems

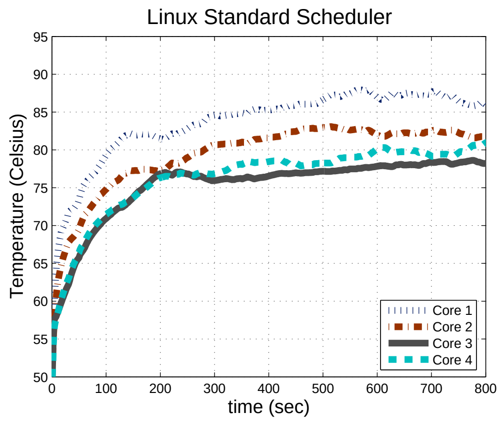
Since Linux standard scheduler is not aware of operating temperature for cores in multicore environments, we propose a temperature-aware scheduler that exploits this variability in the context of multicore systems. Although the proactive schemes have been proposed in [5], the scheme in [5] fails to consider the difference of temperature increasing pattern in different cores, because T_{ss} and the thermal parameter b are impractically assumed to be the same in each core within a single chip. Most importantly, in [5], the authors propose to migrate tasks from a potentially overheated core to the future coolest core based on the temperature prediction results. However, we believe that the target core of task migration should be determined by the core which needs the longest time period to reach the predefined temperature threshold, because temperature of the coolest can be increased faster than others due to the thermal correlation effects and applications' thermal behaviors. Each core's and the application's thermal behavior is different by T_{ss} and the thermal parameter b. Therefore, the migrating task from a potential overheated core to the coolest core is unnecessary and improper.

As shown in algorithm (3), we first profile T_{ss} of applications offline, and then classify them as thermal behavior groups using K-means clustering method. Whenever an application starts to run, the slope of the application is calculated after triggering a start threshold. According to an execution time, t, it is possible that this application can be classified into which thermal behavior group. As a result, we acquire T_{ss} and thermal parameter b for the application from this thermal behavior group at runtime. Based on these T_{ss} and thermal parameter b, a temperature-aware scheduler starts to predict each core's future temperature and the time period before reaching the desired temperature threshold. If the estimated time period is less than

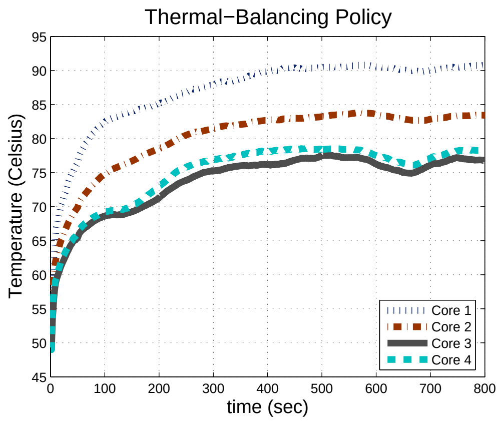
Algorithm 3 Temperature-Aware Scheduler based on Grouping for Multicore Systems

```
1: Classification $T_{ss}$ into the thermal behavior group by K-means clustering
2: $T_{cur} \leftarrow$ Access($DTS_Temperature$)
3: $slope_t \leftarrow$ Calculate($T_{cur}$, $T_{prev}$) at the time, $t$
4: $Thermal\_Group_i \leftarrow$ Find($slope_t$, $t$) for $application_i$
5: $T_{ss} \leftarrow$ Get($Thermal\_Group_i$)
6: $b \leftarrow$ Get($Thermal\_Group_i$)
7: for $T_{cur} \geq T_{gate}$ do
8:   for $j = 1$ to $MAX_{cores}$ do
9:     Calculate $timeperiod_{est}$ in Current_CORE
10:      if $time\_period_{est} \leq 2$ sec then
11:        Target_Core ← Longest_time_CORE($T[]$)
12:        MIGRATION($process_i \rightarrow$ Target_Core)
13:      end if
14:   end for
15: end for
```

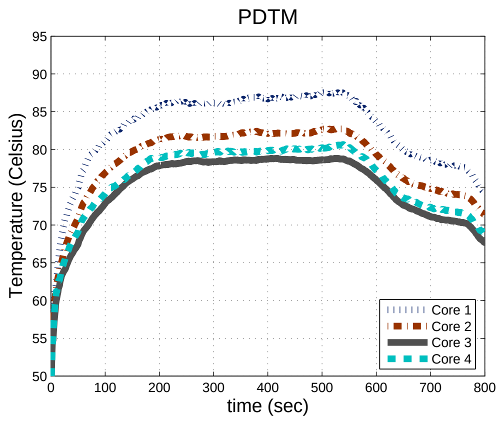
2 seconds, it means the cores are going to be overheated in the near future, and the task migration should be triggered. The migration target core is determined by which other core needs the longest time to reach the desired temperature threshold. Then, all the tasks on the potentially overheated core can be migrated to the target core to balance the heat within multicore environments.



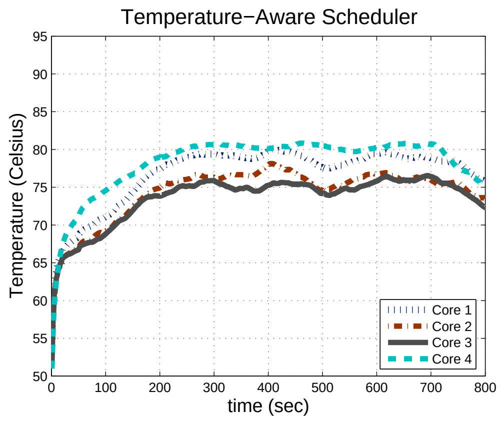
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



(d) Temperature-Aware Scheduler

Fig. 28. DTM evaluations in 4-core system using test group 2 (*bzip2* + *libquantum*)

Table VII. Experimental systems descriptions

	System I	System II
Cores	4 cores	8 cores
Processor	Intel Quad-Core Q6600	two Intel Quad-Core Xeon E5310
Memory	1 GB	1 GB
OS	SUSE 10.3	RedHat Enterprise 4

C. Experimental Results and Analysis

In order to demonstrate the applicability of our temperature-aware scheduler for various applications, we utilize several thermal behavior groups classified by K-means clustering method. In this work, we used twelve applications in SPEC CPU 2006 benchmark suite for profiling. In our experiments, we choose *bzip2* and *libquantum* in SPEC CPU 2006 benchmark, *vacation* from STAMP benchmark [39]. We select *bzip2* and *libquantum* because they are CPU-intensive. Also, *vacation* is a client/server travel reservation system benchmark that is appropriate to present the demand of thermal control in the server systems.

To compare with other schemes, we also rebuild the Predictive Dynamic Thermal Management (PDTM) [5] and Thermal Balancing Policy (TBP) [6] in our systems. All experiments in this work is under the ambient temperature control, and the speed of cooling fan is also fixed.

1. 4-core system

To verify a temperature-aware scheduler for 4-core system, two applications run simultaneously. As shown in Fig. 28, compared to Linux Standard Scheduler, a temperature-aware scheduler reduces peak temperature up to 8 °C. However, PDTM reduce the peak temperature by 2 °C, while TBP is increased by 2 °C. For the performance overhead evaluation, the temperature-aware scheduler presents less than 12% performance overhead, and PDTM has 8% compared to the Linux Standard Scheduler, while TBP incurs 35%. Since the temperature-aware scheduler finds the longest core instead of the coolest core under thermal threshold, our scheme reduces the number of migration while providing better thermal-balancing for cores compared to other DTMs. Therefore, the temperature-aware scheduler based on thermal behavior grouping provides the better effectiveness in temperature control for multicore systems.

2. 8-core system

In 8-core system, the temperature-aware scheduler outperforms PDTM and TBP in both temperature control effectiveness and efficiency. The temperature-aware scheduler reduces peak temperature by 5 °C with 7.52% performance overhead compared to Linux Standard Scheduler, while PDTM and TBP reduce peak temperature by 4 °C and 13.2% performance overhead and 2 °C and 35% performance overhead, respectively. Although TBP also decreases the peak temperature and presents smoother thermal pattern compared to Linux Standard Scheduler, TBP causes impractically huge performance overhead. Moreover, the exchanged threads cannot effectively reduce core 1's temperature. Since PDTM is not aware of the different thermal effects contributed by running applications, PDTM cannot accurately predict future temperature and react in time. Therefore, PDTM fails to control the temperature under the desired level.

D. Conclusions

In this work, we propose a temperature-aware scheduler based on thermal behavior grouping in multicore systems. To classify applications according to the thermal behavior, we use T_{ss} value as a classification feature in K-means clustering method. We observe that among thermal parameter b and T_{ss}, T_{ss} is more proper to explain the application's thermal pattern. The proposed temperature-aware scheduler finds a core which takes the longest time to reach a temperature threshold instead of the coolest core for process migrations. To verify the temperature-aware scheduler, we implement it on two multicore systems such as a 4-core (Intel Quad-Core Q6600) and 8-core (two Quad-Core Intel Xeon E5310 processors) systems. We demonstrate that the temperature-aware scheduler is able to reduce the overall temperature and provide the thermal fairness among cores. Also, the temperature-aware scheduler can provide more accurate prediction and more efficient temperature management by using the thermal behavior grouping and the method to find the longest core with lower performance overhead compared to other schemes such as Linux Standard Scheduler, Thermal-Balancing Policy, and Predictive DTM.

CHAPTER VII

A THERMAL MODEL BASED ON WORKLOAD CHARACTERISTICS USING CDF

In this work, we propose a Proactive Correlation-Aware Thermal Management (Pro-CATM) that incorporates three main components: a representative workload estimation, a future temperature estimation model and a thermal-aware thread scheduler. The representative workload estimation utilizes the workload probability distribution to measure each running thread-level workload behavior locally and core-level workload behavior within each core globally. The representative workload is estimated using the cumulative distribution function (*cdf*) at runtime. Thus, the thermal impacts contributed by various threads are distinguished by the estimated representative workload. We further model the thermal correlation by profiling the thermal impacts from neighboring cores under the specific workload. Once the thermal behavior of each running thread is obtained and the thermal correlation is modeled for the neighbor cores, the future temperature estimation model can then estimate each core's future temperature by taking both the thermal behaviors and the thermal correlation into account. Therefore, based on the estimated future temperatures, the thermal-aware thread scheduler moves the running thread from the possible overheated core to the future coolest core (migration), or reduce the processor resources (priority scheduling) while migration is not possible within multicore systems to avoid thermal emergency and provide thermal fairness with negligible performance overhead.

A. A Representative Workload Estimation Based on CDF

In this section, we introduce a statistical model to estimate workload. To capture the dynamic workload change, first we define workload with an execution time information for a given time inverval, then we model a representative workload through a cumulative distribution function (*cdf*) and standard deviation based on workload history information.

1. The definition of workload

An application consists of a sequence of instructions to be executed. Execution time (t_{app}) of the application can be represented in terms of Cycles Per Instruction (CPI), the number of instructions being executed, and the CPU frequency as follows: [40]:

$$t_{app} = \frac{IC \cdot CPI}{f_c}, \tag{7.1}$$

where IC is the dynamic instruction count, CPI is the average number of cycles per instruction, and f_c is the operating frequency. Therefore, we can define workload (W_{app}) by the execution time of the application, t_{app}. Although Linux kernel provides CPU utilization, we exploit Performance Monitoring Counters (PMC) [34] to measure t_{app} more accurately in our experiments.

2. The statistical representative to estimate workload

Instead of using simple average of W_{app}, we attempt to use a representative workload that can capture the system dynamics at runtime. In this study, we propose to derive the representative workload from a cumulative distribution function (*cdf*) of W_{app} and its standard deviation. We denote the *cdf* as $F(x)$ for a random variable X for W_{app} according to a probability density function (*pdf*), $f(x)$, and probability p using Equation (7.2)

$$P(W^{min} \leq X \leq W^{max}) = \int_{W^{min}}^{W^{max}} F(x)dx, \tag{7.2}$$

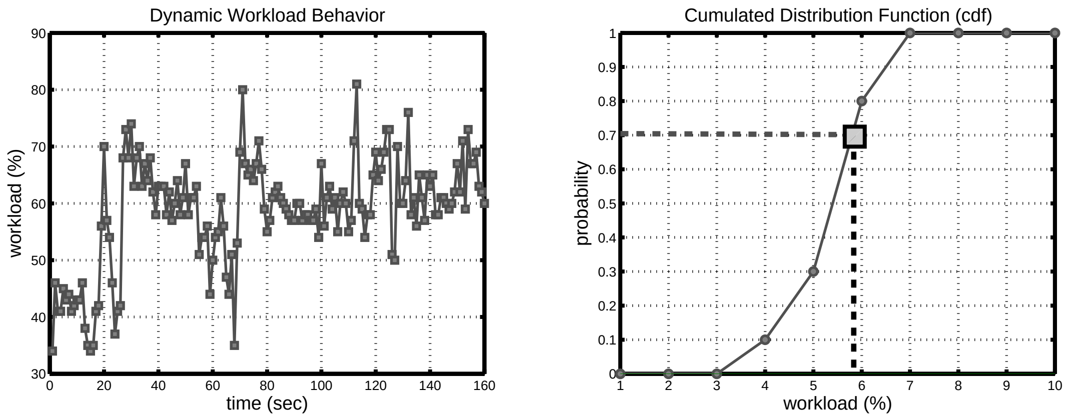
where X is in the interval $[W^{min}, W^{max}]$ and $F(x) = P(X \leq x)$. And W^{min} implies that there is no workload in the system and W^{max} implies 100% workload, respectively. To satisfy a various computational requirements, the representative should be decided by the probability requirements for application workload, W_{app}, in a given time period. Specifically, let ρ be the probability required for application workload in a given time period. In our observations, even dynamic workload of applications can be defined as the representative workload by a probability ρ as follow:

$$P[X \leq W_{app}] \geq \rho. \tag{7.3}$$

As shown in Fig. 29, we can exploit the representative workload using *cdf* when playing a multimedia application. Moreover, in order to distinguish the threads with stable workload behaviors from those with highly unstable workload behaviors, the standard deviation, denoted as σ, is considered. In this study, we classify the threads with σ less than 7.0 as the threads with stable workload behaviors in our systems. Therefore, we use $\rho = 0.5$ to represent these stable threads' workload and $\rho = 0.7$ to represent those threads with highly unstable workload behaviors for the thermal safety in the *cdf*.

$$\rho = \begin{cases} 0.5 & if\ \sigma < 7.0, \text{ stable workload,} \\ 0.7 & if\ \sigma \geq 7.0, \text{ dynamic workload} \end{cases}. \tag{7.4}$$

Here, we consider the thread-level workload estimation as local, while core-level workload estimation as global. For global workload estimation, the overall workload in a single core is also monitored at runtime. The same as the thread-level, the concepts of σ and ρ are also adopted in the core-level. Thus, the representative workload for each core will be used to estimate the future temperature as explained in the next section, and the different thermal effects contributed by different threads could



(a) Dynamic workload behaviors (b) The representative workload calculated by *cdf*

Fig. 29. The representative workload is 58% when the probability (ρ) is 0.7 in dynamic workload behavior

also be distinguished by the representative workloads if there are multiple threads running in a single core. Moreover, to effectively control the temperature with less performance overhead, we set 30% as the workload threshold. That implies that the ProCATM would only control those threads with workloads higher than 30%, because the threads with workloads under 30% only affect the temperature at most 2 °C in our systems.

3. Thermal parameters in CMP systems

In order to provide a thermal model of a processor, we should consider the relationship between temperature and workload of applications on the processor. By modeling the power dissipation, more precise models can be derived from a simple model [30]. We analyze Fourier's Law of heat conduction where the formula states that the rate of heating or cooling is proportional to the difference in temperature between the object and the environment [30]. We define $T(t)$ and $P(t)$ as temperature and power at time

t, respectively. Then we can use the Fourier's Law as the following [41, 29]:

$$T'(t) = P(t) - bT(t), \tag{7.5}$$

where b is a positive constant representing the power dissipation rate. Now, we define $f(t)$ as processor frequency at time t. Since the power consumption of a processor is an increasing convex function of the frequency, power consumption can be represented by frequency [41]. Most studies assume that power and processor frequency are relevant to the followings:

$$P(t) = a(f^{\alpha}(t)), \tag{7.6}$$

for some constant a and $\alpha > 1$. With an assumption that $T_0 = 0$ (the initial temperature is the ambient one), the solution of Equation (7.5) using Equation (7.6) can be presented as follows:

$$T(t) = \int_{t_0}^{t} P(\tau)e^{-b(t-\tau)}d\tau + T_0 e^{-b(t-t_0)}, \tag{7.7}$$

$$T(t) = \int_{t_0}^{t} a(fr^{\alpha}(\tau)e^{-b(t-\tau)})d\tau + T_0 e^{-b(t-t_0)}. \tag{7.8}$$

Then, for the variation of temperature, we deal with two cases of the variation at any point t [30]. First, the case that temperature is non-decreasing, by Equation (7.5) and Equation (7.6), can be derived like the following:

$$f(t) \geq (\frac{b(T(t))}{a})^{\frac{1}{\alpha}}. \tag{7.9}$$

Then, the case that temperature is non-increasing can be expressed like the follow.

$$fr(t) \leq (\frac{b(T(t))}{a})^{\frac{1}{\alpha}}. \tag{7.10}$$

Therefore, we can observe that scaling the frequency to change temperature can be performed for the desired direction. Finally, We can derive the following Equation if we maintain the frequency constant at $f(t) = f_c$ during the time interval at $[t_0, t]$.

$$T(t) = \frac{a(f_c^\alpha)}{b} + (T(t_0) - \frac{a(f_c^\alpha)}{b})e^{-b(t-t_0)}, \tag{7.11}$$

$$\frac{dT}{dt} = -b(T(t) - \frac{a(f_c^\alpha)}{b}). \tag{7.12}$$

where f_c is the current frequency on the processor. In order to determine thermal parameters, a and b, we assume $\alpha = 3.0$ [41], and then we can obtain the values for a and b. Although the values of a and b are processor-specific, b is more relative to application's workload at runtime. Because b can be affected by executed cycles for applications. When we run an application infinitely with the maximum CPU utilization and observe the heating and cooling curves, thermal parameters can be determined by using Equation (7.11). After a sufficient time of execution in the maximum CPU utilization, the infinite steady-state temperature value $T(\infty) = T_{ss}$ can be observed. By setting $T(t) = T$ and $\frac{a(f_c^\alpha)}{b} = T_{ss}$, Equation (7.11) is transformed as follows:

$$T = T_{ss} + (T_{init} - T_{ss})e^{-bt}, \tag{7.13}$$

$$\frac{dT}{dt} = -b(T - T_{ss}), \tag{7.14}$$

where T_{init} is the initial temperature.

Using T_{ss} and sampling the temperature every millisecond, from Equation (7.14), the rate of increase, dT/dt, is plotted against $(T - T_{ss})$ at each point. The resulting set of points is fitted to a straight line using least mean square error fitting. From the Equation (7.14), the slope of this straight line represents the value of b. In order to measure a and b more accurately, we should know the meaning of those values. The change in temperature is based on individual component's thermal resistance and capacitance in specific processors [20]. To obtain current and future temperatures, we should take account for thermal resistance R_{th} and thermal capacitance C_{th}, while changing in temperature from T_{old} to T_{new} over a time interval Δt like Equation (7.15).

$$T_{new} = P \cdot R_{th} + (T_{old} - P \cdot R_{th}) e^{\frac{-\Delta t}{R_{th} \cdot C_{th}}}, \tag{7.15}$$

where R_{th} is thermal resistance and C_{th} is thermal capacitance. With Equation (7.15) and (7.11), we can derive the thermal parameters a and b as follows:

$$a = \frac{1}{C_{th}},\ b = \frac{1}{R_{th} \cdot C_{th}} \tag{7.16}$$

By Equation (7.16), the thermal parameter a is represented as thermal capacitance C_{th}. Thermal capacitance is defined as the amount of thermal energy required to raise temperature of one mole of material by 1 Kelvin and can be measured at constant volume or at constant pressure [29]. Therefore, this value is practically constant in the same material. In contrast, the thermal parameter b is related to an application's workload. This is because the thermal resistance is in inverse proportional to the power consumption. Hence, characterizing the workload behavior is critical for distinguishing the different threads' thermal effects. As shown in Fig. 30, the workload dominates the temperature change in a core. Therefore, it is important to characterize an application's workload in thermal control.

B. Thermal Mode Based on Workload

In this section, we propose a proper thermal model for CMPs to estimate future temperature considering thermal correlation among neighboring cores.

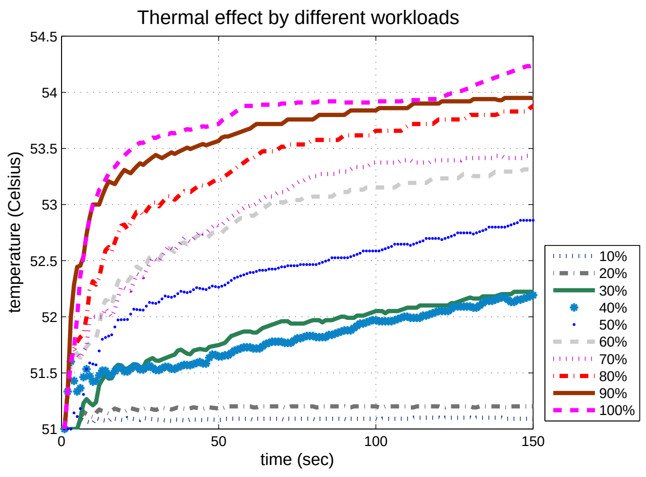


Fig. 30. Thermal effects by different workloads

Table VIII. Each core's respective T_{ss} and thermal parameter b for a generated example process with 100% workload running in the Intel Quad Core Q6600 system

	Core 1	Core 2	Core 3	Core 4
b	0.0199	0.0175	0.0169	0.0181
T_{ss}	78 °C	72 °C	68 °C	71 °C

Table IX. The thermal parameter b and T_{ss} according to workload in 4-core system

	Core 1		Core 2		Core 3		Core 4	
Workload (%)	b	T_{ss}	b	T_{ss}	b	T_{ss}	b	T_{ss}
20%	0.0139	59 °C	0.0092	58 °C	0.0053	52 °C	0.0065	57 °C
40%	0.015	64 °C	0.0058	62 °C	0.0085	57 °C	0.0065	58 °C
60%	0.0187	68 °C	0.0092	65 °C	0.0078	61 °C	0.0113	63 °C
80%	0.0179	73 °C	0.0164	70 °C	0.0165	67 °C	0.0138	68 °C
100%	0.0199	78 °C	0.0175	72 °C	0.0169	68 °C	0.0181	71 °C

1. Prior thermal model of a single core

The heat transfer equations are introduced to model the steady state temperature of systems with heat sources in [37]. With those heat transfer equations, Wang and Bettati present that the rate of temperature change is proportional to the difference between the current temperature and the steady state in [30]. Let T_{ss} be the steady state temperature of an application. Then, we denote $T(t)$ as the temperature at time t and T_{init} as the initial temperature when an application starts execution ($T(0)$=T_{init}). Thus,

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{7.17}$$

where b is a thermal parameter. Solving Equation (7.17) with $T(0) = T_{init}$ and $T(\infty)$ $= T_{ss}$, we can obtain

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{7.18}$$

Using Equation (7.18) and our measurements, we can obtain T_{ss} and b using following steps:

1. We first run an application with 100% workload for a long time, and then measure the steady sate temperature (T_{ss}) when temperature is not changed any more.

2. We calculate the thermal parameter b by measure temperature through Digital Thermal Sensor (DTS) within the core using Equation (7.18).

As the result, we obtain each core's respective value b and T_{ss} for the generated process in Table VIII by executing a generated process with 100% workload in each core individually. Therefore, once the thermal parameter b and the steady state temperature are obtained, we can estimate the core's future temperature ($T(t)$) after time t by Equation (7.18). We can notice that each core's thermal parameter b and

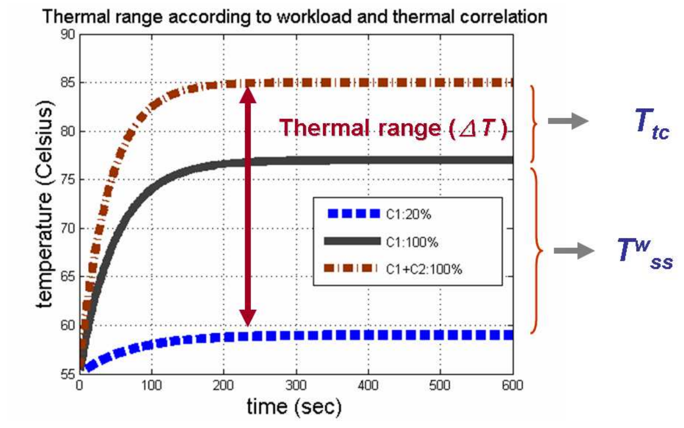


Fig. 31. The thermal range (ΔT) using T^w_{ss} and T_{tc} to calculate T'_{ss} for core 1

T_{ss} are different even though the cores are within the same package as shown in Table VIII. Moreover, we have observed that T_{ss} and thermal parameter b are different according to the workload in each core, as well as thermal correlation effect among neighboring cores in the CMP systems. Therefore, we are motivated to improve the prior thermal model by including the workload behavior and thermal correlation concepts.

2. The thermal impacts contributed by different workloads

In the real world applications, the workload is fluctuant, and each core's T_{ss} and b are changed by the variance of workload at runtime. Therefore, by running processes with several different workloads on each core, we observe the relationship between workload and thermal parameter b, as well as T_{ss} in Table IX.

3. New T'_{ss} according to thermal correlation

We classify new T'_{ss} into two parts: T^w_{ss} (according to its own workload), and T_{tc} (a thermal correlation affected by neighboring cores' temperature). Thus, we calculate new T'_{ss} according to own workload by the following Equation (7.19).

$$T'_{ss} = T^{w}_{ss} + T_{tc}. \tag{7.19}$$

First, we can obtain T^{w}_{ss} from Table IX. Since neighboring cores' temperature is relative to their own workloads, T_{tc} should also consider each cores' workloads as well as their temperature. As shown in Fig. 31, the thermal range of core 1 is determined by the thermal correlation effect from core 2, core 3, and core 4 in our 4-core system. In order to calculate the T'_{ss} for core 1, we develop Equation (7.20) to obtain T_{tc} to model the thermal correlation impact from other cores.

$$T_{tc} = \sum_{i=2}^{n} \Delta T \times W_i, \tag{7.20}$$

where ΔT is the thermal range between core 1's temperature with and without thermal correlation from neighboring cores. W_i is each core's representative workload estimated. For example, there are 4 threads with different workloads running on each core individually in the 4-core system (Core 1 : 100%, Core 2 : 50%, Core 3 : 30%, Core 4 : 20%). We first obtain T^{w}_{ss} as 78 °C and b as 0.0199 from Table IX. Then, we calculate the thermal correlation effect from each neighboring core with 100% workload, as shown in Table X.

Therefore, by Equation (7.19) and (7.20), the T'_{ss} can be obtained by the following:

Table X. T_{tc} and b according to thermal correlation profiled for core 1

	T_{tc}	b
Only $Core1$ (100%)	78 °C	0.0199
$Core1$ (100%) + $Core2$ (50%)	85 °C	0.0246
$Core1$ (100%) + $Core3$ (30%)	84 °C	0.0195
$Core1$ (100%) + $Core4$ (20%)	83 °C	0.0176

$$\begin{aligned} T'_{ss} &= 78 + (85 - 78) \times 50\% \\ &+ (84 - 78) \times 30\% \\ &+ (83 - 78) \times 20\% \\ &= 84.3\ (^\circ\ C) \end{aligned}$$

In above example, the calculated T'_{ss} (84.3 °C) is higher than the original T_{ss} (78 °C). The difference between these values represents thermal correlation effect, T_{tc} (6.3 °C), among neighboring cores.

4. New b' according to thermal correlation

Also, in order to advance the new b' by considering the thermal correlation effect, we define b' as $b' = b^w + b_{tc}$ and develop the following equations (7.21) and (7.22):

$$b_{tc} = \sum_{i=2}^{n} \Delta b \times W_i \tag{7.21}$$

$$b' = b^w + (b_{tc} \times \frac{(T'_{ss} - T_{cur})}{(T'_{ss} - T_{init})}), \tag{7.22}$$

where b^w is determined according to own workload and b_{tc} is thermal parameter affected by neighboring cores. And T_{cur} is current temperature and T_{init} is initial temperature. In Equation (7.21), Δb is the difference between core 1's thermal parameter b with and without thermal correlation by neighboring cores. In contrast with T'_{ss}, thermal parameter b' is changeable according to current temperature (T_{cur}). Therefore, even if the thermal parameter b' can be changed by current temperature and thermal correlation, b' determines only temperature increase rate.

5. Future temperature estimation model

In this section, we propose a new thermal model to estimate future temperature for each application in CMPs. We focus on obtaining both new T'_{ss} and new thermal parameter b' according to the estimated workload and profiled thermal correlation impacts.

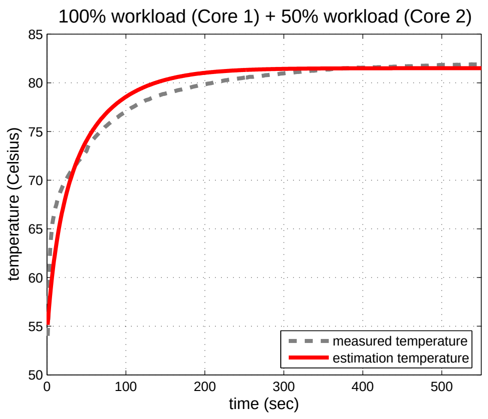
The original thermal models for estimating the future temperature at time t is improved from Equation (7.18) to the following Equation (7.23) for a specific core with workload estimation and thermal correlation by neighboring cores.

$$T'(t) = T'_{ss} - (T'_{ss} - T_{init}) \times e^{-b't}$$

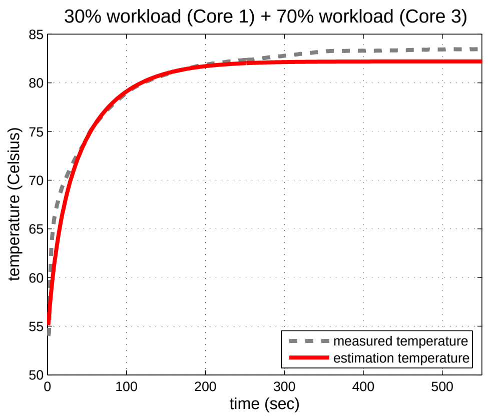
$$T'(t) = T^w_{ss} + T_{tc} - (T^w_{ss} + T_{tc} - T_{init}) \times e^{-(b^w_{tc} + (b_{tc} \times \frac{(T'_{ss} - T_{cur})}{(T'_{ss} - T_{init})})) \times t} \tag{7.23}$$

In order to validate our new thermal model, we conduct several experiments running some applications with different workload. The estimated future temperature for core 1 through our new thermal models are compared with the monitored temperature by the Digital Thermal Sensor in Fig. 32. As shown in Fig. 32, the estimated future temperature by the improved thermal models is very accurate, especially within the first 200 seconds, which is much longer than enough to react against to the increasing temperature.

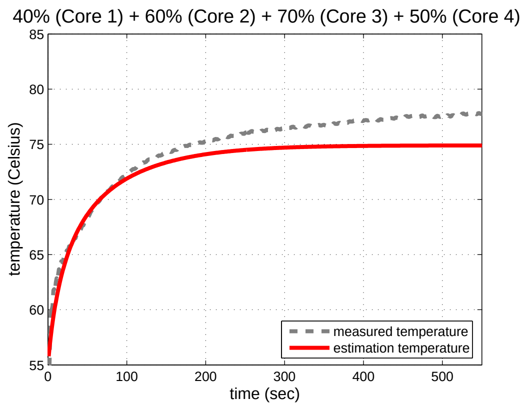
Moreover, in order to demonstrate that the improved thermal models can be effective even under the fluctuant workload, we also evaluate our thermal models by executing multimedia data, which generates two individual threads. We first calculate the representative workload through the cumulative distribution function (*cdf*), and then estimate temperature by considering both the representative workload and the thermal correlation in the equations above. The result of workload estimation by *cdf* is shown in Fig. 33(a), and the estimated temperatures compared with the monitored real temperature is shown in Fig. 33(b). Thus, the results also demonstrate the accuracy of our improved thermal models under fluctuant workloads considering both workload and thermal correlation from neighboring cores. Also, since thermal control based on current temperature may overheat by physical nature of temperature.as shown in Fig. 34(b). In order to overcome this problem, we propose thermal control based on future temperature as shown in Fig. 34(b). Therefore, the proposed Future Temperature Estimation Model estimates each core's future temperature (T_{est}) for its individual steady state temperature according to the application and core representative workloads ($W_{app\_rep}, W_{core\_rep}$) estimated by Representative Workload Estimation (RWE). The estimated future temperature is validated against the measured temperature for actual processors with Digital Thermal Sensors (DTS), with an average error of 2.4%. Eventually, the time duration (Δt) before the temperature reaches the migration threshold can be calculated and passed to TATS for thread control along with (T_{est}). (The detailed explanations will be brought in the following sections.) Therefore, instead of blindly migrating all the running threads or rescheduling all their priorities, the proposed ProCATM is able to adaptively cope the threads according to their different thermal effects, based on their representative workloads and neighboring thermal correlation effects. Consequently, ProCATM can



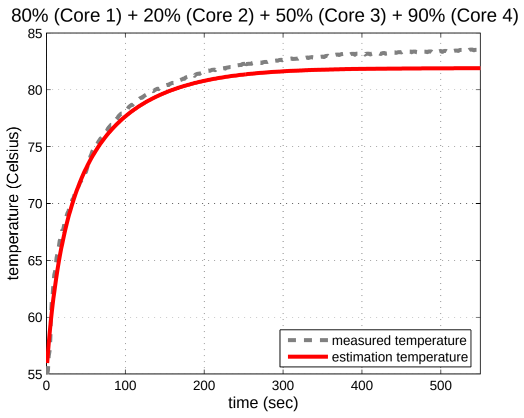
(a) 100% (Core 1) + 50% (Core 2)



(b) 30% (Core 1) + 70% (Core 3)

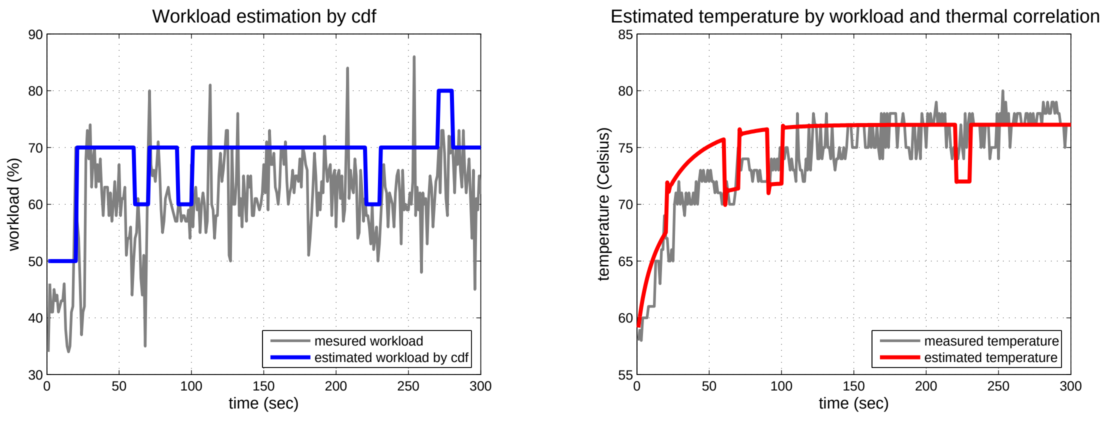


(c) variable workloads on all cores



(d) variable workloads on all cores

Fig. 32. Validation of improved thermal model with workload estimation and thermal correlation in static application. (Only core 1's temperature is drawn)



(a) Workload (b) Temperature

Fig. 33. Validation of new thermal model with fluctuant workload: whiling playing the *Transformer* movie, the Mplayer software would generate two threads. One is the X windows daemon with stable workload, and the other one is for decoding with fluctuant workload as shown above.

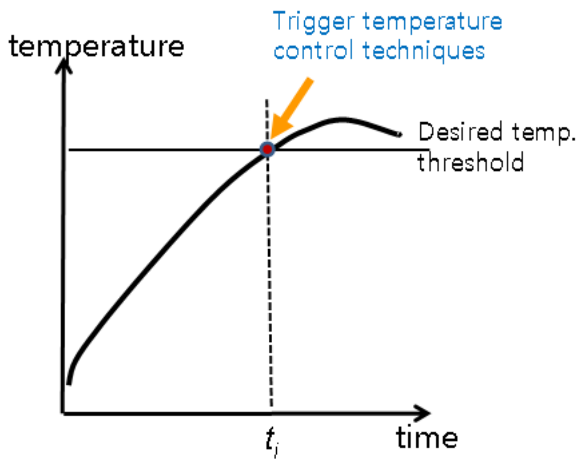
control the temperature at a desired level with negligible performance overhead.

C. A Proactive Correlation-aware Thermal Management

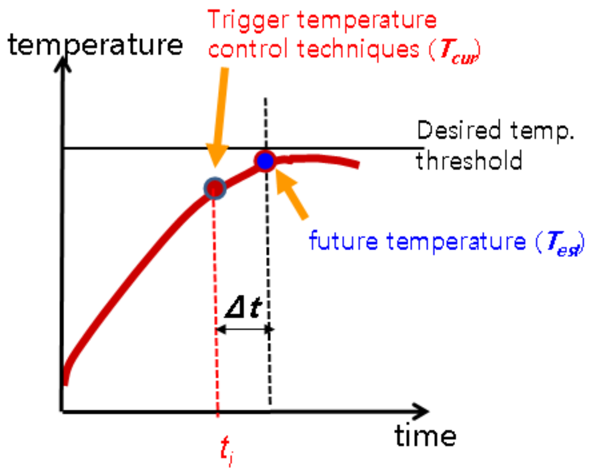
In this section, we introduce the system design and architecture of the proposed ProCATM. Moreover, we present how Thermal-Aware Thread Scheduler (TATS) utilizes the workload behavior and thermal correlation information to achieve thermal balancing and lower the peak temperature.

1. System overview

Basically, a Proactive Correlation-Aware Thermal Management (ProCATM) consists of three major components: Representative Workload Estimation (RWE), Future Temperature Estimation Model (FTEM) and Thermal-Aware Thread Scheduler



(a) Based on current temperature


temperature
Trigger temperature control techniques (T_{cur})
Desired temp. threshold
future temperature (T_{est})
Δt
t_i
time

(b) Based on future temperature

Fig. 34. The difference of thermal control based on current temperature and future temperature

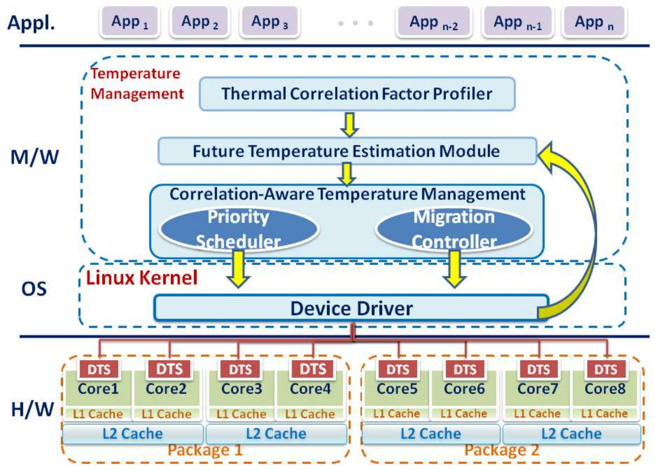


Fig. 35. ProCATM system architecture

(TATS). As shown in Fig. 35, we depict the system architecture on a 4-core (Intel Quad Core Q6600 processor) machine. We developed a specific device driver for Linux to access Digital Thermal Sensor (DTS) for monitoring each core's temperature, and temperature information would be used in the FTEM. As mentioned before, RWE is used to exploit the representative workload in both thread and core levels to present each application's workload behavior, while FTEM utilizes the representative workload and thermal correlation information to estimate the future temperature (T_{est}) and the time duration (Δt) before temperature reaches the migration threshold. Hence, the TATS is able to react against to the thermal emergency appropriately using the estimated information. In the following section, we discuss about the TATS in details.

2. Thermal-aware thread scheduler (TATS)

To guarantee the thermal safety, Thermal-Aware Thread Scheduler (TATS) consists of two schedulers: the priority scheduler and migration scheduler. Basically, when current temperature reaches the trigger threshold, the RWE starts to monitor the application's workload behavior and calculate the representative workloads through *cdf* for the running thread and core. Hence, the core representative workload (W_{core_rep}) and application representative workload (W_{app_rep}) can be utilized in the FTEM. In FTEM, the time duration (Δt) before reaching migration threshold can be estimated based on the profiled T'_{ss} and b' for different workloads. According to the Δt, TATS migrates the running threads from the possible overheated core to another core. Here, since a thread under 30% workload affects the core temperature at most 2 °C in our observations, TATS deals with the threads with workload higher than 30% to reduce the performance overhead. In TATS, migration can be adopted in most cases, unless all the cores' temperature reaches the priority scheduling threshold. In this case, TATS should utilize the priority scheduler to adjust the *nice* value in the Linux process scheduler to reduce the thread's priority and increase the cooling time, because migration cannot effectively reduce the core temperature if all the core temperatures are near the maximum allowable temperature. Also, we ignore the difference of performance overhead caused by migrating threads with different memory usages, because we observe that the migration performance overhead is dominated by the thread suspending and restarting processes in the Linux kernel, rather than the different memory usage. For example, by comparing the *libquantum* benchmark and a generated transaction thread, the difference of migration overhead is just 0.0346 millisecond, although both of them maintain almost 100% workload, but the generated transaction thread has about 51% memory usage in Linux kernel, while the *libquantum* has only around 3% memory usage.

Therefore, by considering the thermal effect of different workloads and the thermal correlation, TATS is able to effectively reduce the peak temperature of each core and achieve thermal balancing with ignorable performance overhead.

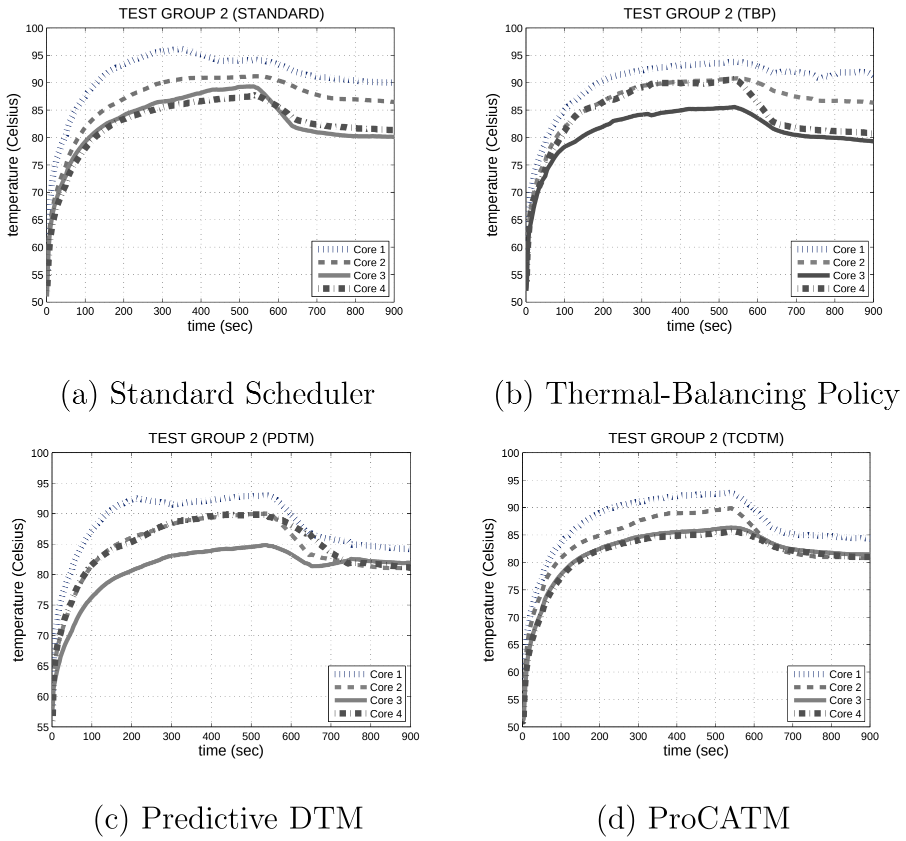


Fig. 36. DTM evaluation in Intel Quad Core Q6600 system for stable workload behaviors: *libquantum* + *vacation*

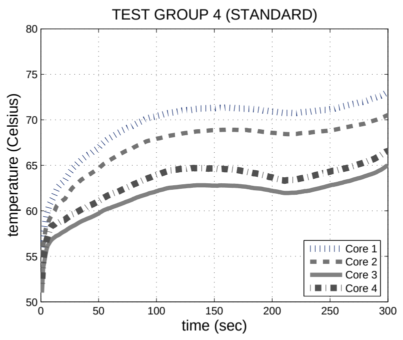
D. Experimental Results and Analysis

In this section, the detailed experimental environment and results would be brought, along with the analysis of the efficiency and effectiveness of the proposed ProCATM.

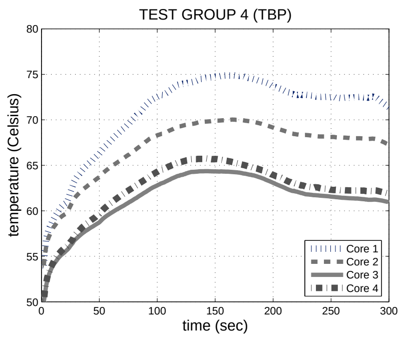
To compare the effectiveness and efficiency of the proposed ProCATM, we also rebuild the Predictive Dynamic Thermal Management (PDTM) [5] and Thermal Balancing Policy (TBP) [6] in our systems. All the experiments in this work are under ambient temperature control, and the speed of cooling fan is also fixed.

As shown in Fig. 36, all the DTMs have lower peak temperature compared to the Linux Standard Scheduler in 4-core system. Compared to the Linux Standard Scheduler, both ProCATM and PDTM reduce the peak temperature by 3.13%, while TBP is reduced by 2.08%. For the performance overhead evaluation, the ProCATM and PDTM present less than 0.46% performance overhead compared to the Linux Standard Scheduler, while TBP incurs 6.64%. This is also the reason why the temperature in TBP does not decrease obviously after executing 600 seconds. Since there are only two threads running in the system, the thermal correlation effect is minor. Moreover, both of the treads maintain 100% workload stably, and the difference of thermal behaviors can be ignored. Therefore, PDTM presents the similar effectiveness in thermal control compared to ProCATM.

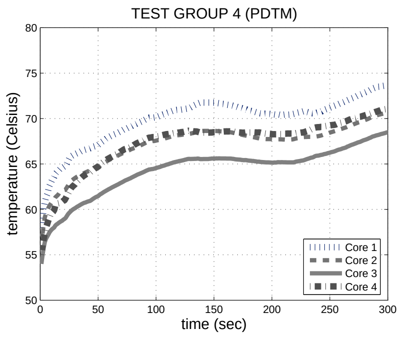
As shown in Fig. 37, we first notice that ProCATM presents a smoother temperature pattern, and provides better thermal fairness by having narrower temperature gaps among all cores in multimedia applications. ProCATM reduces the peak temperature by 1.35% compared to Linux Standard Scheduler, while both TBP and PDTM increase the peak temperature by 1.35%. Since there is only one non CPU-intensive multimedia application executed simultaneously, the temperature decrease in the proposed ProCATM is minor. In PDTM, the temperature pattern seems to be similar to the pattern of the Linux Standard Scheduler; however, the temperature of core 3 and core 4 in PDTM is higher than in Linux Standard Scheduler, because PDTM tends to migrate the threads into core 3 and core 4. Although PDTM rarely migrates the threads into core 1, some system threads can be assigned to core 1. Since the T_{ss} and thermal value b are higher, core 1 is more sensitive in temperature changing. Therefore, the system threads still keep core 1 in higher temperature, although the multimedia threads are running on core 3 and core 4. On the contrary, the core 1's temperature in TBP is even higher than in Standard Scheduler. Besides the higher T_{ss} and b of core 1, TBP trigger threads exchange while the thresholds are reached.



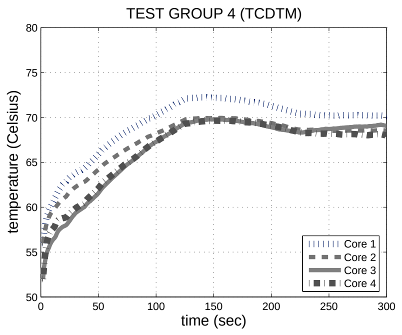
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



(d) ProCATM

Fig. 37. DTM evaluation in Intel Quad Core Q6600 for dynamic workload behaviors: Multimedia

Therefore, even though the thread in core 1 are exchanged out to avoid increasing core 1's temperature, the thread exchanged into core 1 still can potentially keep increasing the core 1's temperature. Therefore, the thermal safety cannot be guaranteed in TBP.

E. Conclusions

In this work, to avoid thermal emergencies and provide thermal fairness in CMP systems, we propose and implement an adaptive and scalable run-time thermal management scheme, called a Proactive Correlation-Aware Thermal Management (ProCATM), on the real-world CMP products. Since the significant variations in the thermal behaviors among different applications and the severe thermal correlation effect among multicores are ignored by all the prior DTM works. We suggest to characterize each application's distinct thermal behavior by applying a cumulative distribution function into the application workload and a proper thermal model for CMP systems to analyze the thermal correlation effect by profiling the thermal impacts from neighboring cores under the specific workload. Thus, the future temperature of each core can be more accurately estimated for adopting an appropriate reaction against the thermal emergency through the proposed ProCATM. To demonstrate the scalability and effectiveness, we implement and evaluate the proposed ProCATM in Intel Quad Core Q6600 processor system running grouped multimedia application and benchmarks. According to the experimental results, ProCATM reduces the peak temperature by up to 9.09% in our 4-core system with only 2.28% performance overhead compared to the Linux standard scheduler.

Algorithm 4 SWETM algorithm

```
1: while $T_{curr} < T_{threshold}$ do
2:     Calculate $CDF(W_{app})$ {according to the S.D.}
3:     Calculate $CDF(W_{core})$ {according to the S.D.}
4:     $T'_{ss} \leftarrow CDF(W_{core})$
5:     $\Delta t \leftarrow$ Calculate $WhenOverheated(T_{limit}, T'_{ss})$
6:     for $i = 1$ to $App\_Num$ (running within that core) do
7:         if $\Delta t \leq Response\_Time$ and $W_{appi}$ >$Workload\_Threshold$ then
8:             for $j = 1$ to $Core\_Num$ do
9:                 $W_{fj} \leftarrow W_{appi} + W_{corej}$
10:                $T'_{ssj} \leftarrow CDF(W_{fj})$
11:                $T_{fj} \leftarrow$ FTEM $(T'_{ssj}, \Delta$t)
12:                if $(T_{fj}$ - $T_{limit}) < 1$ then
13:                    Action $\leftarrow PRIOTIRY\_SCHEDULING$
14:                else
15:                    Action $\leftarrow MIGRATION$
16:                end if
17:            end for
18:            $Tigger\_Control(Action)$
19:        end if
20:    end for
21: end while
```

CHAPTER VIII

A THERMAL MODEL FOR CMPS CAPTURING WORKLOAD CHARACTERISTICS AND NEIGHBORING CORE EFFECTS

Skadron *et al.* first proposed an architectural thermal model for microprocessors, HotSpot [16], which constructs a multi-layer lumped thermal RC network to model the heat dissipation path from the silicon die through the cooling package to the ambient. In HotSpot, the silicon die is partitioned into functional blocks based on the floorplan of the microprocessor, with a thermal RC network connecting the blocks. However, due to the complexity of component block level thermal models and insufficient physical information extracted from floorplans in modern processors, it is not feasible to design a thermal model according to the hottest chip block. Therefore, we need a high-level thermal model that capture the thermal effect incurred by application behavior and can be managed by operating systems according to applications' runtime behavior.

A. The Lumped Thermal RC Model

In this section, we explain a lumped thermal RC model to capture thermal characteristics of processors as well as external cooling effects such as fans or cooling packages. We assume that the speed of external fans is static to be used as a constant in a thermal model. We apply Fourier's Law of heat conduction, which states that the cooling rate is proportional to the difference in temperature between the object and the environment. Hence, the heating rate is proportional to the difference between the current temperature and the steady state temperature reachable by the input power that is the heat source for processors. We define $T(t)$ and $P(t)$ to be the temperature and the power consumption at time t, respectively. Then, we formulate the Fourier's

Law as the follows [41, 28]:

$$T'(t) = \frac{P(t)}{C} - b \cdot T(t), \tag{8.1}$$

where b is a positive constant that represents the power dissipation rate, which is the inverse of time, $\tau = R \cdot C$. The parameters R and C are the thermal resistance and capacitance, respectively, and represent thermal characteristics of the chip. The heat transfer equations are introduced to model the steady state temperature of systems with heat sources in [37]. With those heat transfer equations, Wang and Bettati present that the rate of temperature change is proportional to the difference between the current temperature and the steady state in [30]. Let T_{ss} be the steady state temperature of an application. Then, we denote $T(t)$ as the temperature at time t and T_{init} as the initial temperature when an application starts execution ($T(0)$=T_{init}). Thus,

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{8.2}$$

where b is a thermal parameter. Solving Equation (8.2) with $T(0) = T_{init}$ and $T(\infty) = T_{ss}$, we can obtain

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-b \cdot t} \tag{8.3}$$

Fig. 38 shows the thermal RC circuit model for a single core in a CMP architecture. We assume that the initial temperature is T_{init}, i.e., $T(t_0) = T_{init}$. The core's temperature at time t ($T_c(t)$) is calculated using a lumped thermal RC model [16, 42] and expressed as follows.

$$T_c(t) \;=\; R_c \cdot P_c \cdot (1 - e^{-b_c \cdot t}) + R_p \cdot P_p \cdot (1 - e^{-b_p \cdot t}) + T_{init,c} \tag{8.4}$$

where $T_{init,c}$ and $T_{init,p}$ are the initial temperatures of the core and the package, respectively, R_c and R_p are the thermal resistances of the core and the package,

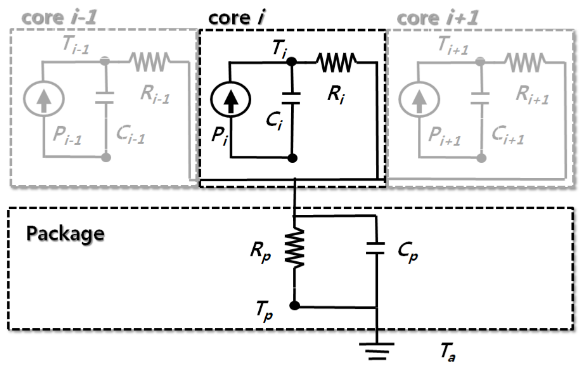


Fig. 38. An extended lumped thermal RC circuit model for a single core in a CMP architecture

respectively, and b_c and b_p are the thermal parameters of the core and the package, respectively. Also, P_c and P_p are the average power consumption of the core and the package at time interval t, respectively. The above approximation is derived from the fact that the thermal time constant of the core, $\tau_c = R_c \cdot C_c$, is much smaller than that of the package, $\tau_p = R_p \cdot C_p$ [42]. The thermal parameter values (R_c, R_p, b_c, b_p) in Equation (8.4) are determined from the temperature curve of a SPEC CPU 2006 benchmark.

We obtain the thermal parameters using nonlinear regression analysis provided by the SPSS statistics tool.

B. Workload-aware Thermal Model

Although the temperature of a core can be calculated by a lumped thermal RC model as shown in Equation (8.4), this model cannot provide the temperature variations caused by running applications. The temperature variations can be affected by individual functional blocks as mentioned in Hotspot [16], but it is difficult to obtain detail information of the cores at runtime. Therefore, we need to find new metrics to explain the thermal effects by workload in running applications. We propose a

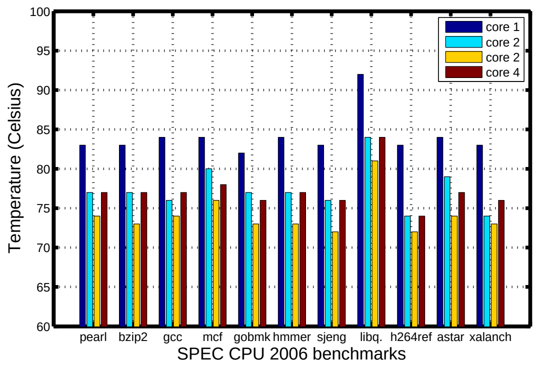


Fig. 39. T_{ss} of SPEC CPU 2006 benchmarks

workload-aware thermal model capturing more accurate workload characteristics of applications based on the architectural information. First, we approximate the steady state temperature (T_{ss}) using only thermal parameters of the cores. And then, a workload estimation factor that describes workload characteristics is calculated from a regression analysis using the variance of T_{ss} in running eleven SPEC CPU 2006 benchmarks. Using the workload estimation factor, we estimate temperature affected by workload characteristics of applications. In Equation (8.4), the steady state temperature (T_{ss}) can be represented when t becomes ∞.

$$\begin{aligned} T_c(\infty) &= R_c \cdot P_c + R_p \cdot P_p. \\ T_{ss} &\approx R_c \cdot P_c. \end{aligned} \tag{8.5}$$

Since $R_p \cdot P_p$ for the package is much smaller than $R_c \cdot P_c$ for the core and R_p and P_p are dependent on processor specifications, we approximate T_{ss} considering only the effect of cores, as shown in Equation (8.5).

We first run SPEC CPU 2006 benchmarks on each core until the temperature does not change anymore to obtain the respective steady state temperature. As shown in Fig. 39, T_{ss} of each SPEC CPU 2006 benchmark is different from each other, although their CPU utilizations are almost 100%. To represent various workload characteristics that affect T_{ss}, we add a workload estimation factor (w_x) describing workload characteristics for an application x as shown in Equation (8.6).

$$T_{ss}(x) = w_x \cdot R_c \cdot P_c, \tag{8.6}$$

where $T_{ss}(x)$ is the steady state temperature of an application x. To estimate w_x, we use the architectural information provided by Performance Monitoring Counters (PMC) which are a set of special-purpose registers built in modern microprocessors to store the counts of hardware-related activities within computer systems [19]. We use two counters in PMC; the number of accumulated cycles of functional blocks monitoring active clock cycles and the number of completed memory transactions monitoring memory transactions at runtime. "Unhalted CPU Cycle" counter is used as an indicator for a CPU-intensive work and "Bus Transactions Memory" counter as an indicator for a Memory-intensive work. However, "Bus Transactions Memory" counter does not provide how much memory is used for running applications. To provide more sufficient information about the memory characteristics of applications, the memory usage measured by the kernel is used, too. We define the workload estimation factor $w_x(t)$ for a running application x during a time interval $[t-1, t]$, as shown in Equation (8.7).

$$w_x(t) \quad = \quad \alpha \cdot C(t) + \beta \cdot M(t) + \gamma \cdot N(t), \tag{8.7}$$

where $C(t)$ is the unhalted clock cycle, $M(t)$ is the number of completed memory transactions, and $N(t)$ is the memory usage during a time interval $[t-1, t]$. We observed that $C(t)$ works as a positive factor in temperature variations, while $M(t)$ and $N(t)$ work as negative factors. After running four integer and floating-point benchmarks in SPEC CPU 2006 benchmarks, we obtain α, β, and γ of each core using a linear regression. For example, α, β, and γ of core 1 can be calculated to $3.28E{-}005$, $-4.28E{-}007$, and $-8.36E{-}006$, respectively, in SPEC CPU 2006 integer benchmarks. In the case of floating-point benchmarks, α, β, and γ of core 1 can be calculated to $2.58E-005$, $-2.35E-009$, and $-9.85E-006$, respectively. Compared to integer benchmarks, temperature variation of floating-point benchmarks is less affected by architecture information. Since α, β, and γ of each core are obtained by T_{ss} in Equation (8.6) before running applications, the workload characteristics can be described by the variance of $C(t)$, $M(t)$, and $N(t)$ at runtime, as shown in Equation (8.7). Using these three parameters, we can derive a thermal model as follows.

$$T_c(t) \quad = \quad w_x(t) \cdot R_c \cdot P_c \cdot (1 - e^{-b_c \cdot t}) + R_p \cdot P_p \cdot (1 - e^{-b_p \cdot t}) + T_{init,c}, \qquad (8.8)$$

where $T_c(t)$ is the core temperature at time t, $T_{init,c}$ is the initial temperatures of the core. Also, b_c and b_p are the thermal parameters for the core and the package and $w_x(t)$ is a workload estimation factor of a running application x. As shown in Equation (8.8), we estimate temperature using a workload-aware thermal model including workload characteristics of each application and the thermal parameters of the package as well as those of the cores.

To investigate how three factors affect temperature in running applications, we compare the actually measured temperature and the estimated temperature using *bzip2* integer benchmark and *lbm* floating-point benchmark in SPEC CPU 2006 benchmarks. The estimated temperature using only active CPU cycles shows huge estimation error compared to the measured temperature as shown in Fig. 40(a) and 41(a). The thermal model using active clock cycles, the memory transactions, and the memory usage shows more accurate temperature estimations, as shown in Fig. 40(d) and 41(d). Since our workload-aware thermal model considers a positive factor (active clock cycles) and negative factors (the memory transactions and the memory usage), the results show more accurate temperature estimations regardless of a workload type such as CPU-intensive or Memory-intensive applications.

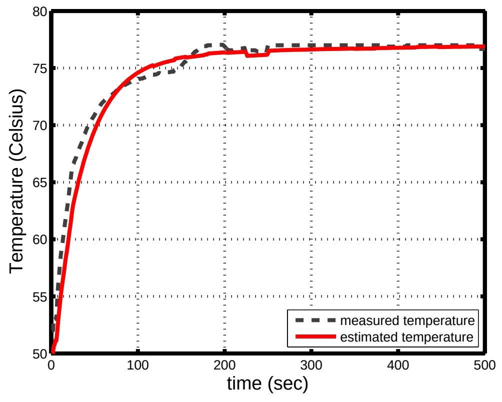
C. Thermal Correlation Effects

In this section, we develop a thermal correlation model for a CMP architecture based on the lumped thermal RC model in the lumped thermal RC model. The thermal model in a CMP architecture is composed of multiple nodes, one for each core in the package. The heat transfer conduction between neighboring cores is modeled by connecting them with a thermal resistance, as shown in Fig. 42. Each node is also connected to a current source, which models its power consumption, and this power is dissipated as heat with uniform power consumption. To calculate the thermal correlation effects among neighboring cores, we define the thermal correlation factor (ψ) that can be represented as a factor to estimate operating temperature affected by each core's own T_{ss} and the amount of heat transfer from neighboring cores. Hence, we calculate the temperature increase ratio (Γ) of T_{ss} as shown in Equation (8.9).

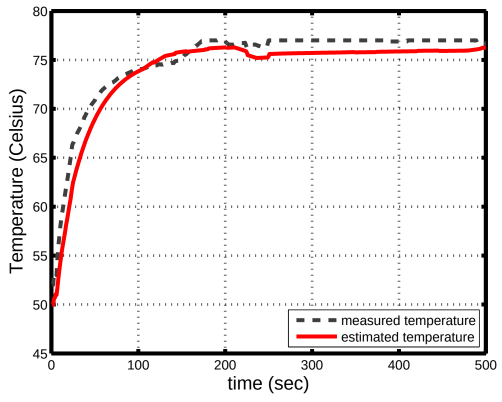
$$\Gamma_{i(j)} = \frac{(T^E_{ss,i(j)} - T_{init,i})}{(T_{ss,i} - T_{init,i})} = \frac{\Delta T^E_{ss,i(j)}}{\Delta T_{ss,i}} \tag{8.9}$$

where $\Gamma_{i(j)}$ is the ratio of temperature incremental change of T_{ss} between the steady state temperature ($T_{ss,i}$) for core i and the overall steady state temperature ($T^E_{ss,i(j)}$) for core i including the heat transfer from core j, and $T_{init,i}$ is the initial temperature of core i. $\Gamma_{1(2)}$ is 1.207 in Table XI, which implies that the overall T_{ss} of core 1 is raised by 20.7% compared to T_{ss} for only core 1 without any heat transfer. Therefore, we can define the thermal correlation factor of core i ($\psi_{i(j)}$) by heat transfer from core j as follows:

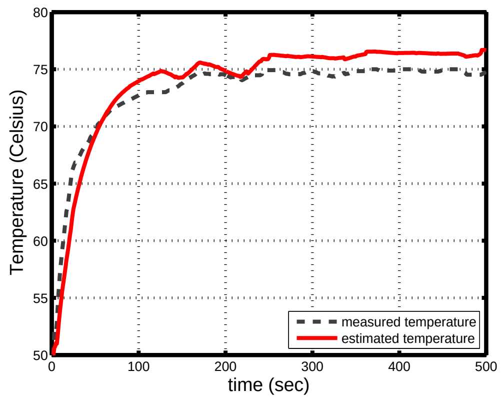
$$\psi_{i(j)} = \Gamma_{i(j)} - 1, \tag{8.10}$$



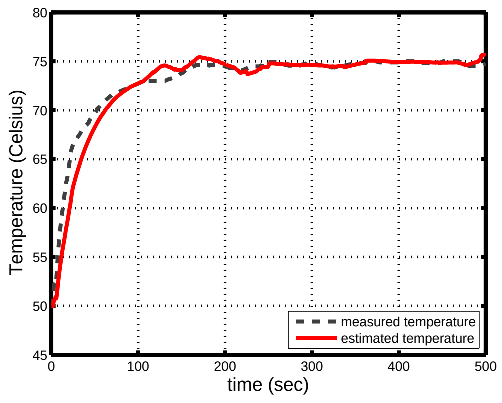
(a) Using only active CPU cycles



(b) Using cycle and memory usage

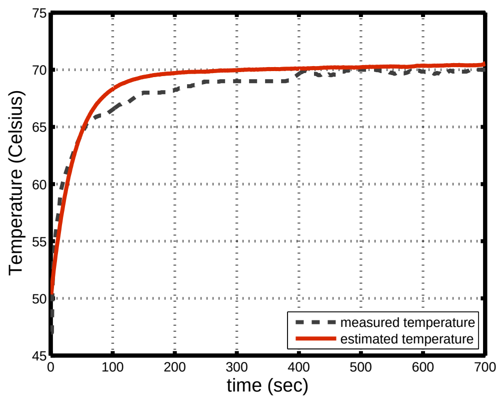


(c) Using cycle and memory transactions

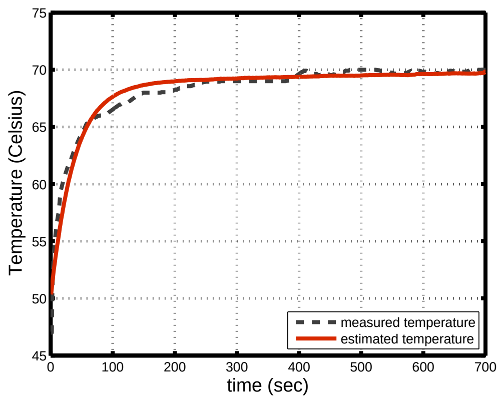


(d) Using active CPU cycles and the memory transactions

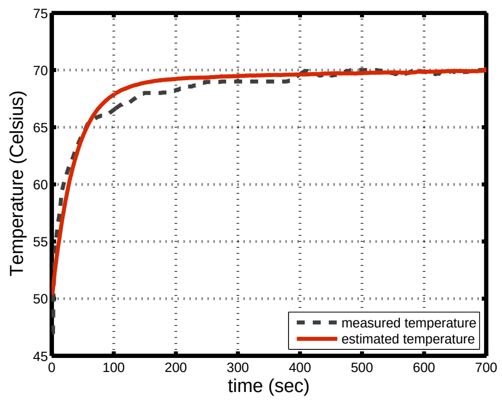
Fig. 40. Temperature tracking using architectural information in SPEC CPU integer benchmarks



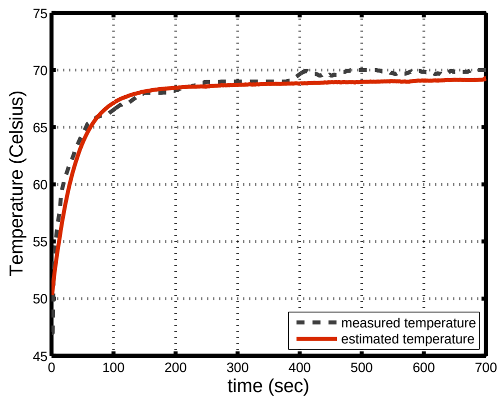
(a) Using only active CPU cycles



(b) Using cycle and memory usage



(c) Using cycle and memory transactions



(d) Using active CPU cycles and the memory transactions

Fig. 41. Temperature tracking using architectural information in SPEC CPU floating–point benchmarks

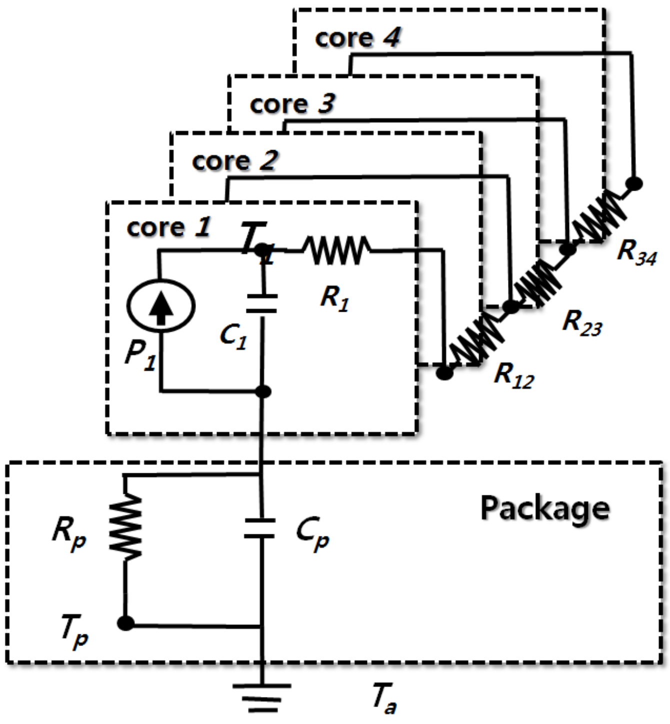


Fig. 42. The extended thermal model for CMP architecture

where $\psi_{i(j)}$ is the thermal correlation factor between cores i and j.

To derive the thermal model for a CMP architecture, each core's thermal variations can be expressed with the same approach as the approximation suggested for the single core.

As shown in Equation (8.11), the temperature of core i can be approximated using core i's thermal parameters and the thermal correlation factor (ψ). Also, we can estimate the temperature for core i using workload estimation factor ($w_(t)$) for cores and the thermal correlation effects from other cores as follows:

$$\begin{aligned} T_i(t) &= w(t) \cdot R_i \cdot P_i \cdot (1 - e^{-b_i \cdot t}) + R_p \cdot P_p \cdot (1 - e^{-b_p \cdot t}) + T_{init,i} \\ &+ \sum_{j=1, j \neq i}^{n} \psi_{i(j)} \cdot T_j(t), \end{aligned} \tag{8.11}$$

where R_i and b_i are the thermal resistance and thermal parameter for core i, re-

Table XI. The ratio of T_{ss} for cores in an Intel Quad-Core processor

	Γ_1	$\Gamma_{1(2)}$	$\Gamma_{1(3)}$	$\Gamma_{1(4)}$
core 1	1.000	1.207	1.034	1.17
	Γ_2	$\Gamma_{2(1)}$	$\Gamma_{2(3)}$	$\Gamma_{2(4)}$
core 2	1.000	1.080	1.040	1.000
	Γ_3	$\Gamma_{3(1)}$	$\Gamma_{3(2)}$	$\Gamma_{3(4)}$
core 3	1.000	1.000	1.000	1.037
	Γ_4	$\Gamma_{4(1)}$	$\Gamma_{4(2)}$	$\Gamma_{4(3)}$
core 4	1.000	1.034	1.034	1.069

spectively, R_p and b_p are thermal resistance and thermal parameter for the package, respectively, ψ is the thermal correlation factor, $w(t)$ is the workload estimation factor of core i, and $T_i(t)$ is temperature of core i at time t. This final model uses the workload-aware thermal model and the thermal correlation effects in a CMP architecture. Unlike prior studies that use simulations to evaluate their own models, we implement and evaluate our thermal model in a 4-core (Intel Quad Core Q6600) system. To evaluate our thermal model capturing workload characteristics and the thermal correlation effects, we developed a specific device driver for Linux to access the Digital Thermal Sensor (DTS) for monitoring each core's temperature for Intel microarchitecture. Also, we developed monitoring and estimation tasks to capture architectural information via Performance Monitoring Counters (PMC) and perform estimating workload characteristics at runtime. As shown in Fig. 43, we show the overall prototype implementation and measurement setup for our experiments. This diagram depicts different aspects of our implementation that correspond

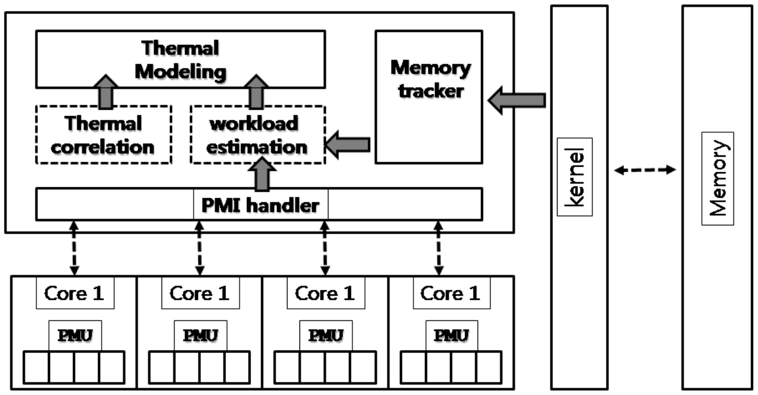
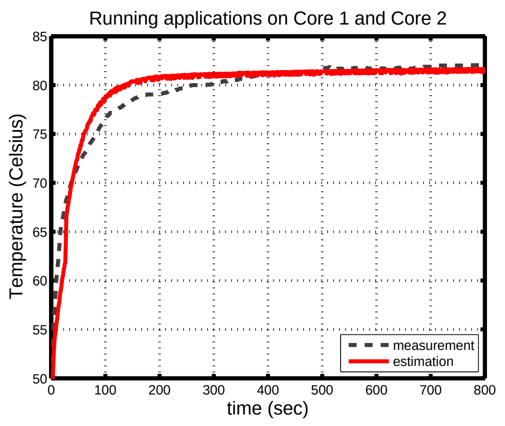
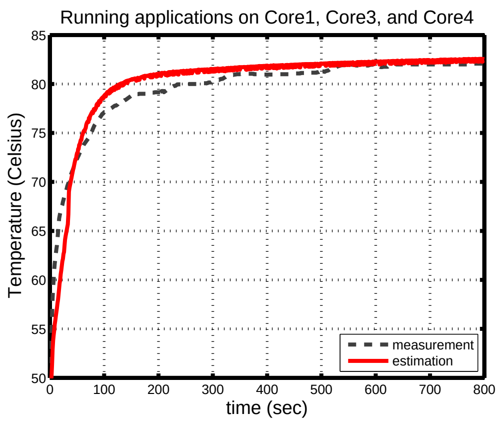


Fig. 43. The proposed platform

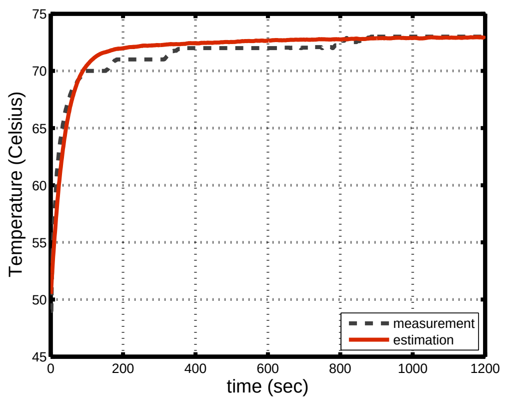
to determine workload characteristics and online thermal tracking in a real-product. To verify the thermal model, we profile thermal parameters and a workload estimation factor of each core using four integer and floating-point benchmarks in SPEC CPU 2006 benchmarks. After profiling, we compare the temperature predicted by our thermal model with the measured temperature, as shown in Fig. 44. The estimated temperature shows above 97 % temperature prediction accuracy compared to an actual temperature measurement of each core, when two *libquantum* integer benchmark and *lbm* floating-point benchmark in SPEC CPU 2006 benchmarks run on cores 1 and 2 simultaneously. When three *libquantum* integer benchmark and *lbm* floating-point benchmark in SPEC CPU 2006 benchmarks run on cores 1, 3, and 4, the thermal model estimate temperature more accurately compared to an actual temperature measurement, as shown in Fig. 44(b) and 44(d), respectively. Although we show only two results due to space limitations, the average temperature prediction accuracy of the thermal model is above 92% while we test the thermal model with seven integer and floating-point SPEC CPU 2006 benchmarks.



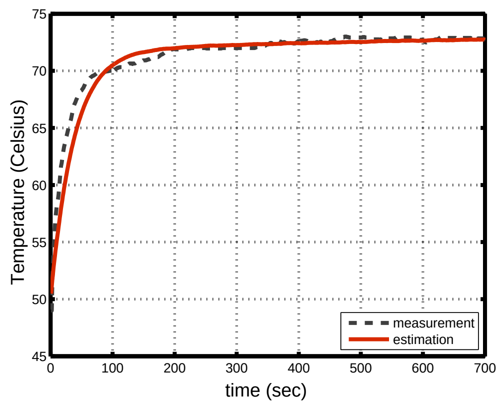
(a) Running *libquantum* SPEC CPU 2006 integer benchmark on Core 1 and Core 2



(b) Running *libquantum* SPEC CPU 2006 integer benchmark on Core 1, Core 3, and Core 4



(c) Running *lbm* SPEC CPU 2006 floating-point benchmark on Core 1 and Core 2



(d) Running *lbm* SPEC CPU 2006 floating-point benchmark on Core 1, Core 3, and Core 4

Fig. 44. The comparisons between the estimated temperature considering workload characteristics and thermal correlation effects and the measured temperature in a CMP architecture

D. Conclusions

As increasing power density with technology advance, the thermal control in CMPs has been a critical issue for improving system reliability. In this work, we propose a more accurate thermal model capturing workload characteristics and the thermal correlations among neighboring cores in a CMP architecture. The thermal model estimates temperature using a lumped thermal RC model enhanced with a workload estimation factor approximated by a regression analysis and the thermal correlation effects factors that describe the amount of heat transfer from neighboring cores. To estimate workload characteristics of running applications, we use active CPU cycles and the number of completed memory transactions provided by Performance Monitoring Counters (PMC), and the memory usage by the kernel.

To demonstrate the scalability and effectiveness of our thermal model, we implement and evaluate it in a 4-core (Intel Quad Core Q6600) system running integer and floating-point SPEC CPU 2006 benchmarks. As the experimental results, our thermal model shows above 92% temperature prediction accuracy, compared to an actual temperature measurement of each core. In the future work, we will develop Dynamic Thermal Management (DTM) for CMPs exploiting accurate temperature prediction provided by our thermal model.

CHAPTER IX

CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize the major results of this thesis and discuss future directions of this work.

A. Conclusions

In Chapter III, we propose an efficient thermal management for multimedia applications considering performance of a multimedia system affected by the complexity of scenes, and then we find an appropriate frequency based on the information on scene complexity.

In Chapter IV, we first derive application characteristics in various multimedia applications by transmitting MPEG-4 and H.264/AVC encoded by two different frame resolutions. By using this applications' characteristics, we estimate a processor speed to execute multimedia application for decoding frames at runtime.

In Chapter V, we presents an advanced future temperature prediction model for each core to estimate the thermal behavior considering both core temperature and applications temperature variations and take appropriate measures to avoid thermal emergencies.

In Chapter VI, the proposed TAS scheme utilizes an advanced future temperature prediction model for each core to estimate different thermal behaviors and measure the time left until each core reaches the desired temperature threshold.

In Chapter VII, to avoid thermal emergencies and provide thermal fairness in CMP systems, we propose and implement an adaptive and scalable run-time thermal management scheme, called a Proactive Correlation-Aware Thermal Management (ProCATM), on the real-world CMP products.

Chapter VIII presents a thermal model based workload characteristics of running applications. We propose the thermal model based on thermal correlation effects and online workload estimation using architectural information via Performance Monitoring Counters (PMC).

B. Future Work

Chip Multiprocessors (CMPs) have been pervasive in modern processor designs, and it is likely that the demand for management temperature under thermal safety in a CMP architecture will remain very high. In order to solve the dilemmatic tradeoff between an efficient thermal management and performance degradation in CMP architecture, we would like to expand this work to cover real CMP products such as 4-core system (Intel Q6600 Quad-Core) and 8-core system (two Intel Xeon E5310 Quad-Core) that are increasingly used in high-performance system. This involves addressing issues associated with efficient overlay self-reconfiguration and maintenance, load balancing among servers in the network, and optimal selection of servers or paths for streaming services. We also have interest in studying online workload estimation of running applications in real environments.

In addition, we have further interest in applications' execution behaviors, which can directly affect temperature in a CMP architecture. As the speed of processors increases and the complexity of a chip becomes higher, the thermal management in a CMP architecture can become very susceptible to application execution behavior because it is difficult to capture it at runtime. Therefore, we need a metric to represent applications' execution behaviors. Although our work succeeded at designing accurate online workload estimation and modeling the thermal correlation effects for Dynamic Thermal Management (DTM), new technologies have introduced such as

CPU hotpulgging and individual Dynamic Voltage and Frequency Scaling of each core in a CMP architecture, and then their integrations can be required to be applied into DTM in future work.

REFERENCES

[1] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," in *Proc. IEEE HPCA*, pp. 171–182, 2001.

[2] S. Heo, K. Barr, and K. Asanovic, "Reducing Power Density through Activity Migration," in *Proc. IEEE ISLPED*, pp. 217–222, 2003.

[3] K. Skadron, "Hybrid Architectural Dynamic Thermal Management," in *Proc. IEEE DATE*, pp. 10–15, 2004.

[4] J. Srinivasan and S. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications," in *Proc. ACM ICS*, pp. 109–120, 2003.

[5] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive Dynamic Thermal Management for Multicore Systems," in *Proc. ACM DAC*, pp. 734–739, 2008.

[6] F. Mulas, M. Buttu, M. Pittau, S. Carta, D. Atienza, A. Acquaviva, L. Benini, and G. D. Micheli, "Thermal Balancing Policy for Streaming Computing on Multiprocessor Architectures," in *Proc. IEEE DATE*, pp. 734–739, 2008.

[7] D. Son, C. Yu, and H. N. Kim, "Dynamic Voltage Scaling on MPEG Decoding," in *Proc. IEEE ICPADS*, pp. 633–640, 2001.

[8] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor," in *Proc. ACM MOBICOM*, pp. 251–259, 2001.

[9] B. Lee, E. Nurvitadhi, R. Dixit, C. Yu, and M. Kim, "Dynamic Voltage Scaling Techniques for Power Efficient Video Decoding," *the EUROMICRO Journal*, vol. 51, no. 10-11, pp. 633–652, 2005.

[10] Z. Lu, J. Lach, M. Stan, and K. Skadron, "Reducing Multimedia Decode Power using Feedback Control," in *Proc. IEEE ICCD*, pp. 489–496, 2003.

[11] I. Yeo, H. K. Lee, K. H. Yum, and E. J. Kim, "Effective Dynamic Thermal Management for MPEG-4 Decoding," in *Proc. IEEE ICCD*, pp. 623–628, 2007.

[12] K. Choi, K. Dantu, W. C. Cheng, and M. Pedram, "Frame-based Dynamic Voltage and Frequency Scaling for a MPEG Decoder," in *Proc. IEEE ICCAD*, pp. 732–737, 2002.

[13] W. Yuan and K. Nahrstedt, "Reduced energy decoding of MPEG streams," *ACM Trans. Computer System*, vol. 24, no. 3, pp. 292–331, 2006.

[14] K. Skadron, M.Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Trans. Architecture and Code Optimization*, vol. 1, no. 1, 2004.

[15] M. R. Stan, K. Skadron, M. Barcella, W. Huang, K. Sankaranarayanan, and S. Velusamy, "HotSpot: a Dynamic Compact Thermal Model at the Processor Architecture Level," *Microelectronics Journal: Circuit and Systems*, vol. 1, no. 1, 2003.

[16] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture," in *Proc. IEEE ISCA*, pp. 2–13, 2003.

[17] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System," in *Proc. ACM ASPLOS-XI*, pp. 260–270, 2004.

[18] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-Aware Task Scheduling at the System Software Level," in *Proc. IEEE ISLPED*, pp. 213–218, 2007.

[19] "Intel 64 and IA-32 Architectures Software Developer's Manual," http://www.intel.com/products/processor/manuals/ (Accessed on 01-07-10).

[20] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," in *Proc. IEEE HPCA*, pp. 17–28, 2002.

[21] K.-J. Lee and K. Skadron, "Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors," in *Proc. IEEE IPDPS*, pp. 232–237, 2005.

[22] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips, "Power-Aware Video Decoding," in *22nd Picture Coding Symposium*, 2001.

[23] M. Mesarina and Y. Turner, "Reduced energy decoding of MPEG streams," *Multimedia System*, vol. 9, no. 2, pp. 202–213, 2003.

[24] X. Liu, P. Shenoy, and M. Corner, "Chameleon: Application Level Power Management with Performance Isolation," in *Proc. ACM MULTIMEDIA*, pp. 839–848, 2005.

[25] P. Michaud, A. Seznec, D. Fetis, Y. Sazeides, and T. Constantinou, "A Study of Thread Migration in Temperature-Constrained Multicores," *ACM Trans. Architecture and Code Optimization*, vol. 4, no. 2, 2007.

[26] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, "HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management," in *Proc.*

ACM DAC, pp. 548–553, 2006.

[27] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha, "Thermal Modeling, Characterization and Management of On-Chip Networks," in *Proc. IEEE MICRO*, pp. 67–78, 2004.

[28] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed Scaling to Manage Energy and Temperature," *Journal of ACM*, vol. 54, no. 1, pp. 1–39, 2007.

[29] J. Sergent and A. Krum, *Thermal Management Handbook*, Columbus, USA, McGraw-Hill, 1998.

[30] S. Wang and R. Bettati, "Reactive Speed Control in Temperature-Constrained Real-Time Systems," in *Proc. IEEE ECRTS*, pp. 73–95, 2006.

[31] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," in *Proc. ACM SOSP*, pp. 149–163, 2003.

[32] J. R. Lorch and A. J. Smith, "Improving Dynamic Voltage Scaling Algorithms with PACE," in *Proc. ACM SIGMETRICS*, pp. 50–61, 2001.

[33] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proc. IEEE ISLPED*, pp. 46–51, 2001.

[34] PAPI, "Performance API," Available from http://icl.cs.utk.edu/papi (Accessed on 01-07-10).

[35] Intel, "Intel Atom Processor," http://www.intel.com/products/processor/atom (Accessed on 01-07-10).

[36] X. Chen, "Recursive Least-Squares Method with Membership Functions," in *Proc. IEEE Machine learning and Cybernetics*, pp. 1962–1966, 2004.

[37] F. Kreith and M. S. Bohn, *Principles of Heat Transfer*, Monterey, USA, Brooks/Cole Publishing Company, 2000.

[38] D. Bovet and M. Cesati, *Understaning the Linux Kernel*, Sebastopol, USA, O'Reilly Media, Inc, 2005.

[39] C. C. Minh, "STAMP - Stanford Transactional Applications for Multi-Processing," Available from http://stamp.stanford.edu/ (Accessed on 01-07-10).

[40] J. Li and J. F. Martinez, "Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors," in *Proc. IEEE ISPASS*, pp. 124–134, 2005.

[41] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic Speed Scaling to Manage Energy and Temperature," in *Proc. IEEE FOCS*, pp. 520–529, 2004.

[42] R. Rao and S. Vrudhula, "Performance Optimal Processor Throttling under Thermal Constraints," in *Proc. IEEE CASES*, pp. 257–266, 2007.

VITA

In Choon Yeo received his B.S. in computer engineering from Dongguk University, Korea, in 1995 and his M.S. in computer engineering from Dongguk University, Korea, in 1997. He graduated with a Ph.D. in computer science and engineering from Texas A&M University in December 2009.

During 1997-2004, he worked as a system engineer for Sindoricoh in Korea. His research interests include high-performance energy-efficient computer architectures, Dynamic Thermal Management (DTM) and Dynamic Power Management (DPM) on Multicore, compiler and hardware support for dynamic optimizations, virtual machines, and binary instrumentation. He may be contacted at:

Department of Computer Science and Engineering

Texas A&M University

College Station, TX 77843-3112