DYNAMIC THERMAL MANAGEMENT
IN CHIP MULTIPROCESSOR SYSTEMS

A Thesis

by

CHIH-CHUN LIU

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

MASTER OF SCIENCE

August 2008

Major Subject: Computer Science

DYNAMIC THERMAL MANAGEMENT IN CHIP MULTIPROCESSOR SYSTEMS

A Thesis

by

CHIH-CHUN LIU

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

MASTER OF SCIENCE

August 2008

Major Subject: Computer Science

ABSTRACT

Dynamic Thermal Management
in Chip Multiprocessor Systems. (August 2008)

Chih-Chun Liu, B.S., Feng-Chia University
Chair of Advisory Committee: Dr. Eun Jung Kim

Recently, processor power density has been increasing at an alarming rate resulting in high on-chip temperature. Higher temperature increases current leakage and causes poor reliability. In our research, we first propose a Predictive Dynamic Thermal Management (PDTM) based on Application-based Thermal Model (ABTM) and Core-based Thermal Model (CBTM) in the multicore systems. Based on predicted temperature from ABTM and CBTM, the proposed PDTM can maintain the system temperature below a desired level by moving the running application from the possible overheated core to the future coolest core (migration) and reducing the processor resources (priority scheduling) within multicore systems. Furthermore, we present the Thermal Correlative Thermal Management (TCDTM), which incorporates three main components: Statistical Workload Estimation (SWE), Future Temperature Estimation Model (FTEM) and Temperature-Aware Thread Controller (TATC), to model the thermal correlation effect and distinguish the thermal contributions from applications with different workload behaviors at run time in the CMP systems. The proposed PDTM and TCDTM enable the exploration of the tradeoff between throughput and fairness in temperature-constrained multicore systems.

ACKNOWLEDGMENTS

I sincerely appreciate Dr. Eun Jung Kim's academic advising in my research. Due to Dr. Kim's tireless support and guidance, I can finish this thesis and earn valuable experience. Moreover, I would like to express special thanks to my coworker - Inchoon Yeo and all other group members in the High Performance Computing Laboratory.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Chip multiprocessors (CMPs) have already been employed as the main trend in new generation processors. A CMP 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, 2, 3, 4, 5] techniques have been proposed in sense of that they, except [5], start to control the temperature after the current temperature reaches the critical temperature threshold. Dynamic Thermal Management 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 [6]. However, neighboring thermal effect and application thermal behavior are not considered in prior works. Due to packaging technology in

The journal model is *IEEE Transactions on Automatic Control.*

CMP, 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 [4]. There are significant variations in the thermal behavior among different applications [4, 5]. Motivated by these facts, we first propose the Predictive Dynamic Thermal Management (PDTM) in this research to utilize 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.

Furthermore, although thermal-aware thread migration in the CMP environment has been introduced in the studies above, no prior attempt has been made to discover the thermal correlation effect among neighboring cores. However, according to our observation, the temperature of a single core can be affected by 2°C to 16°C depending on different levels of thermal correlation in our 4-cores CMP system. Moreover, it is known that temperature of a component is highly correlated with that of other components in the same chip [4, 7, 8, 9]. The temperature model, capturing a neighboring effect in an uniprocessor, cannot be directly applied to that of CMPs, due to their potential heterogeneity where each core has an independent thread to run. Moreover, the significant variations in the thermal behaviors among different applications have already been introduced in [4, 5]. Although, there have been a handful of studies using simple workload models, such as the average of workload and IPC (Instructions Per Cycle), for the Dynamic Thermal Management (DTM), the workload information is measured offline in their studies [6, 10]. We believe that it is necessary to develop an efficient online workload estimation scheme for DTM to be applicable to the real world applications which have variable workload behaviors and distinct thermal contributions to the increased chip temperature. Instead of conducting our DTM design upon the worst-case analysis, we drive our DTM design toward to the average-case thermal management in the CMP systems. Hence, there is a need to develop an efficient and practical DTM scheme by modeling the thermal correlation effect and distinguishing the thermal contributions from applications with different workload behaviors at runtime. Hence, in this research, we also propose the Thermal Correlative Dynamic Thermal Management (TCDTM) to characterize applications' dynamic workload behaviors and model the thermal correlation effect among neighboring cores. Both PDTM and TCDTM will be thoroughly discussed in the following chapters.

CHAPTER II

RELATED WORK

Nowadays, several thermal control techniques have been proposed and applied in modern processors via either hardware-based or software mechanisms [1, 8]. Hardware-based DTM mechanisms, such as Dynamic Frequency Scaling (DFS) and Dynamic Voltage Scaling (DVS), as well as clock gating, are able to effectively reduce processor's temperature and guarantee thermal safety, but with high execution performance overhead. In [1], the key goals of DTM were stated as: (1) to provide inexpensive hardware or software responses, (2) that reliably reduce power, (3) while impacting performance as little as possible. In [11], HybDTM, a methodology for fine-grained, coordinated thermal management using both software (priority scheduling) and hardware (clock gating) techniques, is proposed. In order to estimate temperature, HybDTM proposed a regression-based thermal model based on using hardware performance counters. However, HybDTM can not effectively reduce overheat temperature without performance overhead, because real temperature cannot be estimated solely by hardware performance counter, and both of priority scheduling and clock gating will introduce high performance overhead. Their performance overhead is 9.9% compared to the case without any DTM. Therefore, as the multicore processors become popular, some software-based thermal management mechanisms, such as thread migration in a CMP has been studied in [6, 10, 12]. In [6], 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; on the other hand, the CMP thread migration cools overheated cores by migrating threads away from overheated cores and assigning them to free SMT contexts on alternate cores to maintain throughput while allowing cooling overheated cores. They showed that for four cores CMP running five threads, 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, 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 has been confirmed in [13]. In [13], a new migration method for temperature-constrained multicore is proposed 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 with HRTM, but requires much less migrations. In [12], the authors further discuss about the migration performance overhead by considering the different memory usages among applications. They propose to calculate the average temperature among all cores and set upper/lower threshold. Once a core (source core) reaches the upper/lower thresholds, the migration would be triggered and the threads on the source core would be exchanged with those on the target core. The target core is determined by considering the temperature difference between source and target cores and the data size of the tasks running on the both cores. A core with higher temperature difference and smaller data size tends to be selected as the target core to provide thermal balance and reduce performance overhead. However, the application workload behaviors are ignored. That means that sets of running threads will be migrated without considering the different thermal effects caused by various threads, while the core temperature reaches the upper/lower temperature threshold. Moreover, it is not guaranteed that the exchanged threads would not keep increasing the source core's temperature and eventually overheated. Furthermore, the migration action in these studies is triggered by the current temperature (when the temperature reaches the predefined threshold);

however, instead of considering the current temperature, we believe that an accurate future temperature prediction model could perform more effectively in lowering the peak temperature. However, as presented in [4], an accurate and practical dynamic model of temperature is needed to accurately characterize current and future thermal stress, 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. Most importantly, except [10], these prior works above are based on simulated results, and neglect the thermal correlation among cores. The power dissipated by the rest of the chips is assumed to be negligible. However, the neighboring thermal correlation plays a significant role in the thermal management in the CMP environment. In [10], the thermal-aware scheduler is proposed and implemented on a 1.2G POWER5 system. However, the thermal correlation is not discussed in this work and the scheduler requires tasks being predefined as cold or hot tasks in advance. This may be unpractical in the real-world to classify all tasks before running.

CHAPTER III

PREDICTIVE DYNAMIC THERMAL MANAGEMENT FOR MULTICORE SYSTEMS

In this chapter, we would discuss about the proposed 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 take appropriate measures to avoid thermal emergencies. To evaluate the proposed PDTM, we 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 `SPEC2006` benchmarks simultaneously show the proposed PDTM lowers temperature by about 5% in average and reduces up to 3℃ 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 [6], and our PDTM outperforms HRTM in reducing average temperature by about 7%, performance overhead by 0.15%, and peak temperature by about 3℃, while running single benchmark.

The main contributions of the proposed PDTM 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.
- 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 PDTM.

1. The Application-based Thermal Model

The Application-based Thermal Model (ABTM) accommodates short-term thermal behavior in order to predict the future temperature in fine-grained. As shown in Figure 1, 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 Recursive

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

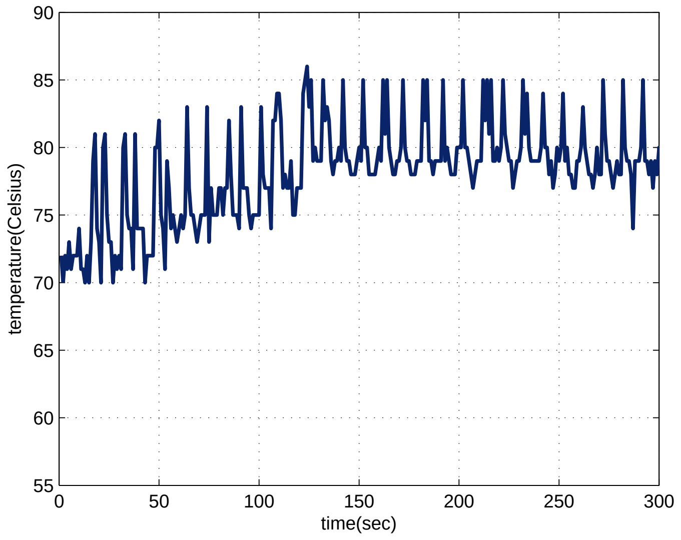


Fig. 1. Real temperature of one core on running `bzip2` benchmark

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

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \cdots + \theta_n f_n(u), \tag{3.1}$$

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:

$$X = \begin{bmatrix} f_1(u_1) & \cdots & f_n(u_1) \\ \vdots & \vdots & \vdots \\ f_1(u_m) & \cdots & f_n(u_m) \end{bmatrix} \tag{3.2}$$

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

$$\theta = [\theta_1, \theta_2, ..., \theta_n]^T \tag{3.3}$$

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

$$Y = [Y_1, Y_2, ..., Y_n]^T \tag{3.4}$$

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

$$\theta = (X^T X)^{-1} X^T Y, \tag{3.5}$$

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 (3.5). Likewise, the recursive least square method at $(k+1)^{th}$ can be developed as

$$P_{k+1} = P_k - \frac{P_k x_{k+1} x_{k+1}^T P_k}{1 + y_{k+1}^T P_k y_{k+1}}, \tag{3.6}$$

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

$$\theta_{k+1} = \theta_k + P_{k+1} x_{k+1} (y_{k+1} - x_{k+1}^T \theta_k) \tag{3.7}$$

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 [14]. With Equation (3.1), ABTM can predict future temperature for an application as shown in Figure 2. How the ABTM applied in PDTM is explained in Section 3.

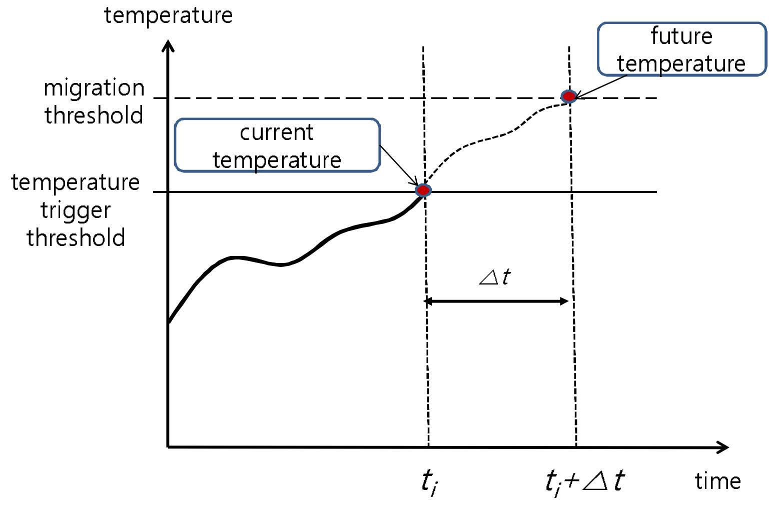


Fig. 2. The calculation of Δt(migration time) using ABTM

2. The Core-based Thermal Model

The heat transfer equations model the steady state temperature of systems with heat sources [15]. 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 [15]. 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 change of temperature is proportional to the difference between the current temperature and the steady state temperature of the application. Thus

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{3.8}$$

Solving Equation (3.8) with $T(0) = T_{init}$ and $T(\infty)$=T_{ss}, we get

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{3.9}$$

where b is a processor-specific constant. The value of b is determined using Equation (3.8) by observing heating and cooling curves corresponding to all `SPEC2006` 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. Then by rearranging Equation (3.9), we get the steady state temperature T_{ss} of the application.

$$T_{ss} = \frac{T(t) - T_{init} \times e^{-bt}}{(1 - e^{-bt})} \tag{3.10}$$

Therefore, with Equation (3.9) and (3.10), 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 run-time 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:

$$T_{predict} = w_s T_{app} + w_l T_{core} \tag{3.11}$$

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 found that the optimal values for w_s and w_l are 0.7 and 0.3 respectively based on our experimental results.

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 programs 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 [16]. 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, the proposed PDTM enables the scheduling

[2]Process is migrated among several cores very frequently.

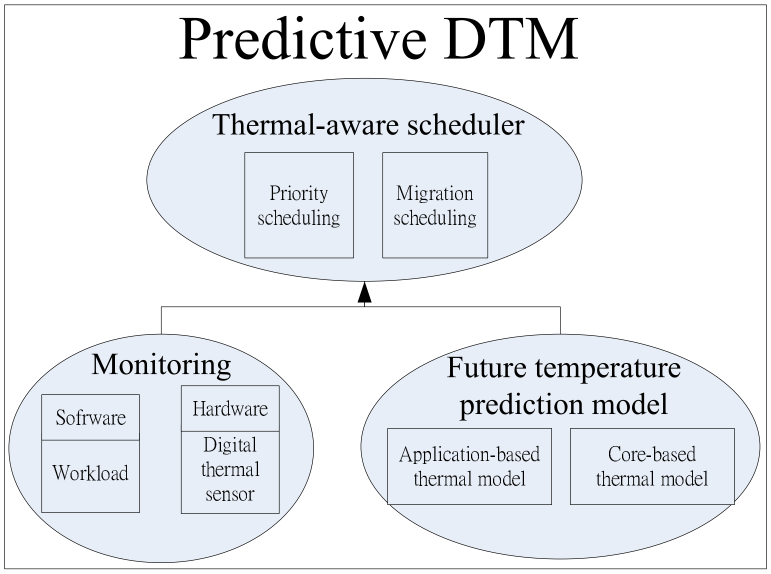


Fig. 3. System overview

policy to accommodate the temperature behavior as well as workloads in a multicore environment.

Our PDTM mainly composes of three components as shown in Figure 3. In the monitoring part, application workload (CPU utilization) is monitored for application's migration to balance workload by Linux standard scheduler. However, it is not aware of temperature. Our PDTM uses Digital Thermal Sensor (DTS) to detect temperature at run-time. The detected temperature information will be used in the future temperature prediction model.

As shown in Figure 4, PDTM determines that migration is necessary when 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

Algorithm 1 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. 4. PDTM scheduler algorithm

minimum value among future temperature ($T[]$) is selected as new core for migration. As shown in Figure 5, our 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.

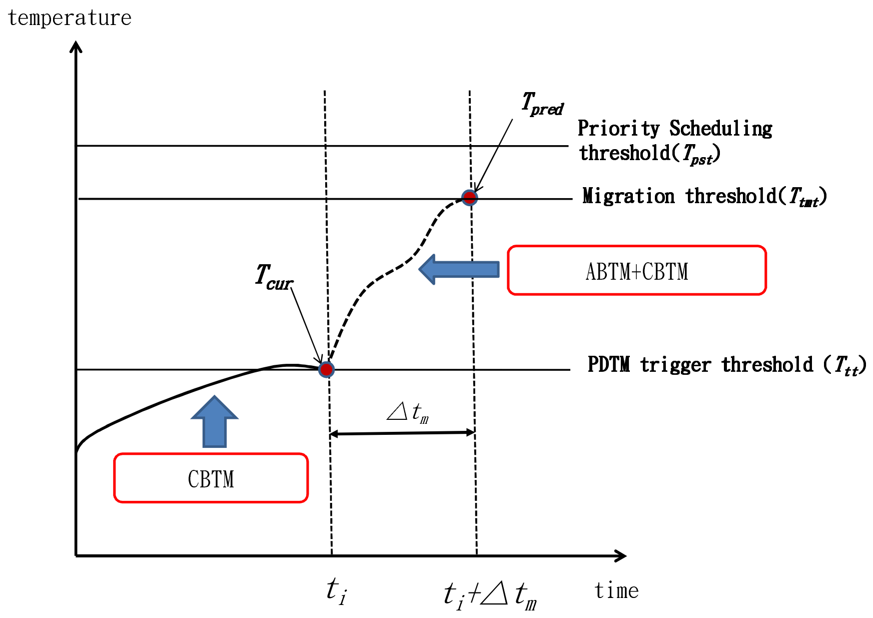
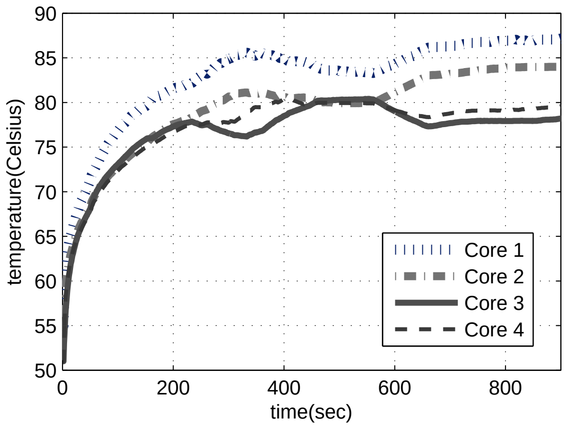

temperature
T_{pred}
Priority Scheduling threshold(T_{pst})
Migration threshold(T_{mt})
ABTM+CBTM
T_{cur}
PDTM trigger threshold (T_{tt})
$\triangle t_m$
CBTM
t_i
$t_i+\triangle t_m$
time

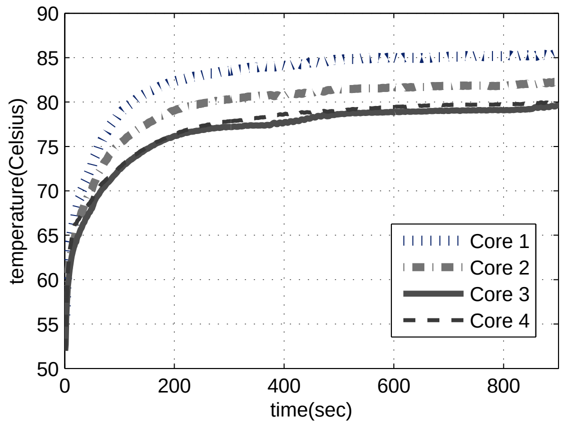
Fig. 5. PDTM utilizes ABTM and CBTM simultaneously to predict both short-term and long-term future temperature for multicore

C. PDTM Implementation and Analysis

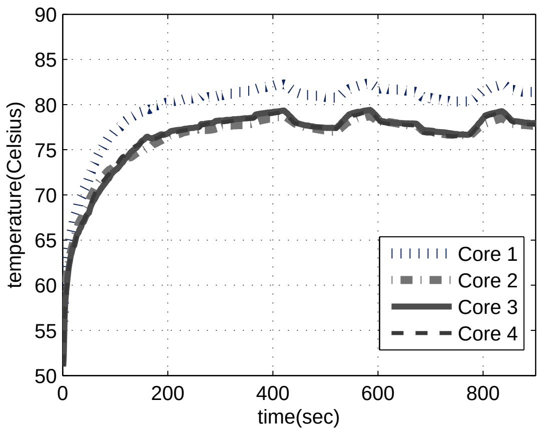
In order to estimate working temperature through Digital Thermal Sensor (DTS) for multicore systems, we develop a specific driver to access them in runtime. In a chip-multiprocessor (CMP) silicon die, each core has a unique thermal sensor that triggers independently. The trigger point of these thermal sensors is not programmable by software since it is set during the fabrication of the processor [17]. In our experiments, we set temperature trigger threshold as 60℃ to start PDTM, and the migration threshold as 70℃ to migrate applications when the predicted temperature exceeds



(a) Without DTM

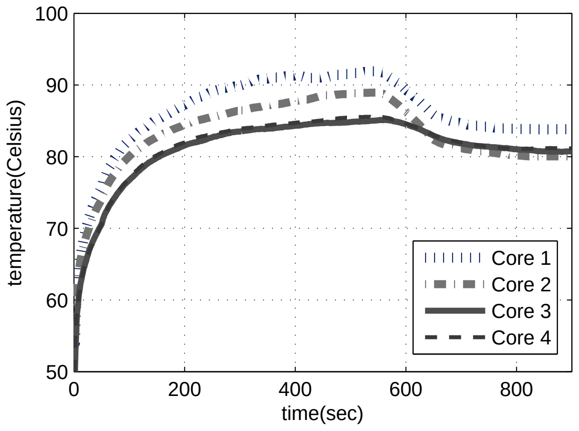


(b) HRTM

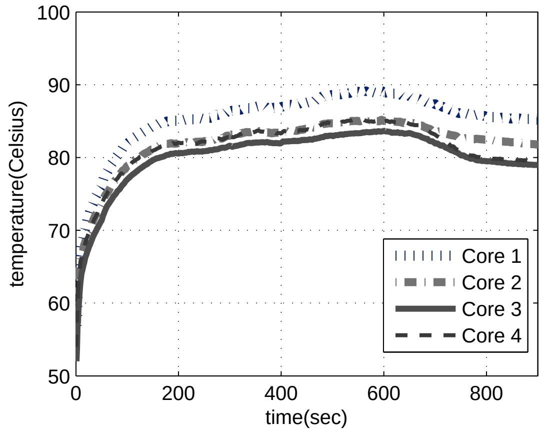


(c) PDTM

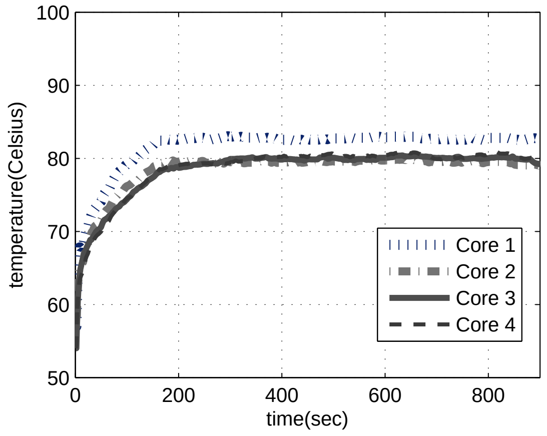
Fig. 6. Comparisons among the three different schemes - "without DTM", "HRTM", and "PDTM", using `libquantum` benchmarks



(a) Without DTM



(b) HRTM



(c) PDTM

Fig. 7. Comparisons among the three different schemes - "without DTM", "HRTM", and "PDTM", using `bzip2` and `libquantum` benchmarks

Table I. 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

the migration threshold. Also, priority scheduling threshold is 82℃. When predicted temperature is reached at priority scheduling threshold, the priority of application can be adjusted as lower value. All experiments are tested under ambient temperature control and fixed fan speed.

1. Digital Thermal Sensor for Core 2 Quad

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 (℃).

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 (3.12).

$$T_{core} = T_{junction} - DTS_{value} \tag{3.12}$$

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

2. Experimental Analysis

To demonstrate the proposed PDTM, we conduct our experiments with a single `SPEC2006` benchmark and a set of two `SPEC2006` benchmarks as shown in Table I.

Running the single benchmark, the presented PTDM can decrease 8% temperature in average (Figure 6), and reduces up to 5℃ in peak temperature with only under 1% performance overhead compared to Linux standard scheduler without DTM. Running two benchmarks simultaneously, the proposed PDTM can even lower about 10% temperature in average and reduces up to 3℃ in peak temperature while running a set of benchmarks with only under 8% performance overhead compared to Linux standard scheduler without DTM (Figure 7). As shown in Figure 8, the performance overhead caused by PDTM is only under 1% in average while running one benchmark. It means PDTM can be more effective to control temperature than Linux standard scheduler when temperature and workload is higher.

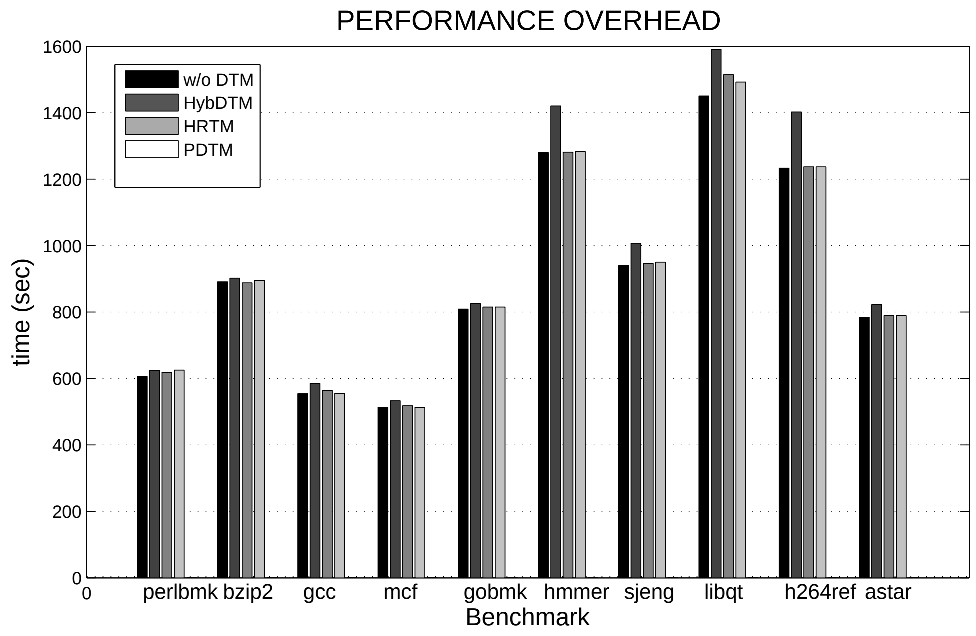


Fig. 8. Performance overhead:PDTM incurs only under 1% performance overhead in average while running single benchmark

In order to make comparison, we also rebuilt HybDTM [11] (the software scheme-changing priority) and HRTM [6] on our Quad-Core system. HybDTM uses priority-based scheme and HRTM uses migration-based scheme. HybDTM scheme relies on hardware performance counter, while HRTM relies on current temperature information. The experimental results show our PDTM outperforms HRTM in reducing average temperature by about 7%, performance overhead by 0.15%, and peak temperature by about 3°C. Additionally, our future temperature prediction model provides more accurate prediction with only less than 1.6% error as shown in Figure 9; on the other hand, the estimation model, introduced in HybDTM, has at most 5% average error. The main reason of the accuracy in our 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 controls the overall temperature lower than other schemes even in CPU intensive situation.

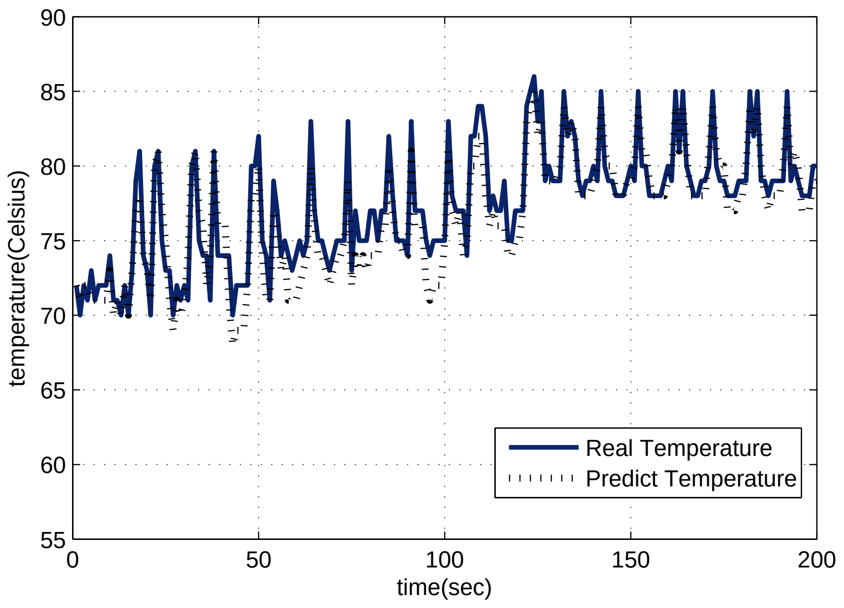


Fig. 9. The prediction model can estimate future temperature with less than 1.6% error on running `bzip2` benchmark

CHAPTER IV

TEMPERATURE MODELING AND MANAGEMENT BASED ON THERMAL CORRELATION AMONG NEIGHBORING CORES IN CMPS

Although, there have been a handful of studies using simple workload models, such as the average of workload and IPC (Instructions Per Cycle), for the Dynamic Thermal Management (DTM), the workload information is measured offline in their studies [6, 10]. We believe that it is necessary to develop an efficient online workload estimation scheme for DTM to be applicable to the real world applications which have variable workload behaviors and distinct thermal contributions to the increased chip temperature. Instead of conducting our DTM design upon the worst-case analysis, we drive our DTM design toward to the average-case thermal management in the CMP systems. Hence, there is a need to develop an efficient and practical DTM scheme by modeling the thermal correlation effect and distinguishing the thermal contributions from applications with different workload behaviors at runtime.

In this thesis, we propose the Thermal Correlative Dynamic Thermal Management (TCDTM) that incorporates three main components: Statistical Workload Estimation (SWE), Future Temperature Estimation Model (FTEM) and Temperature-Aware Thread Controller (TATC). The SWE utilizes the workload probability distribution to measure each running thread's workload behavior locally and overall workload behavior within each core globally. The representative workload is estimated by 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 among neighboring cores 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 FTEM 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, TATC 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 available within multicore systems to effectively lower peak temperature, avoid thermal emergency and provide thermal fairness with negligible performance overhead. To further demonstrate TCDTM's scalability and efficiency, especially to satisfy the demand of thermal control in the recent server environment, we implement and evaluate the proposed TCDTM in the real-world products, 4-core (Intel Quad Core Q6600) and 8-core (two Quad Core Intel Xeon E5310 processors) systems, running grouped applications ranged from multimedia application, popular server applications to several benchmarks without any additional hardware unit.

We conclude the main contributions of this proposed work as follows:

- We analyze and distinguish the different thermal impacts contributed by various applications with different dynamic workload behaviors via utilizing a statistic approach.

- We develop the thermal models to consider the thermal correlation among neighboring cores in CMP systems by profiling thermal parameters under the specific workload.

- Then, we propose an effective DTM, called TCDTM, which is applicable to the real-world applications having fluctuant workload behaviors and scalable to the real CMP machines without any additional component required.

- According to the experimental results, TCDTM reduces the peak temperature by up to 9.09% and 7.94% in our 4-core system and 8-core system with only 2.28% and 0.54% performance overhead respectively compared to the Linux standard scheduler.

- In average, TCDTM outperforms PDTM [18] and Thermal Balancing Policy [12] by 3.8% and 3.16% in lowering peak temperature with 0.3% and 37.6% less performance overhead respectively in our 4-core system; On the other hand, TCDTM outperform PDTM [18] and Thermal Balancing Policy [12] by 4.09% and 3.87% in lowering peak temperature with 0.09% more and 36.94% less performance overhead respectively in our 8-core system.

A. Statistical Workload Estimation

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. Finally, we show the effect of workload on the thermal parameters which will be used in the thermal model, as describe in Section 3.

1. 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 [19, 20]:

$$t_{app} = \frac{\sum_{i=1}^{n} CPI_i}{f^{CPU}}, \tag{4.1}$$

where n is the total number of instructions, CPI_i is the number of CPU clock cycle for the i_{th} instruction, and f^{CPU} is the CPU frequency.

We can define workload t_{app} by the ratio of the execution time, t_{app}, to given slack time, t_{max}, in Equation (4.2).

$$W_{app} = \frac{t_{app}}{t_{\max}} \times 100. \tag{4.2}$$

Actually, Linux kernel provides the ratio for each time interval. Therefore, we use the ratio to model a representative workload of an application as follows.

2. Representative 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 (4.3)

$$P(W_{app}^{min} \leq X \leq W_{app}^{max}) = \int_{W_{app}^{min}}^{W_{app}^{max}} F(x)dx, \tag{4.3}$$

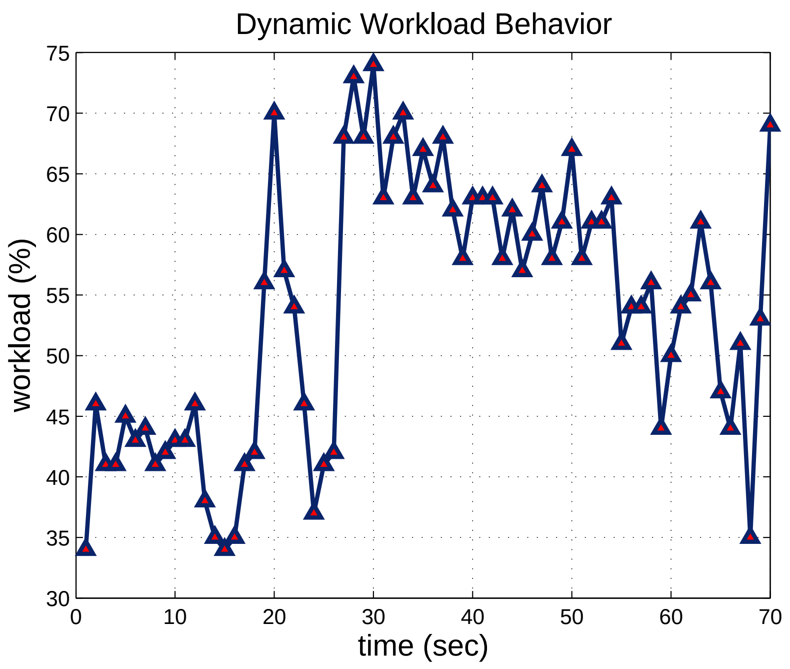
where X is in the interval $[W_{app}^{min}, W_{app}^{max}]$ and $F(x) = P(X \leq x) = \sum_{y:y\leq x} p(y)$. And W_{app}^{min} means 0% and W_{app}^{max} means 100% workload, respectively. To satisfy a various computational requirements, the representative should be decided by the probability requirements for application workload, W_{app}, in the window. Specifically, let ρ be the probability required for application workload in a window. In our observations, even dynamic workload of applications can be defined as the representative workload by a probability ρ as follow:

$$P[X \leq W_{app}] \geq \rho. \tag{4.4}$$

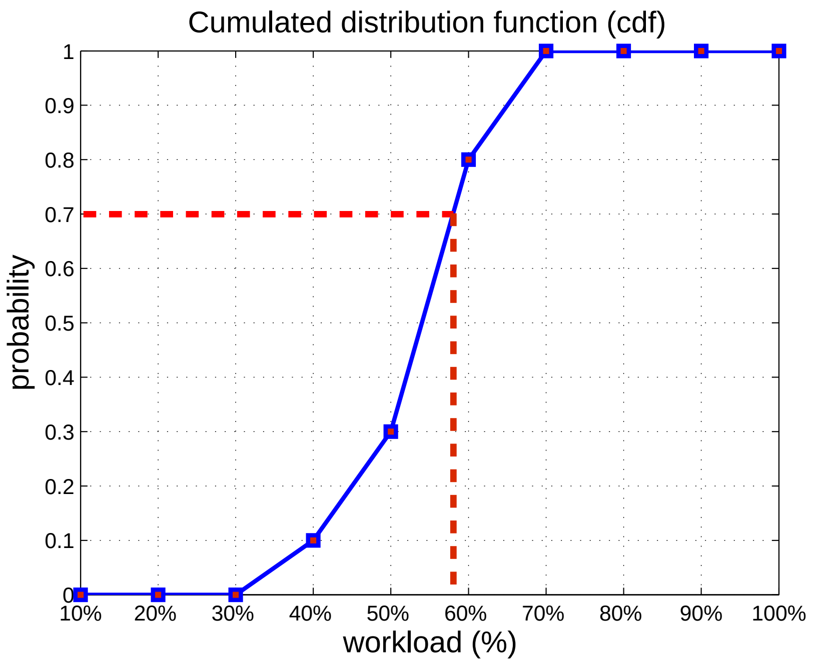
As shown in Figure 10, we can exploit the representative workload using *cdf* when playing a multimedia application. Moreover, in order to distinguish the threads with stable workload behaviors from those with highly unstable workload behaviors, the standard deviation, denoted as σ, is considered. In 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*.

$$\rho = \begin{cases} 0.5 & \sigma < 7.0 \quad \text{stable workload} \\ 0.7 & \sigma \geq 7.0 \quad \text{unstable workload} \end{cases}. \tag{4.5}$$

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 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 TCDTM 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. The detailed thread control policies will be explained in Section C.



(a) Dynamic workload behaviors



(b) Workload cumulative distribution function

Fig. 10. The representative workload is 58% when ρ is 0.7 in dynamic workload behavior

3. Workload Effect on Thermal Parameter b

In order to provide a thermal model of a processor, we should consider the relationship between temperature and workload of applications on the processor. We define $T(t)$ and $P(t)$ as temperature and power at time t, respectively, using Fourier's Law as the following [21, 22]:

$$T'(t) = P(t) - bT(t), \tag{4.6}$$

where b is a positive constant representing the power dissipation rate. If we define $f(t)$ as processor frequency at time t, power and processor frequency are relevant to the followings:

$$P(t) = a(f^{\alpha}(t)), \tag{4.7}$$

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 (4.6) using Equation (4.7) can be presented as follows:

$$T(t) = \int_{t_0}^{t} a(fr^{\alpha}(\tau)e^{-b(t-\tau)})d\tau + T_0 e^{-b(t-t_0)}. \tag{4.8}$$

We can derive the following equation if we maintain the frequency constant at $f(t)$ $= f_c$ during the time interval at $[t_0, t]$.

$$T(t) = \frac{a(f_c^{\alpha})}{b} + (T(t_0) - \frac{a(f_c^{\alpha})}{b})e^{-b(t-t_0)}, \tag{4.9}$$

where f_c is the current frequency on the processor. In order to determine thermal parameters, a and b, we assume $\alpha = 3.0$ [21], and then we can obtain the values for a and 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 [23]. 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 (4.10).

$$T_{new} = P \cdot R_{th} + (T_{old} - P \cdot R_{th})e^{\frac{-\Delta t}{R_{th} \cdot C_{th}}}, \tag{4.10}$$

where R_{th} is thermal resistance and C_{th} is thermal capacitance. With Equation (4.10) and (4.9), we can derive the thermal parameters a and b as follows:

$$a = \frac{1}{C_{th}},\ b = \frac{1}{R_{th} \cdot C_{th}} \tag{4.11}$$

By Equation (4.11), 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 [22]. Therefore, this value is practically constant in the same material. In contrast, the thermal parameter b is related to application's workload. This is because thermal resistance is in inverse proportional to power consumption. Hence, characterizing the workload behavior is critical for distinguishing the different threads' thermal effects. As shown in Figure 11, the workload dominates the temperature change in a core. Therefore, it is important to characterize application's workload in thermal control.

B. Thermal Model

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

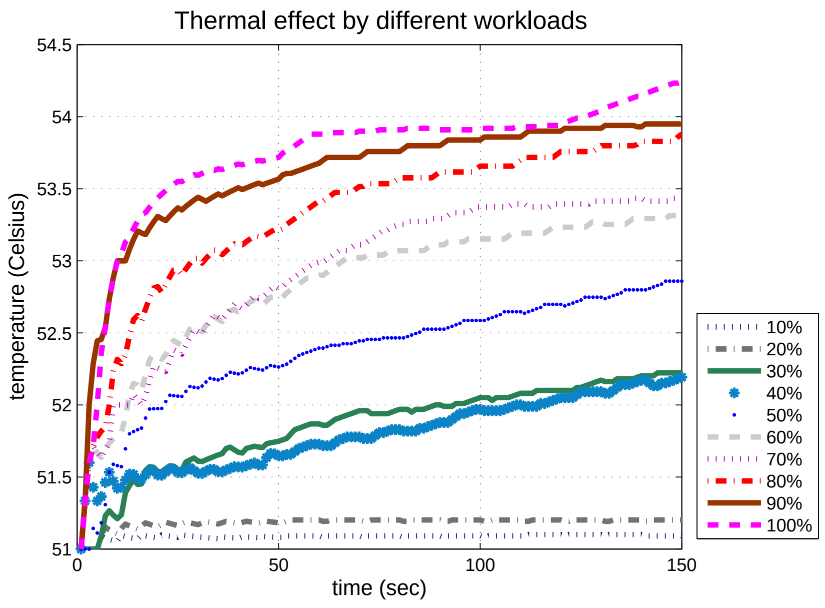


Fig. 11. Thermal effect of different workloads

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 [15]. Within those heat transfer equations, the rate of temperature change is proportional to the difference between the current temperature and the steady state. 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,

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{4.12}$$

where b is a thermal parameter 3. Solving Equation (4.12) with $T(0) = T_{init}$ and $T(\infty) = T_{ss}$, we could obtain

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{4.13}$$

Using Equation (4.13) 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 monitor several current temperature through the Digital Thermal Sensor (DTS) in each core. Thus, we calculate the thermal parameter b for the application within the core using Equation (4.13).

As the result, we obtain each core's respective value b and T_{ss} for the generated process in Table II 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 could estimate the core's future temperature ($T(t)$) after time t by Equation (4.13). we can notice that each core's thermal parameter b and T_{ss} are different even though the cores are within the same package as shown in Table II. 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. Thermal Effect according to Workload and Correlation among Cores

In this section, we first characterize the different thermal impacts contributed by different workloads and then model the thermal correlation among neighboring cores.

Table II. 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 III. 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

a. Thermal Impacts Contributed by Different Workloads

In the real world applications, the workload fluctuates, and each core's T_{ss} and b should be changed according to the variance of workload at runtime. Therefore, by running processes with several different workloads on each core in our 4-core system, we observe the relationship between workload and thermal parameter b, as well as T_{ss} in the Table III. To do that, we experiment on two different environments such as 4-core (Intel Quad Core Q6600) and 8-core (two Quad Core Intel Xeon E5310 processors) systems. We will explain in detail about our environments in Section 1.

b. 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} (affected by neighboring cores' temperature). Thus, we calculate new T'_{ss} according to own workload by the following Equation (4.14).

$$T'_{ss} = T^{w}_{ss} + T_{tc}. \tag{4.14}$$

First, we can obtain T^{w}_{ss} from Table III. 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 Figure 12, 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 the Equation (4.15) to obtain T_{tc} to model the thermal correlation impact from other cores.

$$T_{tc} = \sum_{i=2}^{n} \Delta T \times W_i, \tag{4.15}$$

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

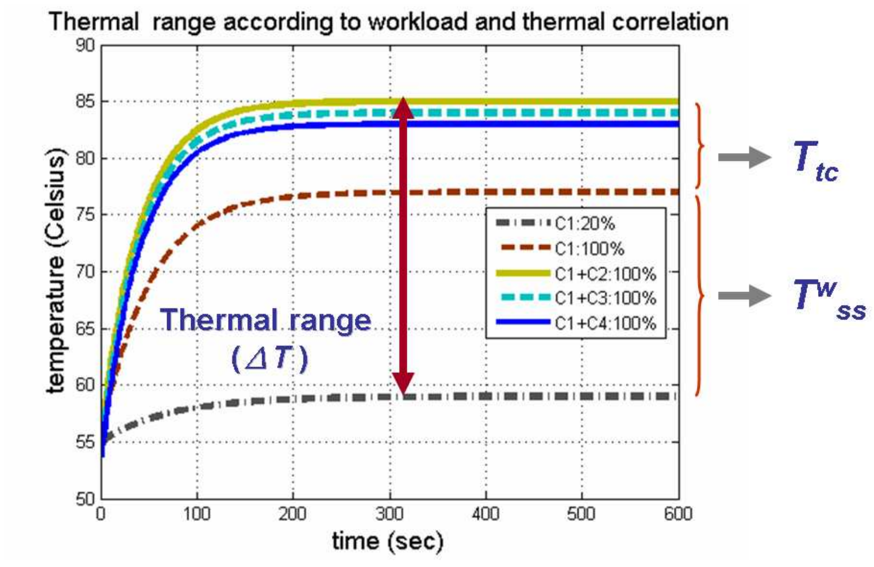


Fig. 12. The thermal range (ΔT) using T^w_{ss} and T_{tc} to calculate T'_{ss} for core 1

estimated described in Section 2. 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℃ and b as 0.0199 from Table III. Then, we calculate the thermal correlation effect from each neighboring core with 100% workload, as shown in Table IV.

Therefore, by Equation (4.14) and (4.15), the T'_{ss} can be obtained by the following equation:

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

Table IV. 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

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.

c. 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(4.16) and (4.17):

$$b_{tc} = \sum_{i=2}^{n} \Delta b \times W_i \tag{4.16}$$

$$b' = b^w + (b_{tc} \times \frac{(T'_{ss} - T_{cur})}{(T'_{ss} - T_{init})}), \tag{4.17}$$

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 (4.16), 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.

3. Future Temperature Estimation Model (FTEM)

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

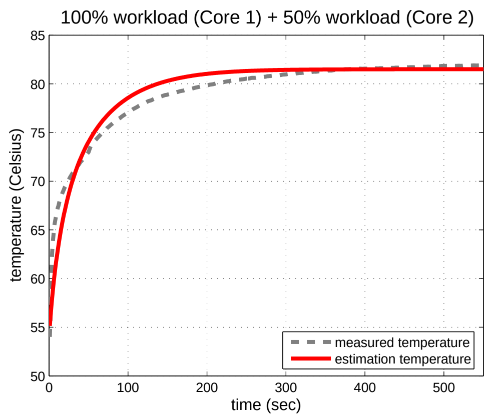
The original thermal models for estimating the future temperature at time t is improved from Equation (4.13) to the following Equation (4.18) for a specific core with workload estimation and thermal correlation by neighboring cores.

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

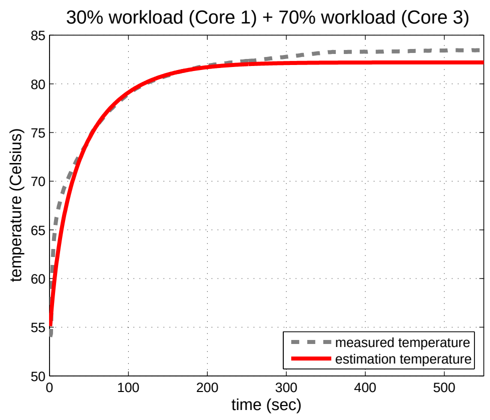
$$T'(t) = T^{w}_{ss} + T_{tc} - (T^{w}_{ss} + T_{tc} - T_{init}) \times e^{-(b^{w}_{tc} + (b_{tc} \times \frac{(T'_{ss} - T_{cur})}{(T'_{ss} - T_{init})})) \times t} \tag{4.18}$$

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 Figure 13. From Figure 13, 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.

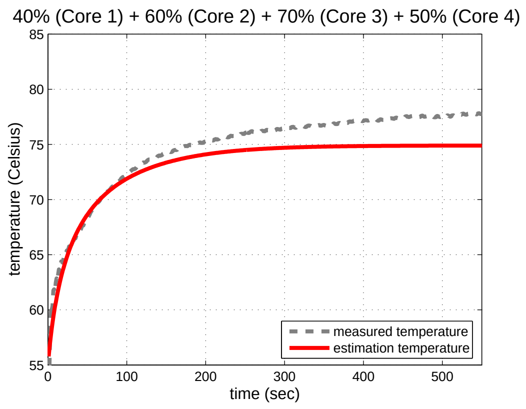
Moreover, in order to demonstrate that the improved thermal models can be effective even under the fluctuant workload, we also evaluate our thermal models by executing multimedia data, which generates two individual threads. We first calculate



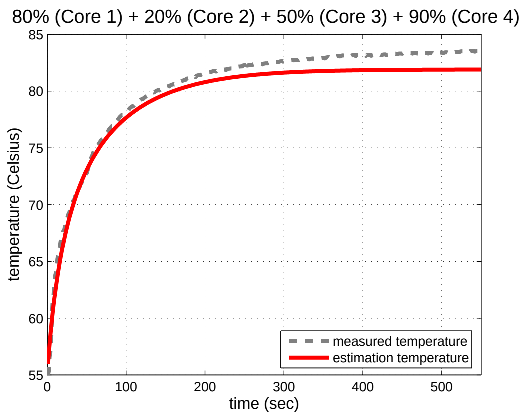
(a) 100% (Core 1) + 50% (Core 2)



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

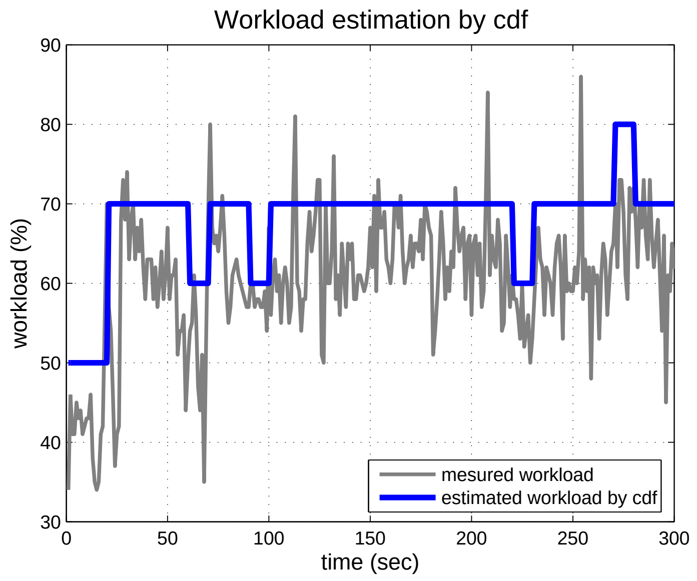


(c) variable workloads on all cores

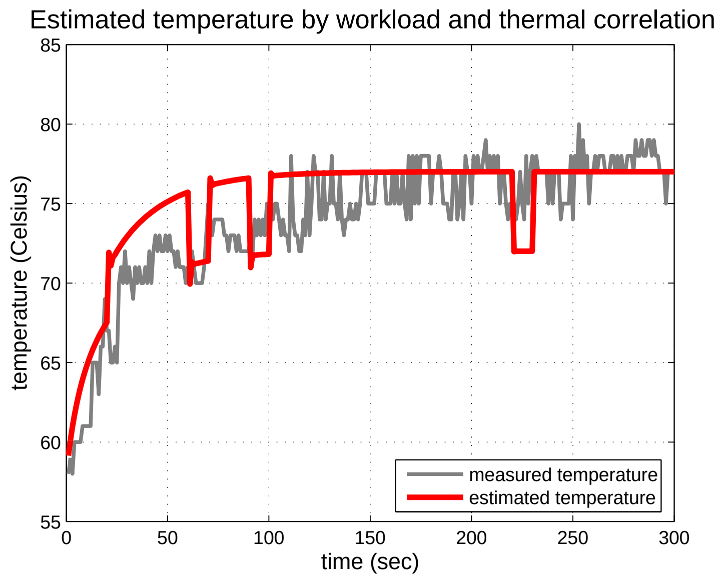


(d) variable workloads on all cores

Fig. 13. 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. 14. Validation of new thermal model with fluctuating workload: while playing the Transformer movie, the Mplayer software would generate two threads. One is the X windows deamon with stable workload, and the other one is for frame decoding with fluctuating workload

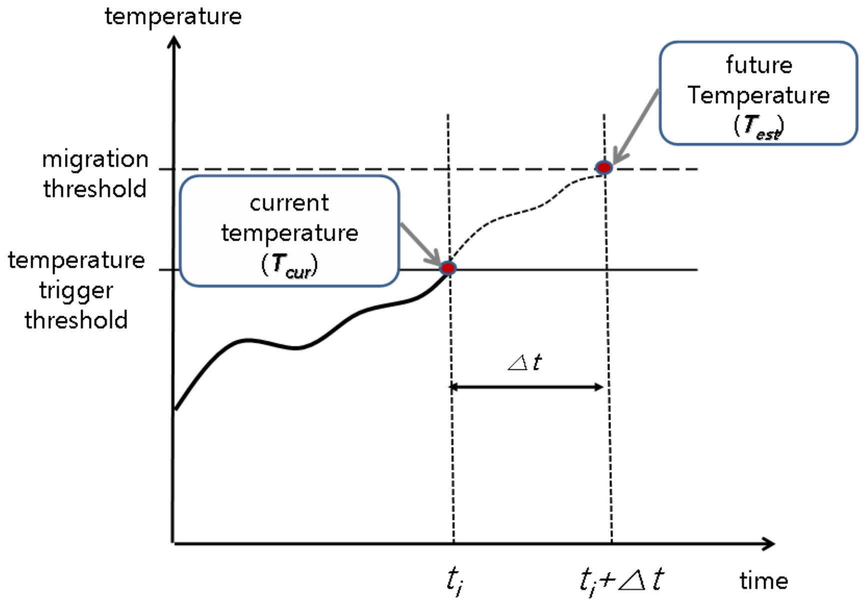

temperature
future Temperature (T_{est})
migration threshold
current temperature (T_{cur})
temperature trigger threshold
$\triangle t$
t_i
$t_i+\triangle t$
time

Fig. 15. Future Temperature Estimation Model (FTEM)

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 Figure 14(a), and the estimated temperatures compared with the monitored real temperature is shown in Figure 14(b). Thus, the results also demonstrate the the accuracy of our improved thermal models under fluctuant workloads considering both workload and thermal correlation from neighboring cores. Therefore, as shown in the Figure 15, the proposed Future Temperature Estimation Model (FTEM) 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 SWE. 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 TATC 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 TCDTM is able to adaptively cope the threads according to their different thermal effects, based on their representative workloads and neighboring thermal correlation effects. Consequently, TCDTM is capable to control the temperature at a desired level with ignorable performance overhead.

C. Temperature Correlative DTM

In this section, we introduce the system design and architecture of the proposed TCDTM. Moreover, we present how Thermal-Aware Thread Controller (TATC) utilize the workload behavior and thermal correlation information to achieve thermal balancing and lower the peak temperature will be explained in details.

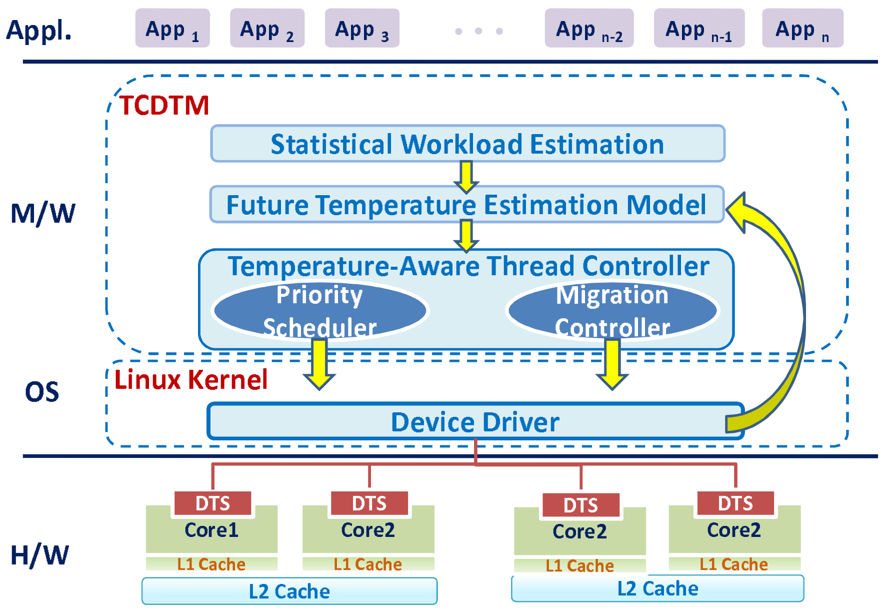


Fig. 16. The TCDTM system architecture

1. System Overview

Basically, the proposed Thermal Correlative Dynamic Thermal Management (TCDTM) consists of three major components: Statistical Workload Estimation (SWE), Future Temperature Estimation Model (FTEM) and Temperature-Aware Thread Controller (TATC). As shown in Figure 16, 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 the Digital Thermal Sensor (DTS) for monitoring each core's temperature, and temperature information would be used in the FTEM. As explained in the Section A and B, SWE 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 time duration (Δt) before temperature reaches the migration threshold and the future temperature (T_{est}). Hence, the TATC is able to react against to the thermal emergency appropriately according to the estimated information. In the following section, we discuss about the TATC in details.

2. Temperature-Aware Thread Controller (TATC)

To guarantee the thermal safety, the Thermal-Aware Thread Controller (TATC) consists of two schemes: the priority scheme and migration scheme. Basically, When current temperature reaches the trigger threshold, the SWE starts to monitor the application's workload behavior and calculate the representative workloads through *cdf* for the running thread and core. Hence, as shown in Figure 17, 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, TATC 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℃ in our observations, TATC deals with the threads with workload higher than 30% to reduce the performance overhead. In TATC, migration can be adopted in most cases, unless all the cores' temperature reaches the priority scheduling threshold. In this case, TATC 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 can not effectively reduce the core temperature if all the core temperatures are near to the maximum allowable temperature.

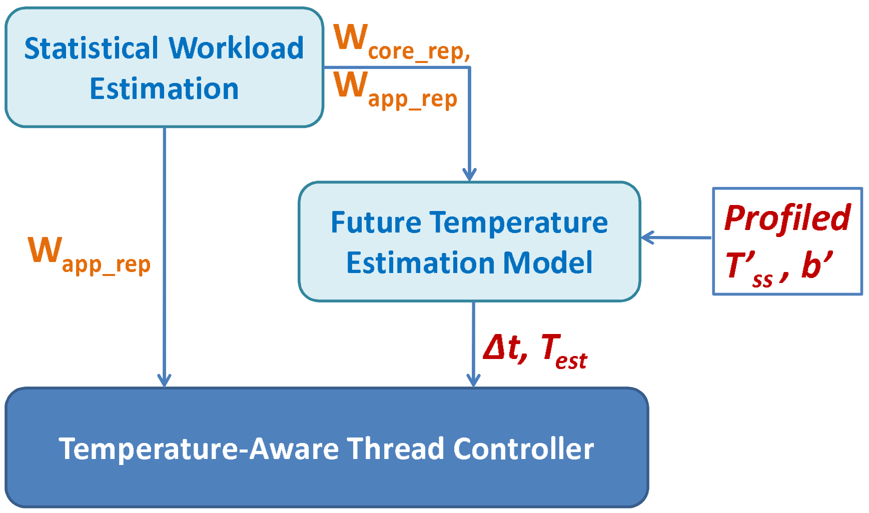

Statistical Workload Estimation
W_core_rep,
W_app_rep
W_app_rep
Future Temperature Estimation Model
Profiled T'_ss , b'
Δt, T_est
Temperature-Aware Thread Controller

Fig. 17. TCDTM control flow: The T'_{ss}, and b' are profiled according to the application's workload. After future temperature estimated, the time duration (Δt) before temperature reaches the migration threshold and future temperature (T_{est}) would be passed to TATC for reaction

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, TATC is able to effectively reduce the peak temperature of each core and achieve thermal balancing with ignorable performance overhead.

D. Experimental Results and Analysis

In this section, the detailed experimental environment and results are explained, along with the analysis of the efficiency and effectiveness of the proposed TCDTM.

1. Experimental Environment

Table V. Experimental systems descriptions

	System I	System II
The number of cores	4 cores	8 cores
Processor	Intel Quad Core Q6600	two Quad Core Intel Xeon E5310
Memory Size	1 GB	1 GB
Operating System.	SUSE 10.3 (Kernel Version: 2.6.22)	RedHat Enterprise 4 (Kernel Version: 2.6.9)

Since the thermal correlation effect is difficult to be simulated, we insist to implement and evaluate the proposed TCDTM in the real CMP products. In order to estimate each core's working temperature individually, we develop a specific device driver for accessing the Digital Thermal Sensor (DTS) on Linux in our multicore systems. The trigger point of these thermal sensors is not programmable by software since it is set during the fabrication of the processor [17]. To evaluate the scalability, we conduct our experiments in two multicore systems as shown in Table V. Moreover, the implementation parameters of TCDTM for the two systems are provided in Table VI.

Table VI. Experimental parameters

	System I	System II
Initial Temperature	54 ℃	52 ℃
Trigger Threshold	65 ℃	55 ℃
Priority Scheduling Threshold	95 ℃	70 ℃
The highest temperature	98 ℃	74 ℃

In order to demonstrate the applicability of the proposed TCDTM to various applications with different workload behaviors, we create several scenarios and test groups which include stable, fluctuant and combined workload behaviors as shown in Table VII. We choose *bzip2* and *libquantum* from SPECCPU2006 benchmark, *vacation* from STAMP benchmark [24], and a multimedia application - Mplayer as our test applications. We select *bzip2* because it is both CPU and memory intensive, while *libquantum* is only CPU intensive. Furthermore, *vacation* is a client/server travel reservation system benchmark that is appropriate to present the demand of thermal control in the server systems. For the application with fluctuant workload, we use Mplayer to execute the "Transformers" video clip encoded by H.264. 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 is the decoding thread whose workload is fluctuant between 40% and 70%.

To compare the effectiveness and efficiency of the proposed TCDTM, we also rebuild the Predictive Dynamic Thermal Management (PDTM) [18] and Thermal Balancing Policy (TBP) [12] in our systems. All the experiments in this research are under ambient temperature control, and the speed of cooling fan is also fixed.

Table VII. Application test scenarios

	Category	Test group	Details
Scenario A	stable workload	1	*bzip2*
		2	*bzip2* + *libquantum*
		3	*libquantum* + *vacation*
Scenario B	fluctuant workload	4	Multimedia
Scenario C	combined workload	5	*bzip2* + Multimedia
		6	*libquantum* + *vacation* + Multimedia
		7	*bzip2* + *libquantum* + *vacation* + Multimedia

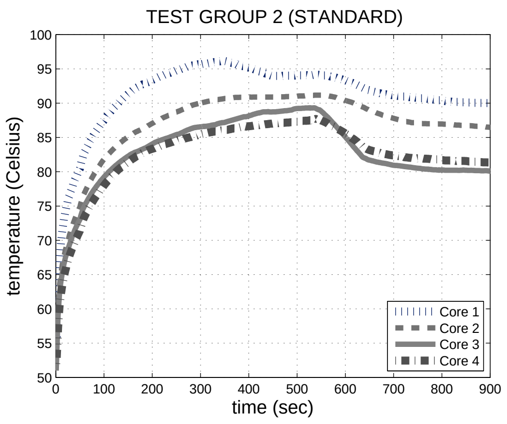
2. Analysis and Evaluation

Here, we select several representative test groups from each scenario to present as figures[1] to demonstrate the proposed TCDTM's effectiveness in thermal control. On the contrary, the performance overhead caused by DTM is discussed for all test groups. Here, we would discuss the effectiveness and efficiency by individual group in our two systems.

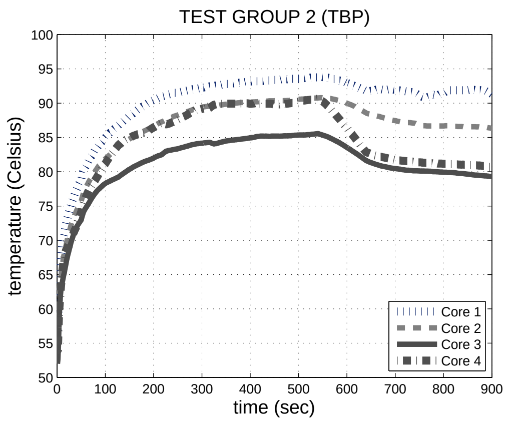
[1]All the temperature figures have been processed by the smooth function in Matlab for clearness.

a. System I (4-core System)

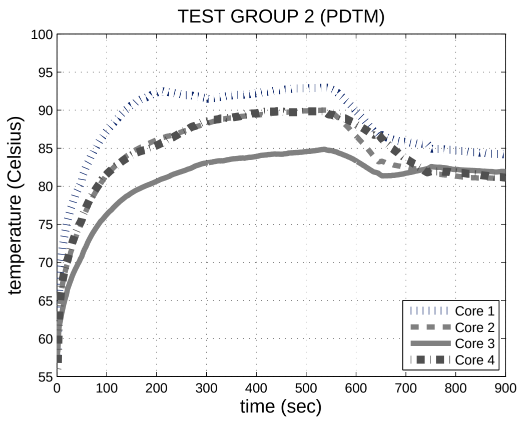
The different scenarios' experimental results in System I are discussed below:



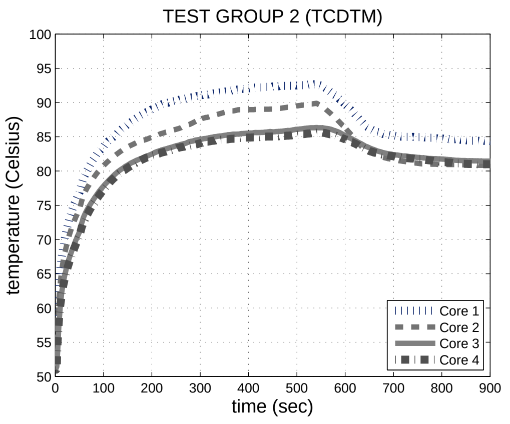
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



(d) Thermal Correlative DTM

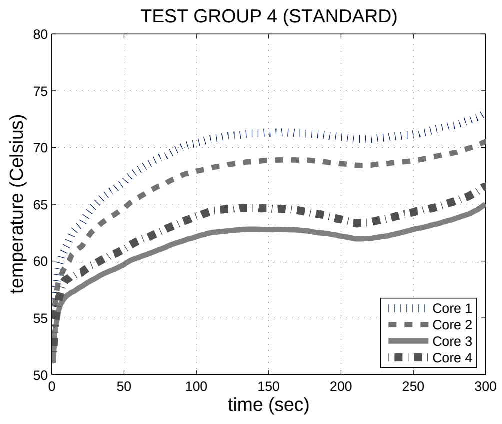
Fig. 18. DTM evaluation in 4-core system for stable workload behaviors: Test Group 2 (*libquantum* + *vacation*)

1. Scenario A (Stable Workload)

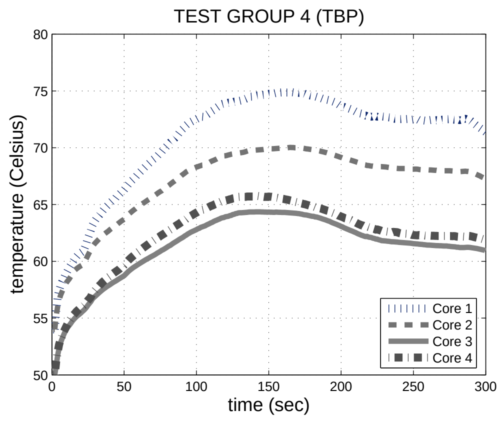
As shown in Figure 18, all the DTMs have lower peak temperature compared to the Linux Standard Scheduler in Test Group 2. Compared to the Linux Standard Scheduler, both TCDTM and PDTM reduce the peak temperature by 3.13%, while TBP reduces by 2.08% as shown in Table VIII. For the performance overhead evaluation in Figure 21, the TCDTM and PDTM present less than 0.46% performance overhead compared to the Linux Standard Scheduler in Test Group 2, while TBP incurs 6.64%. This is also the reason why the temperature in TBP doesn't decrease obviously after executing 600 seconds. Since there are only two threads running in the systems, the thermal correlation effect is minor. Moreover, both of the treads maintain 100% workload stably, and the difference of thermal behaviors could then be ignored. Therefore, PDTM presents the similar effectiveness in thermal control compared to TCDTM.

2. Scenario B (Fluctuant Workload)

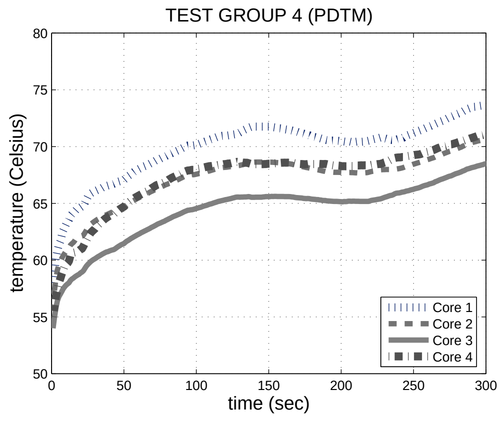
As shown in Figure 19, we first notice that TCDTM presents a smoother temperature pattern, and provides better thermal fairness by having narrower temperature gaps among all cores in the Test Group 4. As shown in Table VIII, TCDTM reduce the peak temperature by 1.35% compared to Linux Standard Scheduler, 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 TCDTM 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



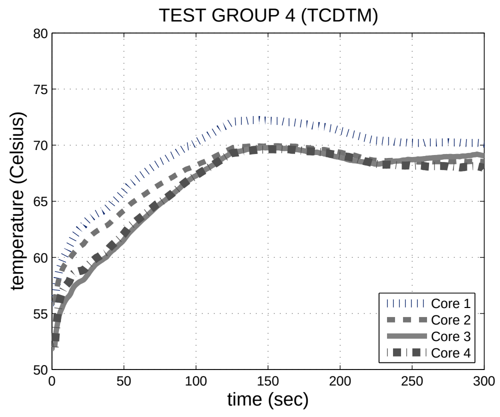
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



(d) Thermal Correlative DTM

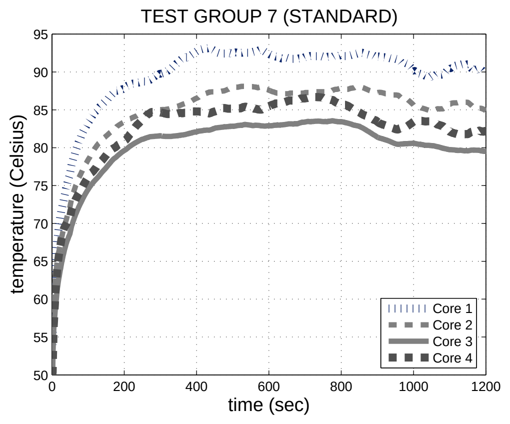
Fig. 19. DTM evaluation in the 4-core system for fluctuating workload behaviors: Test Group 4 (Multimedia)

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. 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.

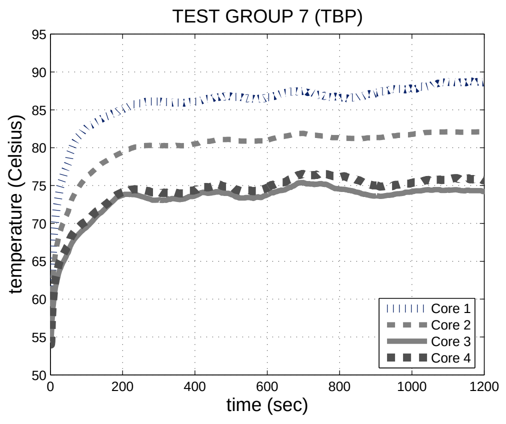
About the performance overhead shown in Figure 21 and Table VIII in Test Group 4, the performance overhead is not available, because there is no obvious frame drop among 4 different schemes. This is due to the high CPU frequency in our test environment. In [25], the authors have demonstrated that 1Ghz CPU frequency is enough for 96% frames for decoding in H.264 codec. Therefore, we cannot compare the performance overhead among the 4 schemes in Test Group 4.

3. Scenario C (Combined Workload)

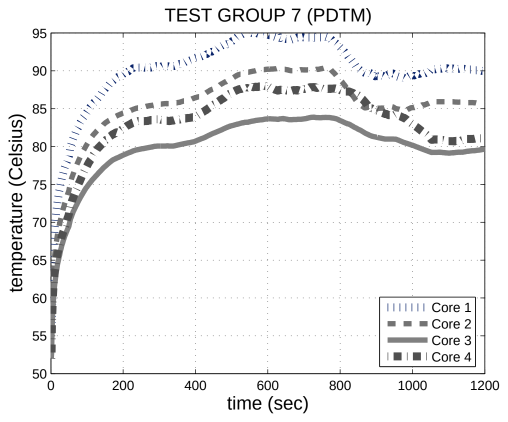
In the Test Group 7, since the number of threads is more than the number of cores, each core runs at least one thread at most time. This is important for the evaluation of the proposed TCDTM, because this group consists of applications with different workload behaviors, and the thermal correlation among cores is severe. From the Figure 20, the proposed TCDTM can effectively lower down the peak temperature by 2.15% compared to the Linux Standard Scheduler with only 6.27% performance overhead, as shown in Table VIII. One may notice that the peak and average temperature in TBP is much lower and smoother than other schemes. However, this



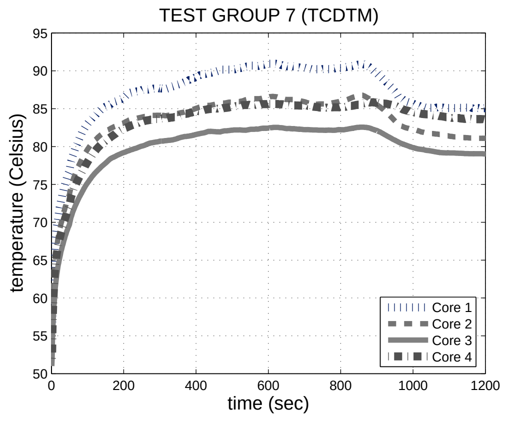
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



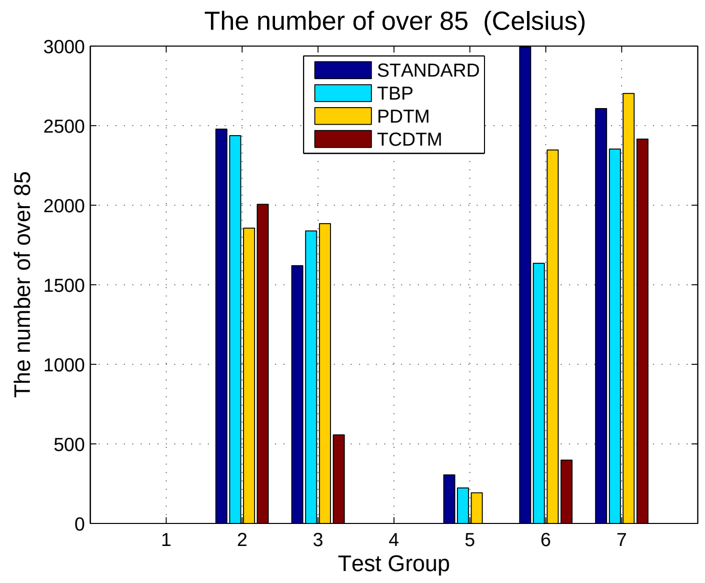
(d) Thermal Correlative DTM

Fig. 20. DTM evaluation in 4-core system for combined workload behaviors: Test Group 7 (*bzip2* + *libquantum* + *vacation* + Multimedia)

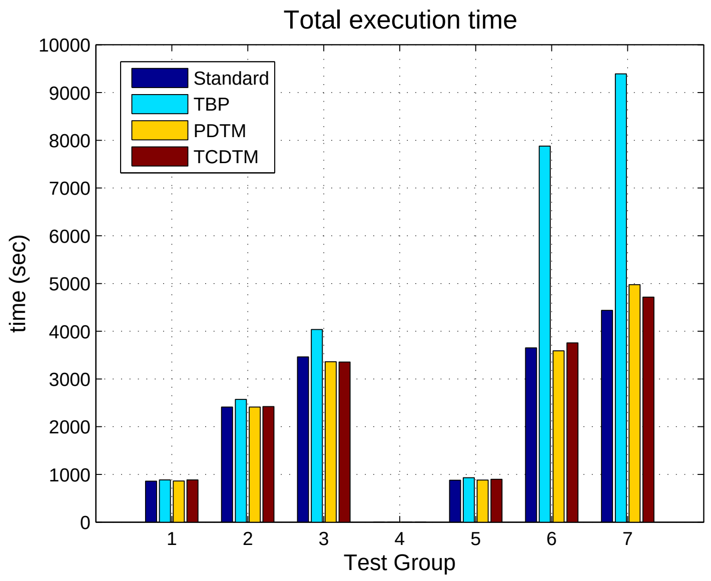
cannot demonstrate that TBP outperforms others in thermal control, because TBP incurs over two times performance overhead in Figure 21, and this is much worse than acceptable for the efficiency. Moreover, the peak temperature in PDTM is even 3.23% higher than that in Standard Scheduler with 12.20% performance overhead, and this is due to that PDTM is not aware of the distinct workload behaviors among threads and the significant thermal correlation issues, so PDTM cannot accurately predict the core's future temperature and trigger migration at the correct time. On the contrary, TCDTM is capable to distinguish the different workload behaviors and estimate the thermal correlation effect. Therefore, TCDTM outperforms TBP and PDTM in both thermal control and efficiency.

Table VIII. Experimental results compared to Linux Standard Scheduler in System I (4-core system): (R.P.T. : Reduced Peak Temperature; P.O. : Performance Overhead)

	TBP		PDTM		TCDTM	
Test Group	R.P.T. (%)	P.O. (%)	R.P.T. (%)	P.O. (%)	R.P.T. (%)	P.O. (%)
1	6.25	3.03	7.50	0.47	8.75	3.26
2	2.08	6.34	3.13	0	3.13	0.46
3	0	16.58	1.10	−2.95	4.40	−3.12
4	−1.35	N/A	−1.35	N/A	1.35	N/A
5	2.27	5.80	1.14	0.34	9.09	2.28
6	2.17	115.86	2.17	−1.67	7.61	2.93
7	3.23	111.74	−3.23	12.20	2.15	6.27



(a) The number of over 85 ℃

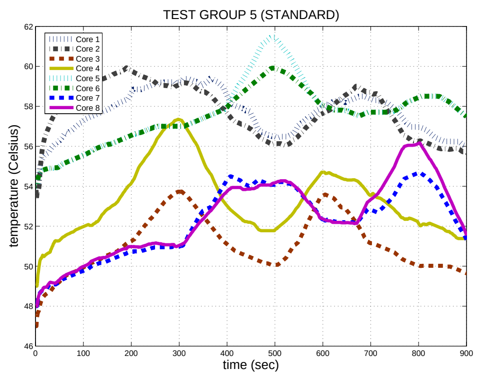


(b) Total execution time

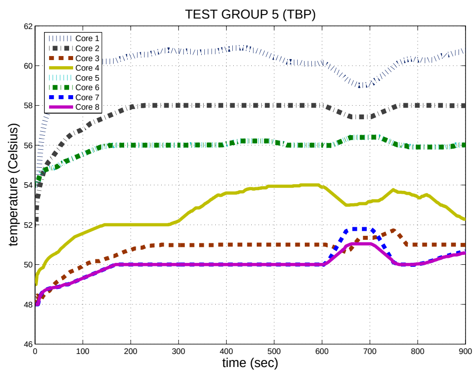
Fig. 21. Evaluation of DTMs' effectiveness and efficiency in System I

b. System II (8-core System)

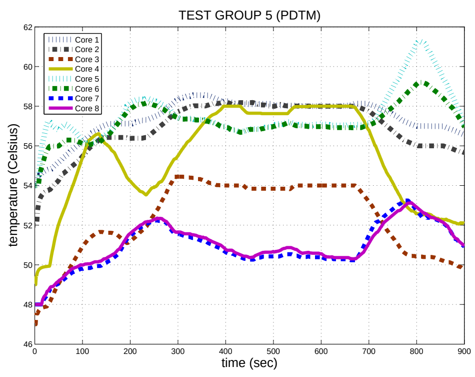
The different scenarios' experimental results in System II are discussed below:



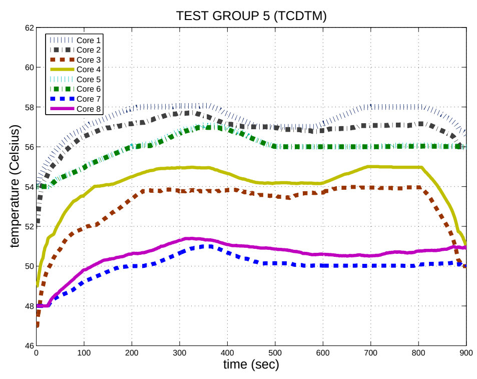
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



(d) Thermal Correlative DTM

Fig. 22. DTM evaluation in 8-core system for combined workload behaviors: Test Group 5 (*bzip2* + Multimedia)

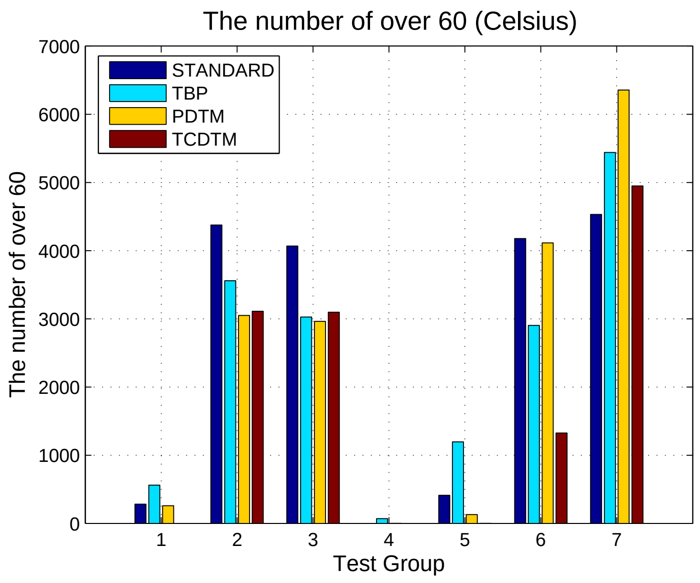
1. Scenario C (Combined Workload)

In System II, the proposed TCDTM outperforms PDTM and TBP in both temperature control effectiveness and efficiency in the Group Test 5. As shown in Figure

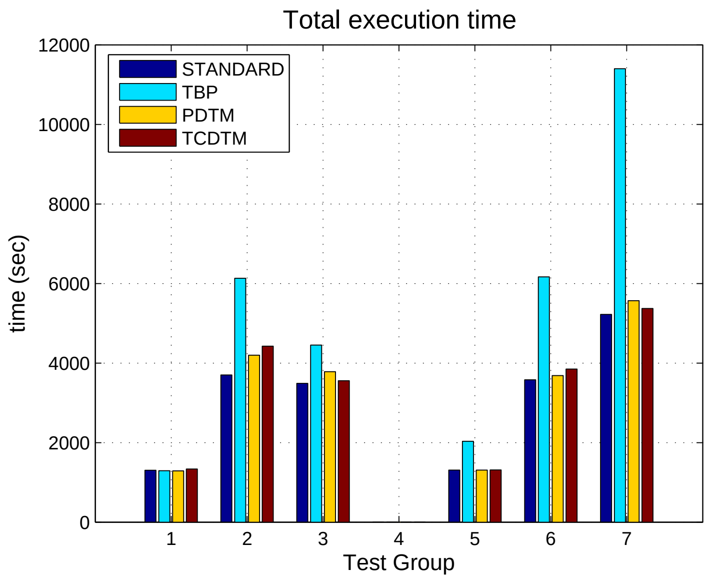
22, TCDTM presents a much smoother temperature pettern than the Linux Standard Scheduler. In Table IX and Figure 23, TCDTM reduces the peak temperature by 7.94% with 0.54% performance overhead compared to Linux Standard Scheduler, while PDTM and TBP reduce peak temperature by 1.59% and 3.17% with 0.15% and 55.5% performance overhead respectively. Although TBP decreases the peak temperature and presents smoother thermal pattern compared to the Standard Scheduler, TBP also offers impractically huge performance overhead. Moreover, the exchanged threads cannot effectively reduce core 1's temperature. Since PDTM is not aware of the different thermal effect contributed by applications with different workload behaviors and the thermal correlation effect, PDTM cannot accurately predict the temperature and react in time. Therefore, around 800 seconds in Figure 22, PDTM fails to control the temperature under the desired level.

Table IX. Experimental results compared to Linux Standard Scheduler in System II (8-core system): (R.P.T. : Reduced Peak Temperature; P.O. : Performance Overhead)

	TBP		PDTM		TCDTM	
Test Group	R.P.T. (%)	P.O. (%)	R.P.T. (%)	P.O. (%)	R.P.T. (%)	P.O. (%)
1	1.61	−0.77	1.61	−1.23	8.06	2.38
2	−4.76	65.73	−4.76	13.49	3.17	19.62
3	−1.61	27.59	0	8.42	−1.61	1.83
4	−1.69	N/A	−1.69	N/A	5.08	N/A
5	3.17	55.50	1.59	0.15	7.94	0.54
6	−1.61	72.39	0	2.99	0	7.58
7	1.52	118.30	−1.52	6.63	1.52	2.89



(a) The number of over 60 ℃



(b) Total execution time

Fig. 23. Evaluation of DTMs' effectiveness and efficiency in System II

c. Analysis

For all the experiments results, we summary the thermal control effectiveness and performance overhead compared to the Linux Standard Scheduler in the Table VIII and Table IX. Among all the results, TCDTM reduces the peak temperature by up to 9.09% in Test Group 5 in System I and up to 7.94% in System II with only 2.28% and 0.54% performance overhead respectively. In TBP, the high performance overhead may due to the high migration frequency, since the migration is easier to be triggered once the current temperature reach either upper or lower threshold. Moreover, in TBP, the temperature gaps among each core is wider than other schemes, because each core has different T_{ss} and thermal value b, and the exchanged threads can still increase the overheated core's temperature. In average, TCDTM outperforms PDTM and TBP by 3.8% and 3.16% in lowering peak temperature with 0.3% and 37.6% less performance overhead respectively in System I; On the other hand, TCDTM outperform PDTM and TBP by 4.09% and 3.87% in lowering peak temperature with 0.09% more and 36.94% less performance overhead respectively in System II.

Therefore, TCDTM outperforms other DTMs in both thermal control and performance efficiency under several different experimental scenarios, because TCDTM is capable to distinguish the different thermal behaviors of various applications with different workload behaviors, as well as aware of the thermal correlation among neighboring cores to react against thermal emergency immediately.

CHAPTER V

CONCLUSION

Due to the ever-increasing power density and current leakage, thermal impact has become critical and needs to be addressed immediately in the modern chip design. Since the cost and complexity are the major challenges in designing thermal packaging for thermal control in CMPs, an efficient Dynamic Thermal Management (DTM) is essential in the design of the high-performance microprocessors. Therefore, in this cooperative research work, we have have proposed two modern Dynamic Thermal Managements, Predictive Dynamic Thermal Management (PDTM) and Thermal Correlative Dynamic Thermal Management (TCDTM),to efficiently control the chip operation temperature under the desired threshold in the Chip multiprocessors (CMPs) systems. The conclusion of each proposed work is given in the following sections:

A. Predictive Dynamic Thermal Management

In the proposed PDTM, we present an advanced future temperature prediction model for multicore systems to predict each core's future temperature with only 1.6% error in average, and evaluated 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). Compared against Linux standard scheduler, PDTM can decrease average temperature about 10%, and peak temperature by 5C with negligible impact of performance under 1%, while running single SPEC2006 benchmark. Moreover, our PDTM outperforms HRTM in reducing average temperature by about 7% and peak temperature by about 3°C with performance overhead by 0.15% when running single benchmark. Most importantly, there is no additional hardware unit required for our prediction models and scheduler.

B. Thermal Correlative Dynamic Thermal Management

Moreover, 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 Thermal Correlative Dynamic Thermal Management (TCDTM), on the real-world CMP products. Since the significant variations in the thermal behaviors among different applications and the severe thermal correlation effect among multi cores 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 TCDTM.

To demonstrate the scalability and effectiveness, we implement and evaluate the proposed TCDTM in the 8-cores (two Quad Core Intel Xeon E5310 processors) and 4-cores (Intel Quad Core Q6600) systems running grouped multimedia application and benchmarks. According to the experimental results, TCDTM reduces the peak temperature by up to 9.09% in our 4-cores system and up to 7.94% in 8-cores system with only 2.28% and 0.54% performance overhead respectively compared to the Linux standard scheduler. Moreover, TCDTM also outperforms PDTM and Thermal Balancing Policy in both thermal control effectiveness and reducing the caused performance overhead.

This is the first study addressing the neighboring thermal correlation effect in CMP systems for Dynamic Thermal Management. We would like to present this work to discover the thermal nature and build a study foundation for Dynamic Thermal Management in Chip Multiprocessor systems in the future.

REFERENCES

[1] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *HPCA ’01: Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA ’01)*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 171–182.

[2] S. Gunther, F.Binns, D.Carmean, and J.Hall, “Managing the impact of increasing microprocessor power consumption,” *Intel Technology Journal*, vol. 5, no. 1, 2001, http://www.intel.com/technology/itj/archive/2001.htm.

[3] S. Heo, K. Barr, and K. Asanovic, “Reducing power density through activity migration,” in *ISLPED ’03: Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED ’03)*. New York, NY, USA: ACM, 2003, pp. 217– 222.

[4] K. Skadron, M.Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture: Modeling and implementation,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, pp. 94 – 125, 2004.

[5] J. Srinivasan and S. Adve, “Predictive Dynamic Thermal Management for Multimedia Applications,” in *ICS ’03: Proceedings of the 17th annual International Conference on Supercomputing (ICS ’03)*. New York, NY, USA: ACM, 2003, pp. 109 – 120.

[6] 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 *ISPLOS ’04: International Conference on Architectural Support for Programming*

Languages and Operating Systems (ASPLOS-XI). New York, NY, USA: ACM, 2004, pp. 260 – 270.

[7] 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: Circuits and Systems*, vol. 34, no. 12, pp. 1153–1165, 2003.

[8] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *ISCA '03: Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-30).* Washington, DC, USA: IEEE Computer Society, 2003, pp. 2– 13.

[9] K. Skadron, "Hybrid architectural dynamic thermal management," in *Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, vol. 1. Washington, DC, USA: IEEE Computer Society, 2004, pp. 10– 15.

[10] J. Choi, C. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," in *ISLPED '07: Proceedings of the 2007 International Symposium on Low Power Electronics and Design (ISLPED '07).* New York, NY, USA: ACM, 2007, pp. 213 – 218.

[11] A. Kumar, L. Shang, L. Peh, and N. K. Jha, "Hybdtm: A coordinated hardware-software approach for dynamic thermal management," in *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation (DAC '06).* New York, NY, USA: ACM, 2006, pp. 548– 553.

[12] 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 *DATE '08: Proceedings of Design, Automation and*

Test in Europe (DATE '08). Washington, DC, USA: IEEE Computer Society, 2008, pp. 734–739.

[13] P. Michaud, A. Seznec, D. Fetis, Y. Sazeides, and T. Constantinou, "A study of thread migration in temperature-constrained multicores," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 2, 2007, http://doi.acm.org/10.1145/1250727.1250729.

[14] X. Chen, "Recursive least-squares method with membership functions," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 2004, pp. 1962– 1966.

[15] F. Kreith and M. S. Bohn, *Principles of Heat Transfer.* Florence, KY, USA: CENGAGE-Engineering, 2000.

[16] D. Bovet and M. Cesati, *Understanding the Linux Kernel.* Sebastopol, CA, USA: O'Reilly Media, Inc, 2005.

[17] "Intel 64 and ia-32 architectures software developer's manual," http://support.intel.com/design/processor/manuals/.

[18] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *DAC '08: Proceedings of the 45th Annual Conference on Design Automation (DAC '08).* Washington, DC, USA: IEEE Computer Society, 2008, pp. 734–739.

[19] J. Hennessy and D. Patterson, *Computer Architecture - A Quantitative Approach.* San Fransisco, CA, USA: Morgan Kaufmann Publishers, Inc, 1996.

[20] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *ISLPED '04: Proceedings of the 2004*

International Symposium on Low Power Electronics and Design (ISLPED '04). New York, NY, USA: ACM, 2004, pp. 174 – 179.

[21] N. Bansal, T.Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS '04).* Washington, DC, USA: IEEE Computer Society, 2004, pp. 520–529.

[22] J.E.Sergent and A.Krum, *Thermal Management Handbook.* Columbus, OH, USA: McGraw-Hill, 1998.

[23] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," in *HPCA '02: Proceedings of the 8th International Symposium on High Performance Computer Architecture (HPCA '02).* Washington, DC, USA: IEEE Computer Society, 2002, pp. 17–28.

[24] C. C. Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis, and K. Olukotun, "An effective hybrid transactional memory system with strong isolation guarantees," in *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA-34).* New York, NY, USA: ACM, 2007, pp. 69 – 80.

[25] I. Yeo and E. J. Kim, "Hybrid dynamic thermal management based on statistical characteristics of multimedia applications," in *ISLPED '08: Proceedings of the 2008 International Symposium on Low Power Electronics and Design (ISLPED '08).* New York, NY, USA: ACM, 2008.

VITA

Chih-Chun Liu has received his B.S. degree from the Information Engineering and Computer Science Department at Feng-Chia University in Taiwan, in 2003. He entered the computer science program at Texas A&M University as a master's degree student in 2006, joining the High Performance Computing Laboratory led by Dr. Eun Jung Kim. Chih-Chun received his Master of Science degree in August 2008. Chih-Chun's research areas of interest include Dynamic Thermal Management, Chip MultiProcessor and Multimedia.

His email is chihchun@cs.tamu.edu, and his address is as follows:

Chih-Chun Liu

Department of Computer Science

Texas A&M University

College Station, Tx-77843-3112

The typist for this thesis was Chih-Chun Liu.