Predictive Dynamic Thermal Management for Multicore Systems

Inchoon Yeo, Chih Chun Liu, and Eun Jung Kim
Department of Computer Science
Texas A&M University
College Station, TX 77840
{ryanyeo, chihchun, ejkim}@cs.tamu.edu

ABSTRACT

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 this paper, we propose a Predictive Dynamic Thermal Management (PDTM) based on Application-based Thermal Model (ABTM) and Core-based Thermal Model (CBTM) in the multicore systems. ABTM predicts future temperature based on the application specific thermal behavior, while CBTM estimates core temperature pattern by steady state temperature and workload. The accuracy of our prediction model is 1.6% error in average compared to the model in HybDTM [8], which has at most 5% error. 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. PDTM enables the exploration of the tradeoff between throughput and fairness in temperature-constrained multicore systems. We implement PDTM on Intel's Quad-Core system with a specific device driver to access Digital Thermal Sensor (DTS). Compared against Linux standard scheduler, PDTM can decrease average temperature about 10%, and peak temperature by 5°C with negligible impact of performance under 1%, while running single `SPEC2006` benchmark. Moreover, our PDTM outperforms HRTM [10] in reducing average temperature by about 7% and peak temperature by about 3°C with performance overhead by 0.15% when running single benchmark.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Reliability, Availability, and Serviceability

General Terms

Design, Experimentation, Temperature, Performance

Keywords

Dynamic Thermal Management, Operating System, Temperature

1. 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 [3]. Hence, there is a need to control temperature at all levels of system design.

Recently, many hardware and software-based Dynamic Thermal Management (DTM) [3, 5, 6, 13, 14] techniques have been proposed in sense of that they, except [14], start to control the temperature after the current temperature reaches at 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 [10]. However, neighboring thermal effect and application thermal behavior are not considered in prior works. Due to packaging technology in 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 [11]. There are significant variations in the thermal behavior among different applications [11, 14].

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 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 `SPEC2006` benchmarks simultaneously show the proposed 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 [10], 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 paper 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 simultaneously.
- 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.

The remainder of the paper is organized as follows : The existing DTM will be introduced in Section 2. Section 3 provides the explanation of proposed temperature prediction model-ABTM and CBTM in detail. In Section 4, we explain the system overview of PDTM and how to apply PDTM in multicore systems. In Section 5, the implementation and analysis results are discussed and conclusions are provided in Section 6.

2. RELATED WORK

Several thermal control techniques have been proposed and applied in modern processors via either hardware-based or software mechanisms [3, 12]. 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. Therefore, as the multicore processors become popular, some software-based mechanisms, such as power density management in a CMP has been studied in [10]. 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 [9]. In [9], 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. However, both of these two works above are based on simulated results, and neglect the thermal-correlation between cores. The power dissipated by the rest of the chip is assumed to be negligible. Most importantly, the migration action in [9] above is triggered by the current temperature (when the temperature is higher than maximum allowed temperature) in these two papers; however, instead of considering the current temperature, we believe that an accurate future temperature prediction model could perform better in lowering the peak temperature.

In [8], 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, we propose PDTM which includes an advanced future temperature prediction model with very low performance overhead for the real-world product (Intel's Quad-Core). Rather than using the performance counter for temperature, we utilize the regression analysis for application-based thermal behavior as fine-grained scheme, and core-based thermal behavior as coarse-grained scheme to provide very accurate temperature prediction model for DTM.

3. 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 paper.

3.1 The Application-based Thermal Model

The Application-based Thermal Model (ABTM) accommodates short-term thermal behavior in order to predict the

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

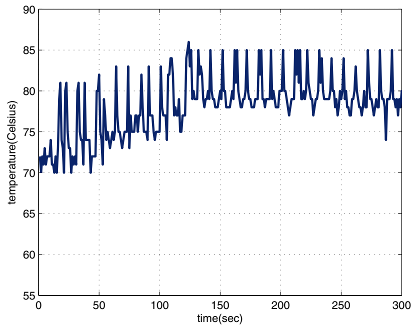


Figure 1: Real temperature of one core on running `bzip2` benchmark

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

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \cdots + \theta_n f_n(u), \tag{1}$$

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:

$$X = \begin{bmatrix} f_1(u_1) & \cdots & f_n(u_1) \\ \vdots & \vdots & \vdots \\ f_1(u_m) & \cdots & f_n(u_m) \end{bmatrix} \tag{2}$$

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

$$\theta = [\theta_1, \theta_2, ..., \theta_n]^T \tag{3}$$

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

$$Y = [Y_1, Y_2, ..., Y_n]^T \tag{4}$$

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

$$\theta = (X^TX)^{-1}X^TY, \tag{5}$$

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 $P_k = (X^TX)^{-1}$ for the k^{th} in Equation (5). Likewise, the recursive least square method at $(k+1)^{th}$ can be developed as

$$P_{k+1} = P_k - \frac{P_k x_{k+1} x_{k+1}^T P_k}{1 + y_{k+1}^T P_k y_{k+1}}, \tag{6}$$

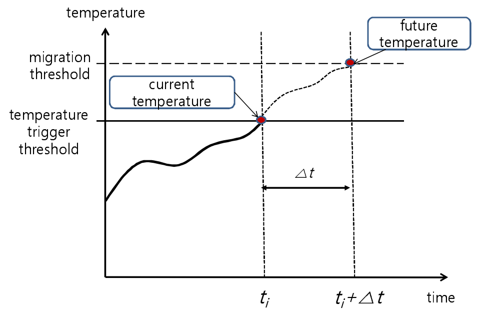


Figure 2: The calculation of Δt(migration time) using ABTM

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

$$\theta_{k+1} = \theta_k + P_{k+1}x_{k+1}(y_{k+1} - x_{k+1}^T\theta_k) \tag{7}$$

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 [4]. With Equation (1), ABTM can predict future temperature for an application as shown in Figure 2. How the ABTM applied in PDTM is explained in 3.3

3.2 The Core-based Thermal Model

The heat transfer equations model the steady state temperature of systems with heat sources [7]. 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 [7]. 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 [15]. Thus

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{8}$$

Solving Equation (8) with $T(0) = T_{init}$ and $T(\infty)$=T_{ss}, we get

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{9}$$

where b is a processor-specific constant. The value of b is determined using Equation (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 (9), we get the steady state tempera-

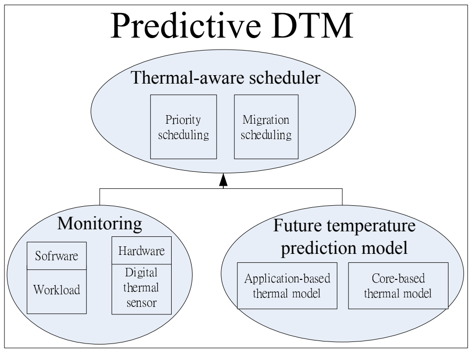


Figure 3: System Overview

ture T_{ss} of the application.

$$T_{ss} = \frac{T(t) - T_{init} \times e^{-bt}}{(1 - e^{-bt})} \tag{10}$$

Therefore, with Equation (9) and (10), we get the future temperature after time t and the steady state temperature, T_{ss}, of each core.

3.3 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:

$$T_{predict} = w_s T_{app} + w_l T_{core} \tag{11}$$

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.

4. 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 [2]. However, when the workload is noticeably unbalanced,

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

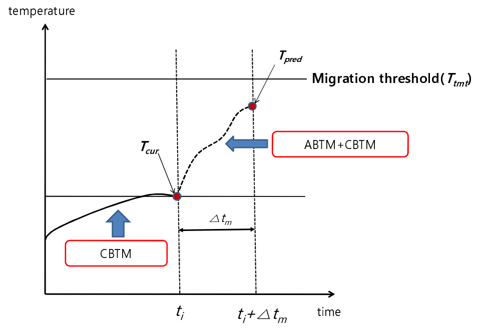


Figure 4: PDTM utilizes ABTM and CBTM simultaneously to predict both short-term and long-term future temperature for multicore

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
```

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 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 Algorithm (1), 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 minimum value among future temperature

Table 1: 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

($T[]$) is selected as new core for migration. As shown in Figure 4, 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.

5. 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 [1]. In our 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. All experiments are tested under ambient temperature control and fixed fan speed.

5.1 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 (°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 (12).

$$T_{core} = T_{junction} - DTS_{value} \quad (12)$$

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

5.2 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 1. Running the single benchmark, the presented PTDM can decrease 8% temperature in average (Figure 5), and reduces up to 5°C in peak temperature with only under 1% performance overhead compared to Linux standard scheduler without DTM as shown in Figure 7. Running two benchmarks simultaneously, the proposed 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 (Figure 6). It means PDTM can be more effective to control temperature than Linux standard scheduler when temperature and workload is higher.

In order to make comparison, we also rebuilt HybDTM [8] (the software scheme-changing priority) and HRTM [10]

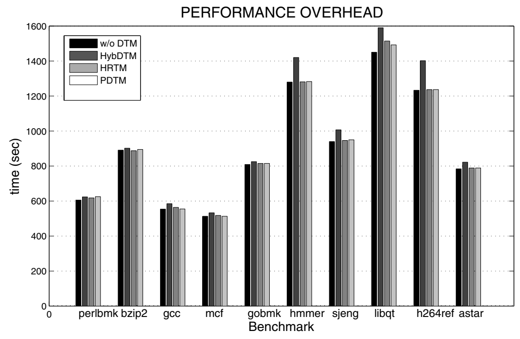


Figure 7: Performance Overhead:PDTM incurs only under 1% performance overhead in average while running single benchmark

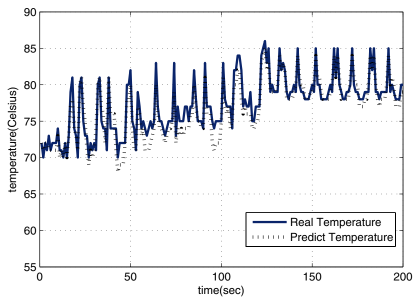
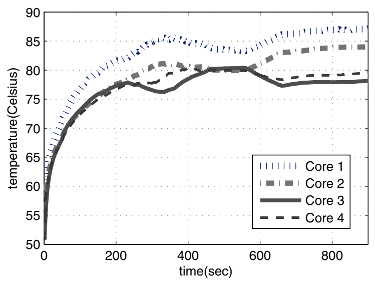


Figure 8: The prediction model can estimate future temperature with only less than 1.6% error on running `bzip2` benchmark

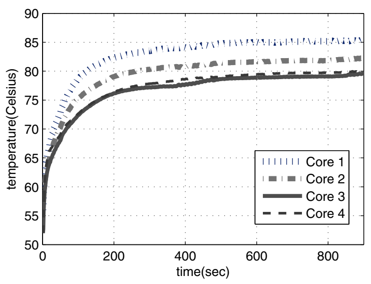
on our Quad-Core system. HybDTM uses priority-based scheme and HRTM uses migration-based scheme. HybDTM scheme relies on hardware performance counter, while HRTM relies on current temperature information. The experimental results show our PDTM outperforms HRTM in reducing average temperature by about 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 8; 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.

6. CONCLUSION

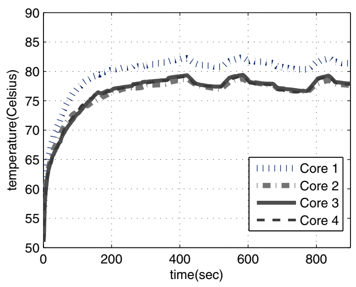
In this paper, we propose the Predictive Dynamic Thermal Management with an advanced future temperature prediction model for multicore systems, and implement PDTM on Intel Quad-Core with a specific device driver to access the Digital Thermal Sensor. We demonstrate that our scheme is able to reduce the overall temperature and provide thermal fairness among four cores. The proposed temperature prediction model can provide more accurate prediction and



(a) Without DTM

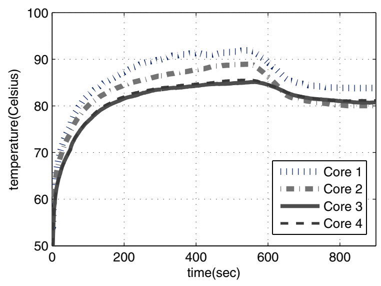


(b) HRTM

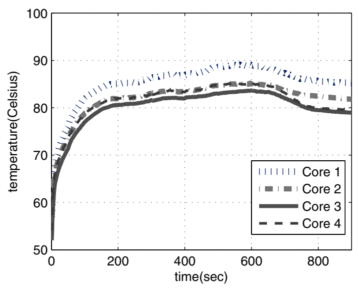


(c) PDTM

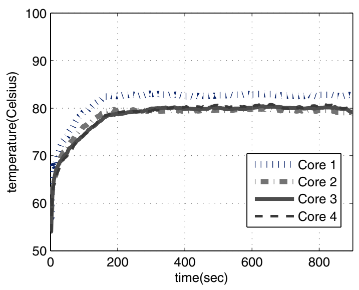
Figure 5: Comparisons among without DTM, HRTM, and PDTM using `libquantum` benchmarks



(a) Without DTM



(b) HRTM



(c) PDTM

Figure 6: Comparisons among without DTM, HRTM, and PDTM using `bzip2` and `libquantum` benchmarks

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. For the future work, we will test our schemes in different platforms with various benchmark such as JBB2005, and WEB2005 to verify their scalability in more general environment.

7. REFERENCES

[1] "Intel 64 and IA-32 Architectures Software Developer's Manual," http://support.intel.com/design/processor/manuals/.

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

[3] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," in *HPCA*, 2001.

[4] X. Chen, "Recursive Least-Squares Method with Membership Functions," in *International Conference on Machine learning and Cybernetics*, 2004.

[5] S. Gunther, F.Binns, D.Carmean, and J.Hall, "Managing the Impact of increasing Microprocessor Power Consumption," *Intel Technology Journal*, 2001.

[6] S. Heo, K. Barr, and K. Asanovic, "Reducing Power Density through Activity Migration," in *ISLPED*, 2003.

[7] F. Kreith and M. S. Bohn, *Principles of Heat Transfer*. CENGAGE-Engineering, 2000.

[8] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, "HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management," in *DAC*, 2006.

[9] 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*, vol. 4, no. 2, 2007.

[10] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System," in *ASPLOS*, 2004.

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

[12] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture," *ISCA*, 2003.

[13] K. Skadron, "Hybrid Architectural Dynamic Thermal Management," in *DATE*, 2004.

[14] J. Srinivasan and S. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications," in *ICS*, 2003.

[15] S. Wang and R. Bettati, "Reactive Speed Control in Temperature-Constrained Real-Time Systems," in *ECRTS*, 2006.