Temperature-Aware Scheduler Based on Thermal Behavior Grouping in Multicore Systems

Inchoon Yeo and Eun Jung Kim
Department of Computer Science and Engineering
Texas A&M University
College Station, TX77840, USA
{ryanyeo, ejkim}@cs.tamu.edu

***Abstract*—Dynamic Thermal Management techniques have been widely accepted as a thermal solution for their low cost and simplicity. The techniques have been used to manage the heat dissipation and operating temperature to avoid thermal emergencies, but are not aware of application behavior in Chip Multiprocessors (CMPs). In this paper, we propose a temperature-aware scheduler based on applications' thermal behavior groups classified by a *K*-means clustering method in multicore systems. The application's thermal behavior group has similar thermal pattern as well as thermal parameters. With these thermal behavior groups, we provide thermal balances among cores with negligible performance overhead. We implement and evaluate our schemes in the 4-core (Intel Quad Core Q6600) and 8-core (two Quad Core Intel XEON E5310 processors) systems running several benchmarks. The experimental results show that the temperature-aware scheduler based on thermal behavior grouping reduces the peak temperature by up to 8°C and 5°C in our 4-core system and 8-core system with only 12% and 7.52% performance overhead, respectively, compared to Linux standard scheduler.**

I. Introduction

As power density increases with high technology, the chip temperature has threatened the system performance, reliability, and even system safety. For high-performance Chip Multiprocessors (CMPs), thermal control has become an important issue due to their high heat dissipation. Thermal packaging and fans can be the primary solution, but suffer from high cost and complexity. Therefore, Dynamic Thermal Management (DTM) has been getting more popular for its low cost and flexibility.

In [2], 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. Although many hardware-based DTM techniques, such as Dynamic Frequency and Voltage Scaling (DVFS) and clock gating, have been proposed and applied in the modern processors in nowadays, the demand of more efficient DTM techniques are prevailing in the multiprocessor server systems [2], [4], [5], [12], [13], [16], [9]. In DTM schemes for CMPs, a thread migration have been proposed to achieve thermal balance among cores without throttling the computation performance in the CMPs systems. However, the studies mentioned above, with the exception of [13], [16], are reactive to the increased chip temperature, while [13], [16] are proactive based on the predicted future temperature. The proactive DTMs are more effectively in temperature control against thermal emergency, since they trigger the control schemes before the core temperature reaching the desired threshold. Since applications have used different functional units that can affect operating temperature, the temperature difference among applications can be up to 9°C [5], [11]. In fact, the temperature difference between on-chip components can be as much as $10 \sim 15$ °C [11]. Also, all applications do not result in the same heat dissipation pattern. In other words, there are significant variations in the thermal characteristics among different applications [11], [13] and different cores in the same chip.

In this paper, we propose a simple and accurate prediction model to profile application's thermal behavior and classify them into several groups offline, and then measure the time duration before reaching the desired temperature threshold for each core. The proposed temperature-aware scheduler is scalable to any current multicore model and architecture with on-chip thermal sensors that can be accessed at the software level.

The main contributions of this paper are summarized as follows:

- We classify applications' thermal behavior groups using *K*-means clustering method with a steady state temperature.
- We propose an efficient temperature-aware scheduler in multicore systems and implement it for Intel Quad-Core Q6600 and two Quad Core Intel XEON E5310 processors systems. We demonstrate that our scheme is able to successfully reduce the overall temperature and provide thermal fairness among cores.
- Most importantly, there is no additional hardware unit required for our temperature-aware scheduler. Our scheme is applicable to any multicore environment in real-world CMP products seamlessly.

The remainder of the paper is organized as follows : Section II notes the previous works and Section III provides the explanation of thermal model in detail. In Section IV, we explain thermal behavior group and how to classify applications using *K*-means clustering method as a thermal behavior group. In Section V, a temperature-aware scheduler based on thermal behavior group is described. The analysis results are discussed and conclusions are provided in Section VI and VII, respectively.

II. Related Work

Several advanced DTM studies [3], [10], [15], [9], [7] have been proposed to provide thermal fairness and reduce the peak temperature through temperature-aware thread migration schemes. However, as presented in [11], an accurate and practical dynamic model of temperature is needed to accurately characterize current and future thermal stress, application-dependent thermal behavior, as well as to evaluate architectural techniques for managing thermal effects.

Donald *et al.* [3] introduce various thermal management polices such as DVFS and thread migration based on current temperature, but their scheme fails to delve into the thermal model for CMPs to determine which core would be best after migration. Powell *et al.* describe techniques for thread assignment and migration to balance temperature [10]. Although their works use performance counter-based information to determine migration, this information can not be a direct representation of thermal behavior.

Yang *et al.* propose the execution ordering between hot threads and cool threads for temperature control [15]. However, it is impossible which treads would be hot or cool at runtime before their executions. Moreover, since applications do not have consistent thermal behavior, the change of execution ordering for threads does not guarantee thermal safety at runtime.

In [9], the thread with less task size tends to be migrated easier than other threads to reduce the performance overhead caused by migration, but the task size does not necessarily means the memory usage, especially the cache usage in the core. Moreover, due to our experiment results, the difference of migration cost between tasks with distinct memory usages is only within 10 ms, and migration cost mainly comes from the task suspension and resumption. Furthermore, these prior 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. Besides, the migration action is triggered by the current temperature (when the temperature is higher than maximum allowed temperature) in these studies; however, instead of considering the current temperature, we propose to estimate each core's future temperature and measure the time duration before reaching the desired temperature threshold for each core to determine the migration target core.

In [7], 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 noticeable performance overhead. This is because the real temperature cannot be estimated solely by hardware performance counter, and both of priority scheduling and clock gating will introduce high performance overhead.

III. Thermal Model

The heat transfer equations are introduced to model the steady state temperature[1] of systems with heat sources in [6]. With those heat transfer equations, Wang *et al.* present that the rate of temperature change is proportional to the difference between the current temperature and the steady state in [14]. 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,

$$\frac{dT}{dt} = b \times (T_{ss} - T). \tag{1}$$

where b is a thermal parameter dependent on hardware specifications. Solving Equation (1) with $T(0)$ = T_{init} and $T(\infty)$ = T_{ss}, we can obtain

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{2}$$

Using Equation (2) and our measurements, we can obtain T_{ss} and b using the following steps:

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

Therefore, once the thermal parameter b and T_{ss} are obtained, they could be used in the proposed prediction model to estimate each core's time duration before reaching the predefined temperature threshold. Moreover, we can notice that each core's thermal parameter b and T_{ss} are different according to which application works even though the cores are within the same package.

IV. Thermal Behavior Group

In this section, we propose to classify the thermal behavior group by T_{ss}. Also, we introduce how to predict future temperature and time duration before reaching the predefined threshold using thermal parameter b and thermal behavior groups.

A. Thermal Behavior Group

As shown in Fig. 1, T_{ss} of each benchmark suite is different from each other, although all of their CPU utilizations are almost 100%. In order to manage temperature at runtime, accurate applications' thermal behavior should be necessary. We observe that among T_{ss} and thermal parameter b in Equation (2), T_{ss} value is more sensitive than thermal parameter

[1]The steady state temperature of an application is defined as the temperature, which can be reached in real system if the application is executed infinitely.

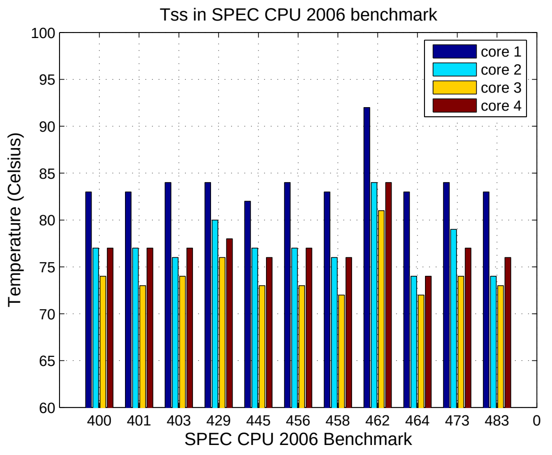


Fig. 1. T_{ss} according to SPEC CPU 2006 benchmark suite

b to different thermal behaviors of applications. As shown in Fig. 2, applications' thermal patterns are similar if their T_{ss} are analogous. In this research, we classify SPEC CPU 2006 benchmark applications with T_{ss} value as several thermal behavior groups using a K-means clustering method.

The K-means clustering method is an algorithm to cluster n objects based on attributes into k partitions, $k < n$. As our preliminary experiments for twelve SPEC CPU 2006 benchmarks, $k = 5$ is the optimal value to classify applications as thermal behavior group as shown in Table I.

For example, `400.perlbench`, `401.bzip2`, `403.gcc`, and `456.hmmer` applications can be classified as the same group (Group A) that has a similar thermal pattern and T_{ss}, as shown in Fig. 2.

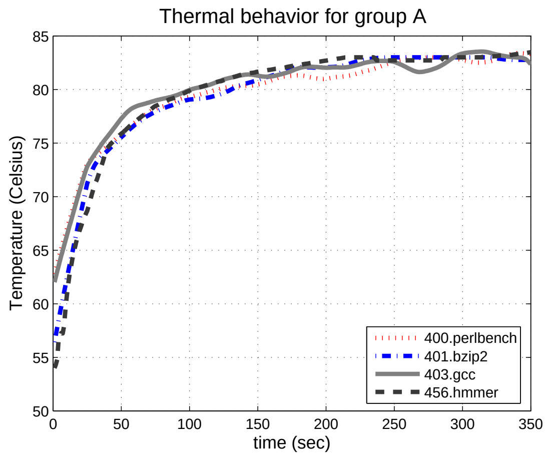


Fig. 2. Thermal Behavior for Group A

B. The region of the thermal behavior group

While we use a steady state temperature value, T_{ss}, for clustering, we need to find another metric for the classification of a new application at runtime. Since before reaching steady state, the T_{ss} value is not available, we use only the measured

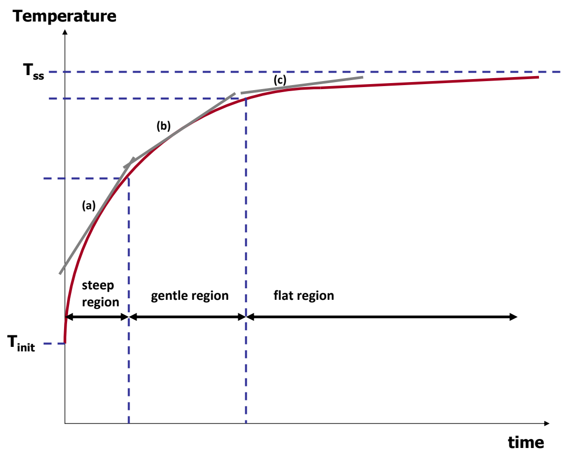


Fig. 3. Slopes for thermal pattern at runtime

temperatures while applications running. In our observation, the thermal pattern can be divided by three regions as shown in Fig. 3. Each region has a different thermal slope that can affect the temperature increasing rate at a given time.

As shown in above Fig. 3, the slope (a), (b), and (c) are different according to the time t. To calculate the slope for operating temperature, we can use a simple equations as follows:

$$S_i = \frac{T(i + \Delta t) - T(i)}{\Delta t}, \tag{3}$$

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

V. Temperature-Aware Scheduler

Since Linux standard scheduler is not aware of operating temperature for cores in multicore environments, we propose a temperature-aware scheduler that exploits this variability in the context of multicore systems. Although the proactive schemes have been proposed in [16], the scheme in [16] fails to consider the difference of temperature increasing pattern in different cores, because T_{ss} and thermal parameter b are impractically assumed to be the same in each core within a single chip. Most importantly, in [16], the authors propose to migrate tasks from a potentially overheated core to the future coolest core based on the temperature prediction results. However, we believe that the target core of task migration should be determined by the core which needs the longest time period to reach the predefined temperature threshold, because temperature of the coolest can be increased faster than others due to the thermal correlation effects and applications' thermal behaviors.

TABLE I
THE RESULT OF THERMAL BEHAVIOR GROUP USING K-MEANS CLUSTERING ON 4-CORE SYSTEM

SPEC CPU Applications	Core 1 T_{ss}	Core 2 T_{ss}	Core 3 T_{ss}	Core 4 T_{ss}	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	**GROUP**
400.$perlbench$	83°C	77°C	74°C	77°C	1	3	2	5	3	**A**
401.$bzip2$	83°C	77°C	73°C	77°C	1	1	2	5	3	**A**
403.gcc	84°C	76°C	74°C	77°C	1	1	2	5	3	**A**
429.mcf	84°C	80°C	76°C	78°C	1	1	1	3	4	**D**
445.$gobmk$	82°C	77°C	73°C	76°C	1	1	2	1	6	**C**
456.$hmmer$	84°C	77°C	73°C	77°C	1	1	2	5	3	**A**
458.$sjeng$	83°C	76°C	72°C	76°C	1	3	2	1	6	**C**
462.$libquantum$	92°C	84°C	81°C	84°C	2	3	2	4	1	**E**
464.$h264ref$	83°C	74°C	72°C	74°C	1	3	4	2	5	**B**
473.$astar$	84°C	79°C	74°C	77°C	1	1	1	3	4	**D**
483.$xalanchbmk$	83°C	74°C	73°C	76°C	1	4	2	2	2	**B**

As explained in Section IV, each core's and application's thermal behavior is different by T_{ss} and thermal parameter b. Therefore, migrating task from a potential overheated core to the coolest core is unnecessary and improper.

In this paper, we first profile T_{ss} of applications offline, and then classify them as thermal behavior groups using K-means clustering method. Whenever an application starts to run, the slope of the application is calculated after triggering a start threshold. According to an execution time, t, it is possible that this application can be classified into which thermal behavior group. As a result, we can acquire T_{ss} and thermal parameter b for the application from this thermal behavior group at runtime. Based on these T_{ss} and thermal parameter b, a temperature-aware scheduler starts to predict each core's future temperature and the time period before reaching the desired temperature threshold. If the estimated time period is less than 2 seconds, it means the cores are going to be overheated in the near future, and the task migration should be triggered. The migration target core is determined by which other core needs the longest time to reach the desired temperature threshold. Then, all the tasks on the potentially overheated core can be migrated to the target core to balance the heat within multicore environments.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

In Intel's Core Architecture [1], Dynamic Thermal Sensor (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). Using these registers, we developed a thermal device driver for getting operating temperature on multicore systems. Our scheme consists of three components such as Application's Thermal Behavior group, The Longest Core Find Module, and Proactive Temperature-Aware Scheduler as shown in Fig. 4. We conduct our experiments in two different multicore systems as shown in Table II.

In order to demonstrate the applicability of our temperature-aware scheduler for various applications, we utilize several thermal behavior groups classified by K-means clustering method, as shown in Table I. In this paper, we used twelve applications in SPEC CPU 2006 benchmark suite for profiling.

Algorithm 1 Algorithm of Temperature-Aware Scheduler

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

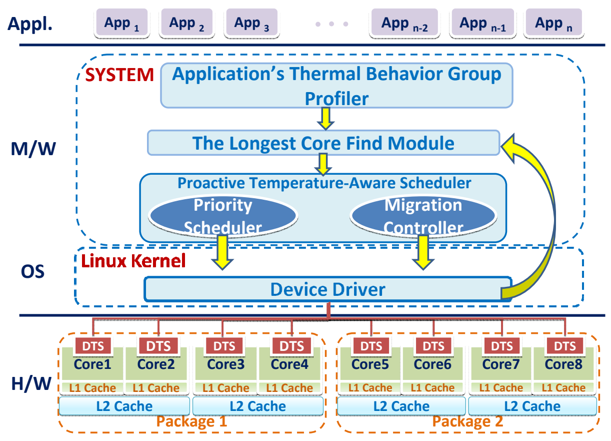


Fig. 4. System Overview (8-core system)

TABLE II
EXPERIMENTAL SYSTEMS DESCRIPTIONS

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

In our experiments, We choose *bzip2* and *libquantum* in SPEC CPU2006 benchmark, *vacation* from STAMP benchmark [8]. We select *bzip2* and *libquantum* because they are CPU-intensive. Also, *vacation* is a client/server travel reservation system benchmark that is appropriate to present the demand of thermal control in the server systems.

To compare with other schemes, we also rebuild the Predictive Dynamic Thermal Management (PDTM) [16] and Thermal Balancing Policy (TBP) [9] in our systems. All experiments in this paper is under ambient temperature control, and the speed of cooling fan is also fixed.

A. 4-core system

To verify a temperature-aware scheduler for 4-core system, two applications run simultaneously. As shown in Fig. 5, compared to Linux Standard Scheduler, a temperature-aware scheduler reduces peak temperature up to 8°C. However, PDTM reduce the peak temperature by 2°C , while TBP is increased by 2°C. For the performance overhead evaluation, the temperature-aware scheduler presents less than 12% performance overhead, and PDTM has 8% compared to the Linux Standard Scheduler, while TBP incurs 35%. Since the temperature-aware scheduler finds the longest core instead of the coolest core under thermal threshold, our scheme reduces the number of migration while providing better thermal-balancing for cores compared to other DTMs. Therefore, the temperature-aware scheduler based on thermal behavior grouping provides the better effectiveness in temperature control for multicore systems.

B. 8-core system

In 8-core system, the temperature-aware scheduler outperforms PDTM and TBP in both temperature control effectiveness and efficiency as shown in Fig. 6. The temperature-aware scheduler reduces peak temperature by 5°C with 7.52% performance overhead compared to Linux Standard Scheduler, while PDTM and TBP reduce peak temperature by 4°C and 13.2% performance overhead and 2°C and 35% performance overhead, respectively. Although TBP also decreases the peak temperature and presents smoother thermal pattern compared to Linux Standard Scheduler, TBP causes impractically huge performance overhead. Moreover, the exchanged threads cannot effectively reduce core 1's temperature. Since PDTM is not aware of the different thermal effects contributed by running applications, PDTM cannot accurately predict future temperature and react in time. Therefore, as shown in Fig. 6, PDTM fails to control the temperature under the desired level.

VII. CONCLUSION

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

REFERENCES

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

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

[3] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," in *ISCA*, 2006.

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

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

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

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

[8] C. C. Minh, "STAMP - Stanford Transactional Applications for Multi-Processing," available from http://stamp.stanford.edu/.

[9] 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*, 2008.

[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-XI*, 2004.

[11] 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, 2004.

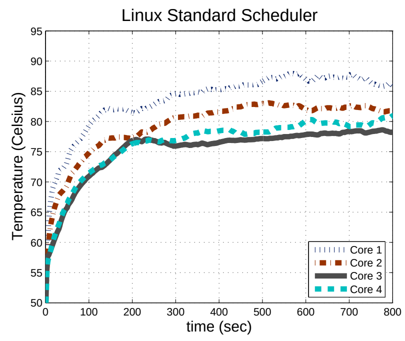
[12] K. Skadron, "Hybrid Architectural Dynamic Thermal Management," in *DATE*, 2004.

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

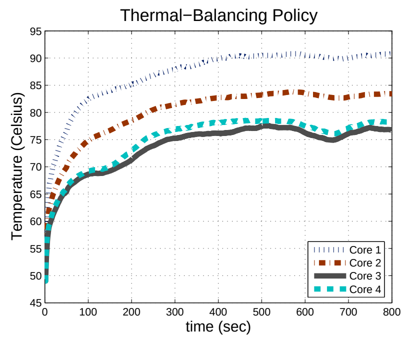
[14] S. Wang and R. Bettati, "Reactive Speed Control in Temperature-Constrained Real-Time Systems," in *ECRTS*, 2006.

[15] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic Thermal Management through Task Scheduling," in *ISPASS*, 2008.

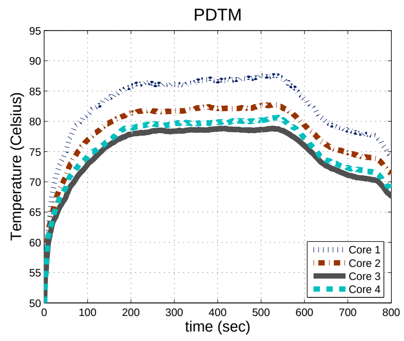
[16] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive Dynamic Thermal Management for Multicore Systems," in *DAC*, 2008.



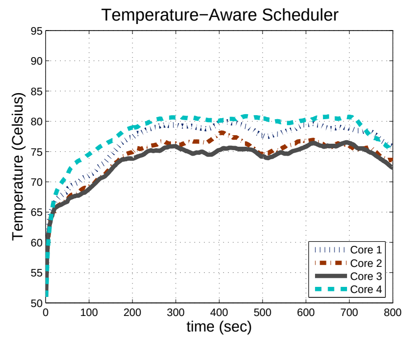
(a) Standard Scheduler



(b) Thermal-Balancing Policy

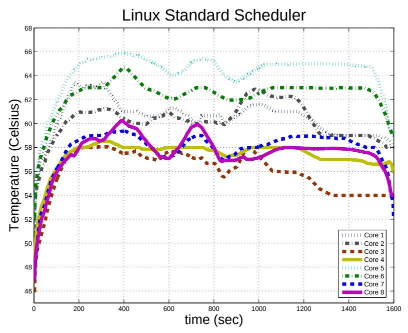


(c) Predictive DTM

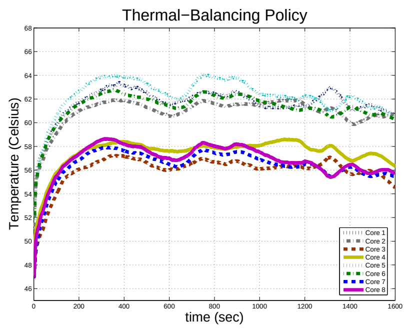


(d) Temperature-Aware Scheduler

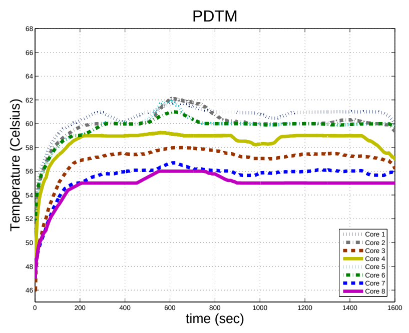
Fig. 5. The evaluation of DTMs in 4-core system (*bzip2* + *libquantum*)



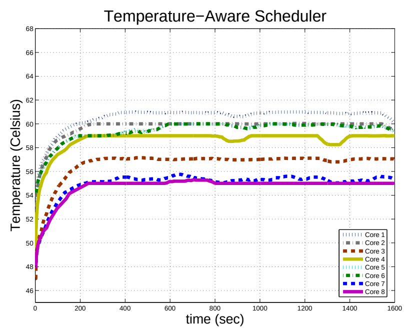
(a) Standard Scheduler



(b) Thermal-Balancing Policy



(c) Predictive DTM



(d) Temperature-Aware Scheduler

Fig. 6. The evaluation of DTMs in 8-core system (*bzip2* + *libquantum* + *sjeng* + *h264ref* + *vacation*)