Physical-Aware Predictive Dynamic Thermal Management of Multi-core Processors¹

Bagher Salami ^{a,d}, Hamid Noori ^{a,d,*}, Farhad Mehdipour ^b, Mohammadreza Baharani ^{c,d}

^a Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

^b E-JUST Center, Graduate School of Information Science and Electrical Eng., Kyushu University, Fukuoka, Japan

^c School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran

^d School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

Email: bagher.salami@stu-mail.um.ac.ir, hnoori@um.ac.ir, farhad@ejust.kyushu-u.ac.jp, m.baharani@ut.ac.ir

* Corresponding Author.

This research was in part supported by a grant from IPM. (No. CS1394-4-12)

¹ This paper is the extension of our ASP-DAC 2014 paper, titled “Physical-Aware Task Migration Algorithm for Dynamic Thermal Management of SMT Multi-core Processors”. The details and the differences between the two versions have been explained in the cover letter.

Abstract— The advances in silicon process technology have made it possible to have processors with larger number of cores. The increment of cores number has been hindered by increasing power consumption and heat dissipation due to high power expenditure in a small area die size. The high temperature can cause degradation in performance, reliability, transistor ageing, transition speed and increase in leakage current. In this paper, we present a method which considers different thermal behavior of cores and uses both physical sensors and performance counters simultaneously to improve thermal management of both SMT multi-core processors with a physical sensor per core and Non-SMT multi-core processors with only one physical sensor for the processor. The experimental results indicate that our technique can significantly decrease the average and peak temperature in most cases compared to Linux standard scheduler, and two well-known thermal management techniques: PDTM, and TAS.

Keywords: dynamic thermal management, multi-core processors, task migration, DVFS, physical features

I. Introduction

The advances in silicon process technology have made it possible to have processors with a larger number of cores. The increment of the cores count has been hindered by rising power consumption, and heat dissipation due to high power expenditure in a small area die size. High temperature causes degradation in performance, reliability, transistor aging, transition speed and increase in leakage current [1]. Therefore, thermal management is becoming a crucial issue for new generations of processors.

In response to these challenges, various Dynamic Thermal Management (DTM) techniques have been proposed to mitigate the thermal concerns of processors [2-10,15-18,20-23]. DTM is a set of techniques that control processor temperature at run-time so that temperature does not go beyond the critical temperature threshold. By keeping the peak temperature lower than the critical temperature

threshold, the chip lifetime is further improved, and the cooling cost, which is a challenge for green computing, decreases [2]. The other goal of efficient DTM techniques is to balance and minimize spatial thermal variation of the processor and avoid hotspots, which is defined as the maximum difference in temperature between the hottest and the coldest core. The benefits of minimizing spatial thermal variation of a processor are to reduce the peak temperature and temporal thermal variation and also to enhance and balance the aging of different cores in a multi-core processor [1].

DTM techniques are available at both hardware (HW) and software (SW) levels. Stop-and-go is one of the primitive hardware-based DTM techniques, where a processor core enters into a sleep state at a temperature threshold and resumes execution once the temperature returns to the normal operating range [1]. Dynamic Voltage and Frequency Scaling (DVFS) is another hardware-based DTM technique [3-5] that dynamically adjusts the processor voltage and frequency to reduce power and temperature. Another hardware-based DTM technique is power-gating to reduce leakage power via inserting sleep transistors between actual ground and virtual ground [6]. Although HW-based approaches decrease temperature and power consumption significantly, they degrade overall system performance due to longer execution time [1]. Software-based is another category of DTM techniques that consist of (a) task scheduling, (b) task migration, (c) idle cycle injection. The aim of task scheduling technique is to distribute tasks among different cores to prevent hot spots [7-8]. Task migration is another technique that moves tasks from a hot core to an appropriate core to control and manage the overall temperature [9-13]. The major challenge of task migration is to find an appropriate target core to decrease the migration frequency while decreasing the temperature. Idle cycle injection technique periodically idles the processor through injecting idle cycles at the scheduler level [14]. By idling the processor for periods of time in-between regular program execution, it briefly enters low-power states and cools down. The significant benefit of software based DTM techniques is that they can reduce the temperature without dramatically performance degradation at any extra hardware cost.

To apply DTM techniques, one needs to measure the temperature of the processor and manage processors temperature efficiently. There are different approaches for cores and applications thermal measurement. Two well-known methods, named as CMOS thermal sensors and performance-counter-based (software-based) sensors [1] are used to measure processor's thermal patterns. Alongside, to characterize application's thermal behavior there are two ways including application thermal profiling and using performance counters [1]. Since application thermal profiling is an offline method, it cannot reflect the real thermal pattern of the processor and applications. Performance counters are mostly used for online application temperature characterization though they are inaccurate [15]. because the reason is that, some models and equations are required to convert the values of performance counters to the temperature. Moreover, reading different performance counters imposes significant overhead on application execution at run-time [1]. Therefore, recently proposed methods model overall core and application temperature with the aid of physical sensor and steady state temperature [16-17].

In order to have more efficient DTM techniques, recent works predict future temperature of cores to reduce overheat temperature with negligible performance overhead [16-17]. Their proactive task migration approaches predict the future temperature and manage the workload to reduce and balance the temperature before reaching the temperature threshold.

In this paper, we propose a DTM algorithm and evaluate it on commercial systems. The unique feature of the proposed algorithm is using what we call the *core unique thermal behavior* (CUTB) explained as follows. Different cores of a processor do not have similar thermal behavior due to process variation [18], the temperature effect of neighbor components [3], and other physical issues [1]. The temperature difference between cores of a processor running the same application can be as much as 10~15°C [16]. In this paper, we name this phenomenon as *CUTB*. It means that the cores of a multi-core processor show different thermal behavior for the same workload.

Motivated by these facts, we propose a method which considers different thermal behavior of cores (*CUTB*) and uses both physical sensors and performance counters simultaneously to improve thermal

management of both SMT multi-core processors with a physical sensor per core and Non-SMT multi-core processors with only one physical sensor for the processor. Simultaneous multi-threading (SMT) multi-core processors have been prevalent recently. SMT multi-core processors can exploit more thread-level parallelism by less hardware compared to non-SMT multi-core processors. Each core of an SMT multi-core processor has only one physical temperature sensor, and it is hardly possible to know the contribution of each thread on the total temperature. Therefore, temperature measurement or estimation of multiple threads running on an SMT core based on only one physical temperature sensor of a core is impossible and we have to utilize methods to measure and estimate the temperature of each thread (application). The same problem exists for Non-SMT multicore processors that have only one physical sensors.

We utilize physical sensors to estimate and predict the future temperature of the cores and performance counters to classify the applications thermal behavior at runtime. Another feature of proposed technique is that it has an adaptive migration threshold which will be explained later in this paper. The experimental results on Intel's Core i7 (SMT enabled with one physical sensor per core) and AMD's octa-core Bulldozer (Non-SMT with only one physical sensor) running up to eight benchmarks indicate that our proposed method, called PATM (Physical-Aware Task Migration), outperforms Standard Linux scheduler in reducing average and peak temperatures. We also rebuilt two already proposed task migration algorithms, PDTM [16] and Task-Aware Scheduler (TAS) [17] for more comparison. Our proposed method outperforms both PDTM and TAS in reducing average temperature and peak temperature while performance overhead is insignificant. To summarize, the main contributions of this paper are as follows:

- We propose a thermal-aware scheduling method for both SMT and Non-SMT multi-core processors based on the different thermal behavior of cores due to their *core unique thermal behavior*.
- Our experimental results on commercial processors indicate that our proposed approach, under

full workloads, outperforms the Linux standard scheduler and two existing DTM techniques i.e. PDTM and TAS.

- There is no additional hardware unit required for our prediction model and thermal-aware algorithm. It means that our approach is scalable for all the multicore systems and can be applied to off-the-shelf SMT multi-core products.

The remainder of the paper is organized as follows: The related work is discussed in Section II. Section III describes our proposed algorithm in details. In Section IV, the implementation and analysis results are discussed, and conclusions are drawn in Section V.

II. Background and Related Work

Stop-and-go, DVFS and power-gating are hardware-based approaches to reducing power consumption and temperature. The stop-and-go technique simply pauses the execution of the cores in case of thermal emergency. Three methods of stop and go are discussed in [19]. The first scheme is a clock gating method, which disables portions of circuitry so that flip-flops do not change, preserving the architectural states. The second scheme saves the core state and cuts voltage supply. This has the benefit of consuming no power, as opposed to the first scheme which still dissipates leakage energy, and achieving faster cool down times but at the cost of saving and restoring states. The third scheme known as the Intermediate scheme uses a sleep state that is a lower voltage than nominal and preserves core execution state.

DVFS is another technique that dynamically adjusts the processor voltage and frequency to reduce power and temperature. DVFS techniques can be classified into (a) global and (b) local. Global DVFS allows for scaling of voltages and frequencies of all cores of a processor simultaneously. This may potentially result in unnecessary performance penalties when applied to avoid a thermal emergency involving only one core [1]. Local DVFS allows for scaling of the voltage of individual cores. The

additional flexibility allows for an overheating core to be slowed or stopped if needed by local changes [1]. The optimal DVFS scheduling problem was addressed in [3-5] as separate problems of task-to-core allocation over migration intervals and voltage speed scaling within migration intervals. For saving energy while considering temperature, Bao et al. [20] proposed a DVFS technique with design-time support. They add a temperature analysis process to the design-time analysis to an existing DVFS technique. The work of [21] predicts the effect of dynamic voltage and frequency scaling (DVFS) on performance, power, and energy.

Software-based DTM techniques consist of (a) task scheduling, (b) task migration, (c) idle cycle injection. The task scheduling discusses how to schedule and assign the tasks (processes or threads) to cores for managing the temperature of processors. The work done in [7] investigates and compares some OS (operating system) task scheduling such as cool loop, heat balancing and deferred execution of hot jobs on both SMT and Non-SMT platforms. Task migration is a common technique that is used for enabling a scheduled or executing thread to be selectively run, preempted, or migrated to another core based on its thermal or power profile. In [9] the issues of thread migration such as frequency of migration that can affect performance have been explained. Idle cycle injection technique periodically idles the processor through injecting idle cycles at the scheduler level [14]. By idling the processor for periods of time in-between regular program execution, it briefly enters low-power states and cools down. [14] examines the benefit and problems of short and long idle periods.

To get better results, most of the proposed dynamic power and thermal management algorithms engage a combination of both hardware and software techniques [3-5]. The work of [16-17] uses both DVFS and task migration to achieve maximum energy saving. It adjusts core voltage and frequency according to the assigned workload. It means that the core, which has the heaviest workload, is assigned to the highest voltage and executes tasks with the highest frequency and vice versa. Applying this technique causes the overall processor power consumption decreases and the temperature is kept at an acceptable level regarding the performance constraints.

The key challenge in task migration technique is to minimize the number of migration among cores in order to lessen performance degradation. The simplest strategy for migration is to move a task from a hot core to a cold one; however, the main problems of this strategy are when and where to migrate. Surprisingly, the coolest core is not always the best option for the migration target [17]. On the other hand, task migration should find a core that takes the longest time to reach temperature threshold instead of the coolest one. Therefore, some DTM techniques predict cores temperature to prevent the core from reaching a critical temperature at early stages and also to decrease the migration frequency [3,16-17].

From another perspective, DTM techniques can be categorized into reactive and proactive methods. Reactive thermal management methods, which act (e.g. task scheduling and migration) after the temperature reaches the threshold, maintain the temperature below a critical level at the cost of performance degradation [2]. Reactive thermal management techniques have other disadvantages [2]: i) they take action after the temperature has violated a threshold and may not be able to prevent damages in certain cases where temperature rises above the safe operating range of the internal components, ii) it is very difficult to determine the optimal threshold. In contrast, proactive task migration approaches, which try to predict the future temperature, manage the workload of processor to reduce and balance the temperature before the temperature reaches the threshold [22].

One of the first attempts of using prediction in DTM is [16] which predicts core temperature based on both application thermal and core thermal models. The work of [3] also presents a neighbor-aware prediction algorithm. In this work, temperatures of neighbors are also considered to predict future temperature more accurately in order to maximize system throughput under peak temperature system constraints.

Yeo et al. [17] categorizes applications according to their thermal behavioral for improving the accuracy of temperature prediction. The authors use K-mean algorithm as a classifier and consider T_{ss} value as a classification factor where T_{ss} is the steady state temperature of an application. The steady

state temperature of an application is defined as the temperature that the processor would reach if the application is executed infinitely [16]. Steady state temperature for each program on a specific processor at the specific ambient temperature and CPU frequency is a fixed value. The authors claim that T_{ss} is a proper factor for grouping and explaining application thermal pattern behavior. We have extended their scheme by offering , i) new migration and task scheduling mechanism by considering *CUTB*, ii) adaptive migration threshold, and iii) improved temperature predictor.

III. Problem Statement and Preliminaries

In this section, first the problem is described and then the term *CUTB* of the cores is explained in details. Finally, the proposed processor temperature predictor is introduced.

A. Problem description

The system considered in this paper consists of a multi-core processor with N cores, denoted as $\{\text{core}_1, \text{core}_2, \dots, \text{core}_N\}$ where for SMT multi-core processors, $N/2$ cores are physical and other $N/2$ are logical, while for Non SMT multi-core all cores are physical. It is assumed that there are up to M tasks for execution on the N Cores where $M \leq N$. The problem discussed in this paper is how to dynamically schedule these tasks among cores and scale the frequencies of the cores such that the average and peak temperature of the processor is minimized under minimum performance loss and also avoid the processor temperature to violate T_{max} . A heuristic method is proposed to solve the above problem based on task migration and DVFS. It is assumed that the processor features global DVFS and performance counters.

First, we describe what we mean by *CUTB* and then we introduce a new temperature prediction method, which predicts the future temperature of a core by considering both *CUTB* and workload of the processor. In this algorithm, task migration is activated at *critical situations* i.e. when there is at least one core that reaches to T_{thr} in less than t_{res} , where T_{thr} is temperature threshold at which tasks are

migrated to better cores in order to reduce the temperature, and t_{res} is the response time for the algorithm to decrease the core temperature.

B. The core's unique thermal behavior (CUTB)

As mentioned earlier, the temperature of each core of a processor is different from other cores under the same conditions including the running workload, fan speed, and the ambient temperature. Table I, II, and III summarizes our experimental results for running different applications of SPEC2006 benchmark suite, on different cores of an AMD's octa-core Bulldozer and two Intel quad-core (Core i7-3770 and Core i7-2600) processors, respectively. These tables show the thermal behavior of cores (at a fixed fan speed) while one core executes an application, and other cores are idle. The reported temperature is the maximum temperature among all cores for Intel Core i7 processors where each physical core has a physical sensor. In case of AMD's octa-core Bulldozer there is only one physical sensor for the processor. For example, in Table I, 43.5°C is the peak temperature of AMD's octa-core Bulldozer, when core 3 executes the bzip2, and other cores are idle.

According to Table I, despite all cores of AMD's octa-core Bulldozer have the same experimental setup, core 0 and core 5 are always the coldest and core 2 is the hottest core among all cores. We tried the same experiments with two Intel quad-core Core i7-2600 and Core i7-3770 and observed similar phenomenon (i.e. the existence of differential cores thermal behavior). As can be seen in Table II, core 2 and core 3 are the hottest and coolest cores for Core i7-3770. For Core i7-2600, Table III shows core 3 and core 1 are always the hottest and coolest cores, respectively. This phenomenon, which we refer it as *core unique thermal behavior (CUTB)* of multi-core processors, motivated us for our proposed DTM algorithm. In the rest of paper, we fully explain how we take advantage of *CUTB* to enhance the thermal management.

TABLE I
The temperature differential between cores of AMD's 8-core Bulldozer.

Benchmark	Executed on core 0	Executed on core 1	Executed on core 2	Executed on core 3	Executed on core 4	Executed on core 5	Executed on core 6	Executed on core 7
Gcc	41.5°C	42°C	45°C	44.5°C	42°C	42°C	43°C	43.5°C
Hmmer	42°C	43.5°C	44.5°C	44.5°C	42°C	41.5°C	43.5°C	42.5°C
bzip2	41°C	41.5°C	43.5°C	43.5°C	41°C	41°C	41.5°C	42.5°C

TABLE II
The temperature differential between cores of Intel Core i7-3770.

Benchmark	Executed on core 0	Executed on core 1	Executed on core 2	Executed on core 3
Gcc	56°C	56°C	57°C	55°C
Hmmer	60°C	60°C	62°C	58°C
bzip2	59°C	59°C	60°C	58°C

TABLE III
The temperature differential between cores of Intel Core i7-2600.

Benchmark	Executed on core 0	Executed on core 1	Executed on core 2	Executed on core 3
Gcc	59°C	58°C	61°C	64°C
Hmmer	66°C	62°C	63°C	66°C
bzip2	69°C	67°C	69°C	71°C

C. Temperature prediction

Our temperature predictor is a modified version of [17]. Let assume T_{ss} as steady state temperature of an application (the steady state temperature of an application is defined as a temperature that the system reaches if the application is executed infinitely [16]). According to [17] the time derivative of temperature is proportional to difference between the current temperature and steady-state temperature (Eq. 1):

$$\frac{dT}{dt} = c \times (T_{ss} - T), \quad (1)$$

where c is a core-specific constant. The Eq. 1 proposed by [17] assumes that only one core is running an application at any time. Based on this assumption, they find the value of c empirically. However, if other cores are also running different applications, their temperatures are affected by the neighbor cores. Therefore, we add a new parameter w to Eq. 1 and extract Eq. 2:

$$\frac{dT}{dt} = c \times w \times (T_{ss} - T), \quad (2)$$

where w relates to the core activity. w is added to reflect the thermal effects of other cores that are active (running applications) which has not been considered in [17]. The value of c and w are determined empirically and offline. In our case, first, by executing various number of applications of SPEC2006 benchmarks (one to eight applications), simultaneously, on the different cores of the processor the corresponding thermal curves of applications are obtained and then using Eq. 3 the values of c and w are determined.

Solving Eq. 2, with $T(0) = T_{init}$ and $T(\infty) = T_{ss}$, we have:

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-c \times w \times t}. \quad (3)$$

Assigning $T(t) = T_{thr}$, we obtain:

$$t_r = \mu \times \ln \left(\frac{T_{ss} - T_{init}}{T_{thr} - T_{ss}} \right); \mu = \frac{1}{c \times w}, \quad (4)$$

where, t_r is the predicted time when the core reaches T_{thr} . According to our experiments, the values of T_{ss} and c are different for each core. Therefore, the value of t_r should be calculated for each core individually. Based on the value of t_r the proposed algorithm decides when to start task migration.

By rearranging Eq.3, we get the steady state temperature T_{ss} of the application at runtime.

$$T_{ss} = \frac{T(t) - T_{init} \times e^{-c \times w \times t}}{(1 - e^{-c \times w \times t})}. \quad (5)$$

As mentioned before, T_{ss} is different for each application. Yeo *et al.* [17] show that, between T_{ss} and thermal parameter c in Equation (3), T_{ss} is more sensitive to different thermal behaviors of applications than thermal parameter c . We classify SPEC CPU 2006 benchmark applications according to the T_{ss} into several thermal behavior groups using a K-means clustering method. Ten applications from the benchmark are executed and their T_{ss} for different cores are extracted. T_{ss} values are inputs to

the K-mean clustering algorithm. We clustered applications for different values of K and calculated the parameter of *Most Total Sum of Distance* (total distance of each point in cluster from centroid), and found that $K = 5$ is the optimal number of classes. Table IV summarizes the extracted T_{ss} for selected benchmark and results of clustering with different K values.

The K-means classifier is an algorithm for categorizing N data points in an I-dimensional space into K clusters. A vector parameterizes each cluster $\underline{m}_i, 1 \leq i \leq k$.

TABLE IV
Results of application clustering using k-mean on a quad core platform

Benchmarks	core 0 Tss	core 1 Tss	core 2 Tss	core 3 Tss	k=2	k=3	k=4	k=5	K=6
Gcc	67	62	64	65	1	3	4	4	6
Libquantum	72	68	69	72	2	1	2	2	1
Astar	63	58	62	63	1	3	3	5	4
bzip2	69	64	64	67	2	2	4	4	2
Mcf	68	65	67	69	2	2	1	3	3
Gobmk	70	67	67	68	2	2	1	1	5
Sjeng	69	66	67	68	2	2	1	3	3
h264ref	71	66	67	69	2	2	1	1	5
Perlbench	71	65	66	68	2	2	1	1	5
Hmmer	72	66	67	69	2	2	1	1	5
Most total sum of distance among clusters					40	36	28	19	26

Also, a vector denotes the data points $\underline{x}_j = [x_j^1 \ x_j^2 \ \dots \ x_j^I]$ where $1 \leq j \leq N$. We use the following metric to calculate distances between points:

$$d(\underline{x}_j, \underline{x}_h) = \frac{1}{2} \sum_{p=1}^I (x_j^p - x_h^p)^2 \quad (5)$$

Finally, for each category, the c and w coefficients (Eq. 3) are calculated.

IV. The Proposed Physical-Aware Dynamic Thermal Management Algorithm

This section discusses the proposed dynamic thermal management algorithm. In the following subsections, different parts of the algorithm are fully explained. The flowchart of the proposed algorithm is depicted in Fig. 1 which briefly illustrates the intuition behind the algorithm.

The main three parts of the algorithm are *Threshold Management*, *Temperature Management*, and *Performance Management*. One of the unique features of the proposed algorithm is that it has an adaptive temperature threshold (T_{thr}), unlike previous work in which all of them assume that T_{thr} is a fixed value. In *Threshold Management*, T_{thr} is tuned according to both migration frequency ($Migration_{\#}$) and migration limitation ($Migration_{limit}$). $Migration_{limit}$ is the maximum allowable task migration that can happen during specific iterations of the algorithm. Migrating more than $Migration_{limit}$ degrades performance and increases temperature due to the Ping-Pong effect [1]. In *Temperature Management* it is checked if cores are in *critical situations* (i.e. when there is at least one core that reaches to T_{thr} in less than t_{res} , where T_{thr} is temperature threshold at which tasks are migrated to better cores in order to reduce the temperature, and t_{res} is the response time for the algorithm to decrease the core temperature.).

If so, the algorithm reschedules and migrates the tasks, based on both application and core temperatures. After rescheduling, t_r for all cores are calculated, and if there is still any core in *critical situations*, it decreases the processor frequency (f_{cur}) to prevent violating T_{max} . In *Performance Management*, the goal is to minimize the performance degradation. In this phase, if the algorithm has not recently performed any migration and current processor frequency is lower than a predefined minimum frequency (f_{min}), it increases processor frequency to improve performance. In the following subsections, the aforementioned parts are thoroughly described.

Figure 1- The flowchart of the proposed PATM algorithm.

A. Threshold Management

Proposed task migration algorithms for DTM of multi-core processors have a temperature threshold that when the core reaches to that threshold the algorithm decides to migrate a task from a hot core to an appropriate core which results in less heat and temperature. We show that having an adaptive

threshold instead of fixed can enhance the performance of these algorithms because of following reasons.

- Temperature differential among cores of multicores

As mentioned before, due to *CUTB* of multi-core processors, the thermal behavior of cores of a processor is not the same. Fig. 2 shows some parts of our experimental results for running different applications on four different cores of an Intel Core i7-2600 processor. This figure shows the temperature behavior of cores while running *hmmer* application. At each time, one core executes the program, and the others are idle.

As illustrated in Fig.2, although all four cores have the same experimental setup, core 1 and core 3 generate lower temperature than core 0 and core 2, and their temperature difference is approximately 6°C. Suppose that the value of temperature threshold is 65°C and a task is assigned to core 3. Since, the maximum temperature of core 3 is lower than the threshold (i.e. 65 °C), the task remains on core 3 and

Figure 2- Temperature comparison among running *hmmer* on (a) core 0, (b) core 1, (c) core 2, and (d) core 3 on Intel Core i7-

peak temperature of the system will be 64°C .

Now, assume that the threshold is adjusted to 60°C , and the scheduler assigns the task to one of four cores. If the appropriate core (i.e., core 1) starts running the task, it will be finished without any migration and peak temperature will be lower than 60°C . If other cores are selected to execute the task, the task will migrate to core 1 due to reaching the temperature threshold. Note that in both cases, core 1 is where the execution of the task finishes and maximum temperature will be below 60°C . In other words, if the temperature threshold is very low, such as 40°C , the task migrates from one core to another repeatedly. This phenomenon is known as Ping-Pong effect and enforces performance degradation to system. This scenario shows that an adjustable threshold can improve peak temperature.

Fig.3 shows previous experiments on another platform (Intel Core i7-3770). As we see the temperature of cores running the same application are variant.

Figure 3- Temperature comparison among running bzip2 on (a) core 0, (b) core 1, (c) core 2, and (d) core 3 on Intel Core i7-

- Temperature differential in Simulations Multi-Threading multicores

Having a fixed threshold is also problematic for Simulations Multi-Threading (SMT) supported multi-core processor. According to our experiments SMT activation and deactivation causes the level of temperature increases or decreases.

As shown in Fig.4, enabling SMT feature on an Intel Core i7-2600 processor increases temperature and vice versa. The temperature threshold also is different for different applications. Therefore, the necessity of having adaptive threshold values for previously proposed works is felt; hence our proposed work adjusts temperature threshold according to run-time situations.

We propose an adaptive T_{thr} unlike other algorithms. Finding a proper T_{thr} is crucial. In this subsection, it is explained how the algorithm adjusts T_{thr} based on changes in workload behavior. At first T_{thr} is initialized to T_{max} . During execution, if the total number of migrations in the last M iterations of the algorithm is higher than $Migration_{limit}$, T_{thr} is incremented increases (should not become greater than T_{max}) and if the total number of migrations in the last M iterations of the algorithm is zero, T_{thr} is decremented. The higher the migration frequency is, the more overall system performance degrades. Therefore, our proposed T_{thr} management tries to control migration frequency and avoid it to increase. Note that rising T_{thr} results in decreasing migration frequency. However, increasing both T_{thr} and task migration deteriorate the overall system performance and temperature. Our proposed *Threshold*

Figure 4- Temperature comparison among running (a) hmmer, (b) gcc at two state of SMT on and SMT off.

Management finds a trade-off between temperature threshold and task migration frequency regarding workload and cores thermal behavior.

B. Temperature Management

In this section, we present temperature management algorithm for both Non-SMT multi-core processor with only one physical sensor and SMT multi-core processor with a physical sensor per core.

- Physical aware temperature management for Non-SMT multi-core processors with only one physical sensor

The main challenge of temperature-aware multi-core task scheduling is to improve the peak or average temperature with the least performance loss [1]. Since task scheduling is an NP-complete problem, it takes a long time to use classical approaches and find the best answer. Therefore, usually heuristic approaches are used to solve the problem. We tried three different heuristic strategies to find the most suitable task assignment in order to minimize the average and peak temperature while minimizing the performance degradation.

In the first strategy, cores and tasks are sorted according to their temperature from the hottest to the coldest. Performance counters are used to estimate the temperature of the cores and tasks since there is only one physical sensor for the processor. After sorting cores and tasks, the hottest task is assigned to the coldest core, then the second hottest task is assigned to the second coolest core and this process is continued.

The second strategy is similar to the first one, except that cores are sorted according to their thermal behavior based on their *CUTB* from the hottest to the coldest.

In our third strategy, we start with the coldest task that is assigned to the coldest core and so on. In the second and third strategies, sorting cores is based on their innate thermal behavior (*core unique thermal behavior*). Learning about *CUTB* can be done offline, and it is needed to be accomplished only once. The evaluations of these three strategies are reported in the experimental result section.

According to our results the second strategy is the best one. Fig.5 illustrates the second task scheduling strategy.

After rescheduling, t_r is again predicted for the processor, and if the processor is still in a critical situation, it means that *Temperature Management* cannot perfectly manage processor temperature at the software level. At this state, DVFS is used to decrease the processor frequency and hence, the temperature.

- Physical aware temperature management of SMT multi-core processor with a physical sensor per core

To enhance the performance of SMT multi-cores, the main challenge is how to co-schedule the complementary threads on individual SMT cores [23] to make better use of shared pipeline resources. However, this way of scheduling causes higher heat generation due to more pipeline resources utilization [23]. To address this issue, we study five different strategies to find the most suitable pairs of tasks that should be co-scheduled to two-context SMT cores in order to minimize the average and peak temperature, while minimizing the performance degradation. Since, each core has one physical, thermal sensor, we use performance counters to distinguish the cold thread from the hot thread on a two-context SMT core.

In the first strategy, the cores and tasks are sorted according to their temperature from the hottest to

Figure 5 – Selected task scheduling strategy for Non-SMT multi-cores

the coldest. Physical sensors and performance counters are used to measure the temperature of the cores and tasks, respectively. After sorting cores and tasks, the hottest and the coolest tasks are paired and co-scheduled to the coldest core, then the second hottest and coolest tasks are paired and co-scheduled to the second coldest core and this process is continued. The second strategy is similar to the first one, except that cores are sorted according to their thermal behavior based on their *core unique thermal behavior* (e.g. for Core i7-2600, core 3 and core 1 are always the hottest and coolest cores, respectively). In our third strategy, after sorting cores according to their thermal behavior based on their *core unique thermal behavior*, the first two hottest tasks are co-scheduled to the coldest core, then the next two hottest tasks are co-scheduled to the next coldest core. The fourth strategy is similar to the third strategy except that in this co-scheduling, the first two coldest tasks are assigned to the coldest core. In the second, third, and fourth strategies, sorting cores is based on their innate thermal behavior. Our fifth strategy reschedules tasks only between the core that is in *critical situation* (the core that has $t_r < t_{res}$) and the core that is predicted as the coolest core (the core that $t_r > t_{res}$) instead of rescheduling all tasks among all cores as done for previous four strategies. In this strategy, the coolest core has the greatest t_r among all cores. The task of the hot core is moved to the coldest core, and other cores are unchanged. The evaluations of these strategies are reported in Section IV. According to our results, the second strategy is the best one. Fig. 6 illustrates selected task scheduling strategy.

After rescheduling, t_r is again predicted for all cores, and if there is still any core in a critical

Figure 6 – Selected task scheduling strategy for SMT multi-cores

situation, it means *Temperature Management* cannot perfectly manage core temperature at the software level. At this moment, DVFS is used to decrease the processor frequency and hence, temperature.

C. Performance Management

As mentioned in the previous section, if *Temperature Management* cannot improve the *critical situation*, the processor frequency is decreased. Although this action decreases temperature significantly, it ruins system performance. Our *Performance Management* function mitigates this problem with the aid of checking the workload of the cores. If the number of migrations is zero in the last M iterations, algorithm increases the global frequency to enhance performance.

V. Experimental Results

This section provides experimental results for different applications from SPEC CPU2006 benchmarks and an analysis of the obtained results.

A. Experimental Setup

The selected programs from SPEC2006 benchmark suite are summarized in Table V, and the specifications of the selected system are depicted in Table VI. The LM sensor [24] is used to read core temperatures. We use `cpufreq` to adjust the processor frequency and use `perf` subsystem in Linux for reading performance counters. In all of our experiments the fan speed has been fixed to a constant RPM. The value of t_{res} , and $migration_{limit}$ are set to two seconds and five, respectively. These values are selected empirically based on different experiments. f_{min} is set to 2 GHz because this is a frequency that if all cores are running the applications, the maximum temperature will be less than T_{max} . The value of T_{thr} is adapted at run-time. At the start of the algorithm, T_{thr} is initialized to T_{max} . The other constant is the number of iterations of the algorithm (M) for counting $migration_{\#}$ which is set to 10. The

TABLE V
SPEC CPU 2006 applications used in experimental results

Benchmarks	hammer	libquantum	sjeng	perlbench	gobmk	gcc	mcf	bzip2
Avg. Temperature(°C)	68.2	67	65.7	65	63.9	63.9	63	62.9

TABLE VI
The systems specification used in experiments

	System #1	System #2	System #3
Core count	4	4	8
Processor	Intel Core i7-2600	Intel Core i7-3770	AMD's octa-core Bulldozer
Main memory size	4 GB	8 GB	8 GB
OS	Linux 3.2.0	Linux 3.2.0	Linux 3.2.0
SMT capability	Yes	Yes	No

temperature threshold that is assumed not to be violated (T_{max}) is 70°C.

B. Performance counter analysis

Our algorithm uses performance counters to distinguish the cold thread from the hot thread on the two-context SMT core and the Non-SMT multi-core processor with only one physical sensor (e.g. AMD Bulldozer) and be able to sort applications according to their temperature. We need to find the performance counter that has the greatest correlation with temperature of programs. To do so, we first run different programs and profile all performance counters, then select one performance counter which has the greatest correlation with the temperature of programs using Pearson Product-Moment Correlation Coefficient (PPMCC) [25]. PPMCC is used as a criterion to measure the correlation between two variables X and Y. The r coefficient is calculated using Eq. 6:

$$r = \frac{\sum_{i=1}^N (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^N (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^N (Y_i - \bar{Y})^2}}, \quad (6)$$

where N is the number of sampled data, $\bar{X}$, $\bar{Y}$ are the averages for X and Y variables, respectively. The relationship between X and Y is perfect when r is 1 or -1. Table VII summarizes the average correlations between performance counters and applications temperature of ten programs: **astar**, **libquantum**, **gcc**, **bzip2**, **mcf**, **gobmk**, **sjeng**, **h264ref**, **perlbench**, and **hmmer**.

According to Table VII *stalled-cycles-backend* event has the strongest correlation among other processor events, so our proposed algorithm uses this event as a metric to analyze the thermal behavior

of applications. The negative value implies that if X variable increases, Y will decrease. Therefore, the larger the *stalled-cycles-backend* value for an application, the colder the application and vice versa.

TABLE VII
Correlation between events and applications temperature.

Events	Correlation	Events	Correlation
stalled-cycles-backend	-0.37	Branches	-0.03
cache-references	-0.35	page-faults	-0.01
stalled-cycles-frontend	-0.35	branch-misses	0.02
cache-misses	-0.33	CPU-migrations	0.04
Cycles	-0.29	Instructions	0.29
task-clock	-0.24	IPC	0.30

We set up an experiment to demonstrate the effect of choosing different events on results. Fig. 7 illustrates the average temperature of four cores while PATM once uses *stalled-cycles-backend* (highest correlation), and once uses *page-faults* (lowest correlation) as the events to measure application thermal behavior, respectively, and sort them from the hottest to the coldest. It is observed that using *stalled-cycle-backend* event improves peak temperature and average temperature by about %6 (3°C) and %4. 7 (2.5°C) respectively, compared to the case when the lowest correlation counter is used.

C. Task scheduling analysis

The five strategies for task scheduling in *Temperature Management* phase (see section IV.B) are

Figure 7 - PATM average temperature using high and low correlation counter for application ordering and Linux standard scheduler.

tried, and their results are compared against Linux scheduler. In each strategy, we study four cases i.e. we execute five to eight different benchmarks simultaneously. The average results of four cases are shown in Fig. 8. As can be seen, the second strategy has the best average and peak temperature improvement but there is about 0.38% performance overhead. The first strategy improves the average and peak temperature less than the second strategy but not only it does not degrade performance but also it improves it by about 0.76%. Since, the focus of this paper is thermal management, we continue with the second strategy for the following experiments. The results show the effectiveness of considering *core unique thermal behavior* of the algorithm on the results.

D. Adaptive threshold analysis

To evaluate the effectiveness of applying adaptive T_{thr} against fixed T_{thr} , we compare the results with the case where the *Threshold Management* section of the algorithm is disabled. Fig. 9 shows the results. Using an adaptive T_{thr} , 1.7% (0.9°C) and 6.3% (4°C) improvement is obtained on average and peak temperature, respectively compared to the fixed T_{thr} .

E. Temperature prediction analysis

Our temperature prediction model based on Eq. 2 predicts future temperature with less than 1°C mean absolute error on running different benchmarks. Fig. 10 illustrates the results of the prediction model of

Figure 8 - performance, average, and peak temperature improvement of different strategies compared to Linux standard scheduler.

Figure 9 - Comparison of our proposed algorithms for two cases with adaptive and non-adaptive thresholds.

TAS [17] vs. ours against the real core temperature. Using our predictor, the mean absolute error (MAE) is 0.6 °C whereas MAE of TAS predictor is 0.8 °C on running `gcc` and `hmm` programs simultaneously, which shows the effectiveness of newly added w parameter to our predictor.

F. Thermal management results

We have done two experiments on the two SMT supported processors (Intel Core i7-2600 and Core i7-3770) and one experiment on the Non-SMT processor (AMD Bulldozer). The first experiment is performed with simultaneous execution of five programs and the others with eight programs. Another experiment is done on AMD's octa-core Bulldozer and is compared to the Linux scheduler because this processor has just one temperature sensor and algorithms such as PDTM and TAS need per core temperature sensor.

Figure 10 - The comparison of PATM and TAS predictors.

In the first test, five programs including gcc, libquantum, mcf, hmmer, and bzip2 are run simultaneously, on two platforms (Intel Core i7-2600, Intel Core i7-3770) with four different schedulers. Our proposed method is compared with Standard Linux scheduler, PDTM, and TAS. Fig. 11 and Fig. 12 show the cores temperature of different schedulers on Core i7-2600, and Intel Core i7-3770, respectively. This experiment is repeated for eight programs. The results are depicted in Fig. 13 and Fig. 14, respectively. Fig. 15 illustrates processor temperature of PATM and Linux scheduler on AMD's octa-core Bulldozer.

Figure 11 - The cores temperature of different scheduler a) Linux scheduler, b) PDTM, c) TAS, d) PATM, running five programs on Intel Core i7-2600.

Figure 12 - The cores temperature of different scheduling a) Linux scheduler, b) PDTM, c) TAS, d) PATM, running five programs on Intel Core i7-3770.

Figure 13 - The cores temperature of different scheduler a) Linux scheduler, b) PDTM, c) TAS, d) PATM, running eight programs on Intel Core i7-2600.

Figure 14 - The cores temperature of different scheduling a) Linux scheduler, b) PDTM, c) TAS, d) PATM, running eight programs on Intel Core i7-3770.

Figure 15 - The processor temperature of PATM vs. Linux scheduler by a) five, b) eight programs on AMD's octa-core Bulldozer.

After running various set of programs on three different processors, it is observed that our proposed technique (PATM) reduces the average temperature by about 1.6% (0.8°C), 1.9% (0.9°C), and 4.4% (2.1°C) compared to PDTM, TAS, and Linux scheduler and reduces peak temperature by almost 4.4% (2.1°C), 6% (2.8°C), and 8.9% (4.2°C) with 0.8%, 0.4%, and 0.9% performance (run-time) overhead, respectively. A detailed comparison is depicted in Table VIII. It should be noted that the average temperature is the mean of four cores temperature running programs simultaneously from beginning to the end. Hence, compared to Linux, PDTM and TAS, our proposed method indeed leads to more peak temperature reduction with negligible performance overhead. In the Linux scheduler, one core starts program execution and terminates it due to high migration threshold assignment. Therefore, it does not use other cooler cores to decrease the hot core temperature. PDTM tries to mitigate problem using core temperature prediction; however, it cannot find a proper core when all cores temperatures are near to the temperature threshold (70°C). In such circumstances, the task migrates between different cores repeatedly. This phenomenon is known as Ping-Pong effect [1]. TAS categorizes applications based on their thermal behavior to improve prediction accuracy. It defines an appropriate core as a core that reaches to migration threshold later. Our proposed technique improves TAS algorithm using an adjustable threshold scheme and an improved predictor. As shown before, adjustable threshold leads the scheduler to assign tasks to a core which reaches T_{ss} as late as possible.

TABLE VIII
A detailed comparison of average and max temperature and performance overhead among PDTM, TAS, Linux, and PATM on an SMT processor.

	Average Temp. (°C)	Max Temp. (°C)	Run Time (sec)
Improvement Compared to PDTM	1.6% (0.8°C)	4.4% (2.1°C)	0.8%
Improvement Compared to TAS	1.9% (0.9°C)	6% (2.8°C)	0.4%
Improvement Compared to Linux	4.4% (2.1°C)	8.9% (4.2°C)	0.9%

VI. Conclusion

In this paper, a dynamic thermal management algorithm with a future temperature predictor for both SMT and Non-SMT multicore processors is presented. The proposed algorithm manages processor temperature taking into account the workload and *core unique thermal behavior*. As demonstrated, considering SMT, *core unique thermal behavior*, applications thermal behavior, and adaptive migration threshold ability are extremely important in the DTM, and they have a significant influence on performance and temperature management. Experimental results obtained by SPEC CPU2006 running on three different desktop platform indicate that our algorithm outperforms Linux standard scheduler, TAS, and PDTM in terms of thermal management with negligible performance overhead.

References

- [1] J. Kong, S.W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Computer Survey*, vol. 44, no. 3, pp. 13:1-13:42, 2012.
- [2] Lee, Eun Kyung, Indraneel Kulkarni, Dario Pompili, and Manish Parashar, "Proactive thermal management in green datacenters," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 165–195, 2012.
- [3] Guanglei Liu, Ming Fan, and Gang Quan, "Neighbor-aware dynamic thermal management for multi-core platform," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 187-192, 2012.
- [4] Vinay Hanumaiah, Sarma Vrudhula, and Karam S. Chatha, "Performance Optimal Online DVFS and Task Migration Techniques for Thermally Constrained Multi-Core Processors," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1677-1690, 2011.
- [5] Qiong Cai, Jos'e Gonz'alez, Grigorios Magklis, Pedro Chaparro, and Antonio Gonz'alez, "Thread shuffling: combining DVFS and thread migration to reduce energy consumptions for multi-core systems," in *Low Power Electronics and Design*, pp. 379-384, 2011.
- [6] Buyoung Yun, Kang G. Shin, and Shige Wang, "Thermal-Aware Scheduling of Critical Applications Using Job Migration and Power-Gating on Multi-core Chips," in *Trust, Security*

and Privacy in Computing and Communications, pp. 1083-1090, 2011.

- [7] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Hendrik Hamann, Alan Weger, and Pradip Bose, "Thermal-aware task scheduling at the system software level," in Proceedings of the 2007 international symposium on Low power electronics and design, pp. 213-218, 2007.
- [8] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, and Lingling Jin, "Dynamic Thermal Management through Task Scheduling," in Performance Analysis of Systems and software, pp. 191-201, 2008.
- [9] Bagher Salami, Mohammadreza Baharani, and Hamid Noori, "An adaptive temperature threshold schema for dynamic thermal management of multi-core processors," in Computer Architecture and Digital Systems (CADS), 2013 17th CSI International Symposium on, pp. 119-120, 2013.
- [10] Wu, Guowei, Zichuan Xu, Qiufen Xia, Jiankang Ren, and Feng Xia, "Task Allocation and Migration Algorithm for Temperature-constrained Real-time Multi-Core Systems," in the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, pp. 189-196, 2010.
- [11] Gabriel Marchesan Almeida, Sameer Varyani, Rémi Busseuil, Gilles Sassatelli, Pascal Benoit, and Lionel Torres, "Evaluating the impact of task migration in multi-processor systems-on-chip," in Proceedings of the 23rd symposium on Integrated circuits and system design, pp. 73-78, 2010.
- [12] Pierre michaud, Andre seznec, Damien fetis irisa/inria, Yiannakis sazeides, and Theofanis constantinou, "A Study of Thread Migration in Temperature-Constrained Multicores," ACM Transactions on Architecture and Code Optimization, vol. 4, no. 2, 2007.
- [13] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks, "Thread Motion: Fine-Grained Power Management for Multi-Core Systems," ACM SIGARCH Computer Architecture News, Vol. 37, No. 3, pp. 302-313, 2009.
- [14] Sironi, Filippo, et al. "ThermOS: System support for dynamic thermal management of chip multi-processors." Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on. IEEE, 2013.
- [15] A. Kumar, L. Shang, L. Peh, and N.K. Jha, "HybDTM: a coordinated hardware-software approach for dynamic thermal management," In Proc. of the 43rd Design Automation Conference, pp. 548-553, 2006.
- [16] I. Yeo, C.C. Liu, and E.J. Kim, "Predictive dynamic thermal management for multicore systems," In Proc. of the 45th annual Design Automation Conference, pp. 734-739, 2008.

- [17] I. Yeo, E.J. Kim, "Temperature-aware scheduler based on thermal behavior grouping in multicore systems," In Proc. of the Conference on Design, Automation and Test in Europe, pp. 946-951, 2009.
- [18] E. Kursun, and C-Y. Cher, "Variation-aware thermal characterization and management of multi-core architectures," In Proc. of International Conference on Computer Design(ICCD), pp. 280-285, 2008.
- [19] Sheikh, Hafiz Fahad, Ishfaq Ahmad, Zhe Wang, and Sanjay Ranka, "An overview and classification of thermal-aware scheduling techniques for multi-core processing systems," Sustainable Computing: Informatics and Systems, 2011.
- [20] Bao, Min, Alexandru Andrei, Petru Eles, and Zebo Peng. "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration." In Proceedings of the 46th Annual Design Automation Conference, pp. 490-495. ACM, 2009.
- [21] Su, Bo, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. "PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration." In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 445-457. IEEE Computer Society, 2014.
- [22] Coskun, Ayse Kivilcim, Tajana Simunic Rosing, and Kenny C. Gross, "Proactive temperature management in MPSoCs," in Proceedings of the 13th international symposium on Low power electronics and design, pp. 165-170, 2008.
- [23] M. Gomaa, M.D. Powell, and T.N. Vijaykuma, "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," In ACM SIGARCH Computer Architecture News, pp. 260-270, 2004.
- [24] Lm sensors linux hardware monitoring [Online]. Available: <http://www.lm-sensors.org>.
- [25] J.L. Rodgers, and W.A. Nicewander. "Thirteen ways to look at the correlation coefficient," The American Statistician 42. no 1, pp. 59-66, 1988.