A Physical-Aware Task Migration Algorithm for Dynamic Thermal Management of SMT Multi-core Processors

Abstract - This paper presents a task migration algorithm for dynamic thermal management of SMT multi-core processors. The unique features of this algorithm include: 1) considering SMT capability of the processors for task scheduling, 2) using adaptive task migration threshold, and 3) considering cores physical features. This algorithm is evaluated on a commercial SMT quad-core processor. The experimental results indicate that our technique can significantly decrease the average and peak temperature compared to Linux standard scheduler, and two well-known thermal management techniques.

I. Introduction

As feature size is shrinking, the ability to have processors with larger number of cores is increasing. By advent of Simultaneous Multi-Threading (SMT), the multi-core processors can exploit more thread-level parallelism by less hardware compared to non-SMT multi-core processors. SMT multi-cores are becoming the main trend in the new generations of processors. However, due to increased density and complexity of these processors, the SMT multi-cores power consumption is increasing. The high power consumed in a small area die size results in increasing power density and generated temperature. Therefore, an expensive processor packaging and cooling equipment are needed to remove hot spots. Moreover, increasing temperature potentially threatens system reliability, decreases both transistor age and transition speed and increases leakage current [1]. Therefore, thermal management at all levels of system design is crucial.

Dynamic Thermal Management (DTM) techniques are proposed to mitigate the aforementioned problems. DTM is a set of techniques that control processor temperature at run-time so that temperature does not go beyond a certain value known as critical temperature threshold. The DTM techniques are available at both hardware (HW) and software (SW) levels. Although HW approaches, such as stop-and-go [1], clock gating [1], and Dynamic Voltage and Frequency Scaling (DVFS) [2-3], decrease temperature greatly, they lengthen execution time, thus degrading overall system performance. On the other hand, software-based DTM techniques such as task scheduling [3-4] and task migration [5-6] can reduce temperature without significant performance degradation and do not require extra hardware.

Lately, researchers proposed different instruments and algorithms for core and application thermal measurement and prediction to manage processors temperature efficiently. Two well-known methods, named as CMOS thermal sensors and performance-counter-based (software-based) sensors [1] are used to measure and predict processor thermal pattern. Alongside, application thermal profiling and performance counters [1] are other two methods for application thermal categorization. Since application thermal profiling is an offline method, it cannot reflect the real thermal pattern of processors. Performance counters are usually used for online application temperature prediction, though they are inaccurate [7]. Moreover, reading different performance counters imposes significant overhead on application execution at run-time [1]. Therefore, recent proposed methods [8-9] model overall core and application temperature with aid of physical sensor and steady state temperature. Nevertheless, application and core temperature estimation of an off-the-shelf SMT multi-core processor based on only physical temperature sensors is inaccurate, because generally each core of an SMT multi-core processor has only one physical temperature sensor and it is hardly possible to know the real temperature of each thread.

Among DTM techniques, some of them [3-4,7,10-12] are targeting temperature management of SMT processors, while others [5-9] do not leverage SMT capability. DTM techniques for SMT processor can be divided into two categories: a) algorithms [3,10-12] that are proposed and evaluated on simulators, and b) algorithms [4,7] that are proposed and evaluated on real platforms.

Qiong et al. [3] introduce a simulation-based technique for parallel applications, called thread shuffling. This technique dynamically maps threads with similar criticality degrees into the same core and then applies DVFS to non-critical cores which execute fast threads. Their use of local DVFS restricts the proposed algorithm to only a few specific processors. Jeonghwan et al. [4] present a DTM method, so called Cool Loop technique for commercial single core SMT processors.

Recent works predict future temperature core to reduce overheat temperature with negligible performance overhead. Their proactive task migration approaches estimate the future temperature using regression, and manage the workload to reduce and balance the temperature before reaching the temperature threshold. PDTM [8] is one of first attempts that predicts core temperature. The prediction is based on both application thermal and core thermal models. PDTM migrates applications from the possible overheated core to the future coolest core in order to maintain system temperature below a threshold temperature. On the other hand, TAS [9] categorizes applications according to their thermal behavior for improving accuracy of temperature prediction.

Different cores of a processor do not have similar thermal behavior due to process variation [13], the temperature effect of core neighbors [2], and other physical issues [1]. The

temperature difference among cores of a processor running the same application can be as much as 10~15°C [8]. In this paper we name these phenomena as physical features of cores. It means the cores of a single chip processor show different thermal behavior for the same workload.

Motivated by these facts, we propose an algorithm which considers different thermal behavior of cores (physical features of cores) and uses both physical sensors and performance counters simultaneously to improve thermal management of SMT multi-core processor. We utilize physical sensor to estimate and predict the future temperature of cores and performance counter to classify the applications thermal behavior at runtime. Another unique feature of the proposed algorithm is that unlike all other proposed algorithms it has an adaptive migration threshold. To the best of authors' knowledge, no prior attempt has been made to implement a thermal-aware task scheduling on a commercial SMT quad-core product (Core i7-3770) under Linux environment. The experimental results on Intel's Core i7-3770 running five to eight benchmarks indicate that our proposed method outperforms both Standard Linux scheduler, PTDM and TAS in reducing average and peak temperatures. The main contributions of this paper are summarized as follows:

- We propose a thermal-aware scheduling for multi-core SMT supported processors based on different thermal behavior of cores due to their physical features.
- Our experimental results on commercial processors indicate that our proposed approach, under full workloads, outperforms the Linux standard scheduler and two existing DTM techniques (PDTM, 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 reminder of this paper is organized as follows: The preliminaries of our algorithm are presented in Section 2. Section 3 describes our proposed algorithm. Implementation and analysis results are shown in Section 4 and finally conclusions are drawn in Section 5.

II. Preliminary

In this section, the preliminary of proposed algorithm is discussed.

A. Problem description

The system considered in this paper consists of an SMT multi-core processor with N cores, denoted as {$core_1$, $core_2$, …, $core_N$ } which each core can execute up to two threads simultaneously. It is also assumed that there are N+1 to 2×N tasks for execution. The reason is that, since in this paper we focus on SMT feature of the processors, the number of tasks should be more than number of the physical cores so that using SMT capability makes sense. The problem discussed in this paper is how to schedule these tasks among cores dynamically such that the average and peak temperature of the system can be minimized under minimum performance loss and also temperature does not exceed T_{max}. T_{max} is the maximum allowable temperature. In this paper, we propose a heuristic method to solve the above problem based on task migration and DVFS. We first introduce a new temperature prediction method, which predicts the future temperature of a core by considering both core physical feature and workload of processor. Temperature management is activated when there is at least one core that reaches to T_{thr} in less than t_{res}, where T_{thr} is the temperature threshold which triggers task migration to act and migrate applications to better cores in order to reduce the temperature, and t_{res} is a constant that shows the response time for the algorithm to take an action to decrease the core temperature.

B. Physical features of cores

As mentioned earlier, the temperature of each core of a processor is different from other cores because of packaging technology [8], process variation [13] and the thermal effects of neighbor components [2]. Table I summarizes our experimental results of running different applications on four different cores of two Intel quad-core (Core i7-2600 and Core i7-3770) processors. This Table shows the thermal behavior of cores while one core executes an application and the others are idle while the fan speed is fixed. This experiment has been done for three applications. Note that the reported temperature is the maximum temperature among all four cores. As an instance, 71°C is the peak temperature among all cores of Core i7-2600, when core 3 executes the bzip2 benchmark.

According to Table I, although all four cores have the same experimental setup, for Core i7-2600, core 3 and core 1 are always the hottest and coolest cores, respectively. We tried the same experiment with Core i7-3770 and we again observed this differential cores thermal behavior. As can be seen in Table I, core 2 and core 3 are the hottest and coolest cores respectively for Core i7-3770. This phenomenon, which we refer it as physical feature of multi-core processors, motivated us for our proposed DTM algorithm. In the rest of paper, we fully explain how we take advantage of physical feature to enhance the thermal management of SMT processors.

TABLE I
Temperature differential between cores. Results are extracted for Intel core i7-2600 and core i7-3770.

Benchmark	Intel Core i7-2600			
	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
	Intel Core i7-3770			
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

C. Temperature prediction

Our temperature predictor is modified version of [9]. Let assume T_{ss} as steady state temperature of an application (The steady state temperature of an application is defined as temperature the system would reach if application is executed infinitely [8]). According to [9] the rate of temperature changes 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 core-specific constants. We add a new parameter w to Eq. 1 to extract Eq. 2:

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

where w relates core activity. w is added to reflect the thermal effects of other cores that are active (running applications). The value of c, and w are determined offline using SPEC CPU2006 benchmarks.

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 can obtain:

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

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 independently. Based on the value of t_r the proposed algorithm decides when to start task migration and rescheduling. In the next section our proposed algorithm is fully discussed.

III. Proposed Algorithm

This section discusses the proposed algorithm for dynamic thermal management of SMT multi-core processors. In following subsections, different parts of algorithm are fully studied.

The flowchart of the proposed algorithm is depicted in Fig.1. The main parts of algorithm are: *Threshold Management*, *Temperature Management*, and *Performance Management*. In threshold management part, T_{thr} is tuned according to both migration frequency ($Migration_{\#}$) and migration limitation ($Migration_{limit}$). The migration limitation is the maximum allowable task migration that can happen during specific time intervals. In critical situation, the algorithm reschedules and moves tasks based on both application and core temperature. Again after rescheduling, t_r for all cores are calculated, and if there is still one core in critical situation, it decreases the core frequency (f_{cur}) to manage processor temperature. In performance management, if algorithm has not performed any migration in the recent past and current core frequency is lower than predefined minimum frequency (f_{min}), it increases core frequency to improve performance. In following subsection, the aforementioned parts are thoroughly studied.

A. Threshold Management

T_{thr} can be affected dramatically by dynamic behavior of runtime workloads and different physical features of hardware platforms from one processor to another. Therefore finding a proper threshold is crucial. In this subsection, it is explained how the algorithm adjusts T_{thr} based on changes of workload behavior. If the total number of migrations in the last period is higher than $Migration_{limit}$, T_{thr} increases and if the total number of migrations in the last period is lower than $Migration_{limit}$, T_{thr} decreases. The higher the migration frequency, the more overall system performance degrades. Therefore, our proposed threshold management tries to control migration frequency and prevent it from increasing. Note that the higher temperature threshold causes the migration frequency to decrease. However, both temperature threshold and task migration increment deteriorate the overall system performance and reliability. Our proposed Threshold management finds a trade-off between temperature threshold and task migration regard to workload and core thermal behavior.

B. Temperature Management

A main challenge in task scheduling of SMT multi-core is co-scheduling of complement threads on individual SMT cores to make better use of shared pipeline resources in order to improve performance. However, this scheduling produces more temperature dissipation due to more pipeline resources utilization. [6] To overcome this problem we try five different strategies to determine which pairs of tasks increase performance while minimizing the average and peak temperature. Temperature management pairs and selects application based on their behavior and orders them from hot to cold one.

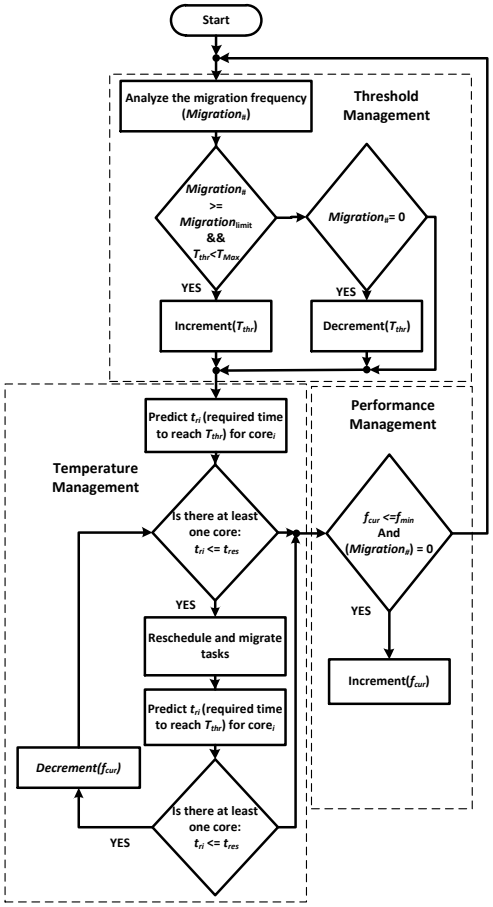

Figure 1- The flowchart of the proposed PATM algorithm.

Since most modern processors provide performance counters to allow monitoring of specific hardware events for the purpose of debugging and system tuning, in this paper performance counters are used to monitor the application behavior. The performance monitoring hardware broadly consists of event detectors and event counters. The event detectors can be configured to detect several hardware events, such as cache misses, pipeline stalls, branch misses, committed instructions, etc. We use Pearson Product-Moment Correlation Coefficient (PPMCC) or *Pearson's r* [14] as a criterion to measure the correlation between two variables X and Y, where in our case X is the core temperature and Y is the number of events detected by performance counters. According to *r* coefficient, stalled-cycle-backend has the strongest correlation with core temperature. In section 4 the results of Pearson's r correlation for different events detected by performance counters are fully presented.

The five strategies are tried and their results are compared against Linux scheduler. In each strategy we executed selected programs (between 5 to 8 different benchmarks) simultaneously. At the first strategy, cores are sorted based on their temperature which is read by physical sensors and tasks are arranged based on stalled-cycle-backend events. After sorting cores and tasks, hottest and coolest tasks are paired and assigned to coldest core, second hottest and coolest task are again paired and assigned to second coolest core and this action is repeated. As mentioned earlier, the temperature of each core is different from each other (physical feature) and there are always hottest and coolest cores. Second strategy is similar to the first one, except that cores are sorted based on their physical feature. In our third strategy after sorting cores according to their physical features and tasks, first two hot tasks are assigned to coldest core. Fourth strategy is similar to the third scenario except that in assignment, first two cold tasks is assigned to coldest core. Since temperature management is activated when there is at least one $core_i$ that reach to T_{thr} in less than t_{res}, instead of rescheduling all tasks similar to four previous strategy, our fifth strategy reschedules tasks between only critical core ($t_i < t_{res}$) and predicted cold core ($t_r > t_{res}$). In this strategy, the coolest core has the greatest t_r among all cores. Task will be moved from hot core to cold core. Other cores which are not in critical mode will be unchanged.

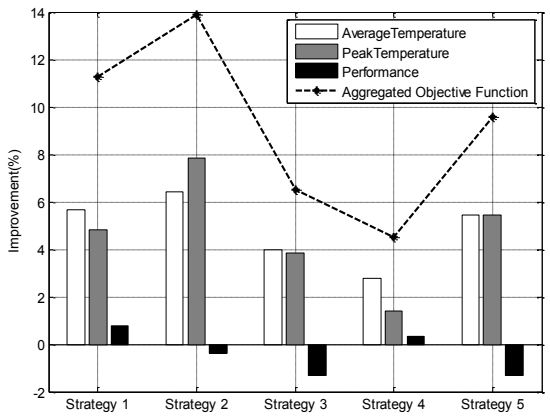

Figure 2 - performance, average and peak temperature improvement of different strategies compared to Linux standard scheduler.

As can be seen from Fig.2 scenario 2 has the most average and peak temperature improvement but there is about 0.38% performance overhead. Scenario 1 improves the average and peak temperature less than scenario 2 but performance has increased about 0.76%.

Since the problem of thermal-aware scheduling on multi-core processors is a multi-objective optimization (MOO) problem, there is not unique solution and it can be vary from one purpose to another ones. Designers can use *weighting approach* technique for optimization problem (maximizing average and peak temperature improvement with minimum performance overhead) to decide which one of two first strategies satisfies their system demands. In the rest of this paper, we used scenario two. Fig.3 illustrate selected task scheduling strategy.

After rescheduling tasks, t_r is again predicted for all cores, and if there is still one core in critical situation, it means temperature management cannot perfectly manage core temperature at software level. At this moment, it uses DVFS technique to decreases the processor frequency.

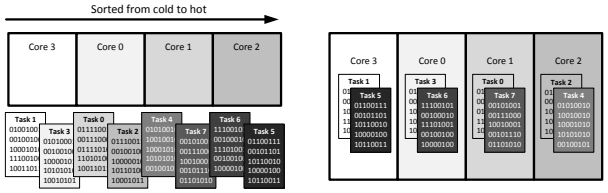

Figure 3 – Second task assignment strategy.

C. Performance Management

As mentioned in previous section, if temperature management cannot improve critical situation, it decreases the processor frequency. Although this action decreases temperature significantly, it ruins overall system performance. Our performance management function mitigates this problem with the aid of checking the workload of cores. If the number of migrations is zero and current cores frequencies are lower than f_{min}, algorithm increases the global frequency to enhance system performance.

IV. Experimental Results

This section provides experimental results under heavy workload (between five to eight applications) of different applications from SPEC CPU2006 benchmarks. In the rest of this section we describe experiment environment and analyze the obtained results.

A. Experimental Setup

The selected benchmark programs are summarized in Table II. These benchmarks are executed simultaneously on the processor. The processor we use is an Intel Core i7-3770 while the SMT capability of processors is activated. The size of the main memory of the system is 8 GB. The kernel version of Linux is 3.2.0. The LM sensor [15] application is used to read the temperature of the cores. We use cpufreq tool to adjust the processor frequency. In all of our experiments the fan speed has been fixed to a constant RPM (Rotation per Minutes). The value of tres, and migration# are set to 2

Table II
SPEC CPU 2006 benchmarks used in experimental results

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

seconds and 5 respectively, these values are selected empirical based on different experiments. f_{min} is set to 2 GHz because this is a frequency that if all cores are running applications, the maximum temperature will be less than critical temperature. Note that the value of w, and T_{thr} are adaptive and modified by algorithm at run-time. The other tentative constant of our algorithm is number of intervals for counting $migration_{\#}$ is set to 10. The temperature threshold that we do not want to violate is 70°C.

B. *Performance counter analysis*

Table III summarizes the correlations between core temperature and performance counter running ten benchmarks: gcc, libquantum, astar, bzip2, mcf, gobmk, sjeng, h264ref, perlbench, and hmmer.

As can be seen from Table FF, since stalled-cycle-backend event has the strongest correlation (absolute value is considered) among other processor events, 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.

TABLE II
Correlation between different events and core temperature.

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

We set up an experiment to demonstrate the effect of choosing different events on final algorithm outputs. Fig 4

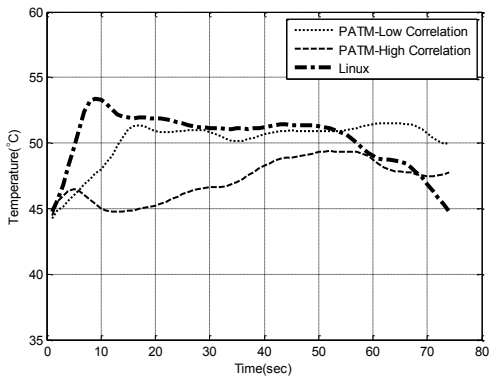

Figure 4 - Comparison of average temperature of cores in our proposed algorithm at two state of using high and low correlation counter for application ordering and Linux standard scheduler by running 6 programs.

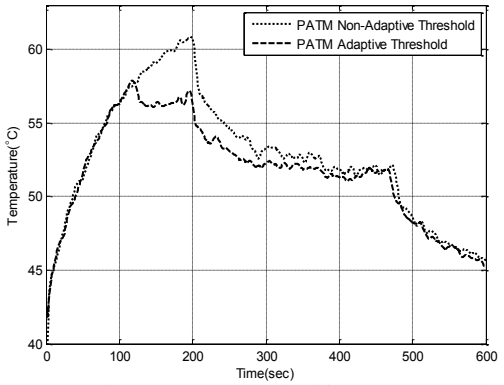

Figure 5 - Comparison of our propose algorithms at two state of adaptive and non-adaptive threshold

illustrates the average temperature of four cores while task assigned by Linux standard scheduler and our proposed algorithm. For PATM, task assignment used stalled-cycle-backend (high correlation), and page-faults (low correlation) as event to analyze application behavior and order them from hot to cold. As can been seen, using stalled-cycle-backend events can help algorithm to reduces temperature more efficiently.

C. *Adaptive threshold analysis*

For evaluating having adaptive threshold how much can improve temperature of system, we set up an experiment which at two state of fix and adaptive temperature threshold to extract amount of improvement at each state that depicted at Fig 5.

D. *Temperature prediction analysis*

Our temperature prediction model based on equation (2) predicts future temperature with less than 1°C least square error on running different benchmarks. Fig.6 illustrates the accuracy of our prediction model against the real core temperature with only 0.679 °C mean absolute error on

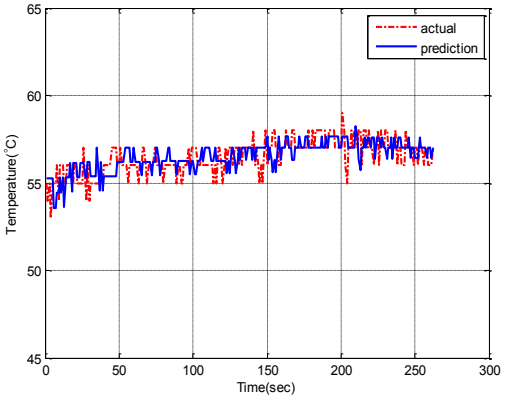

Figure 6 - The predicted model can estimate future temperature while its MAE is 0.679°C.

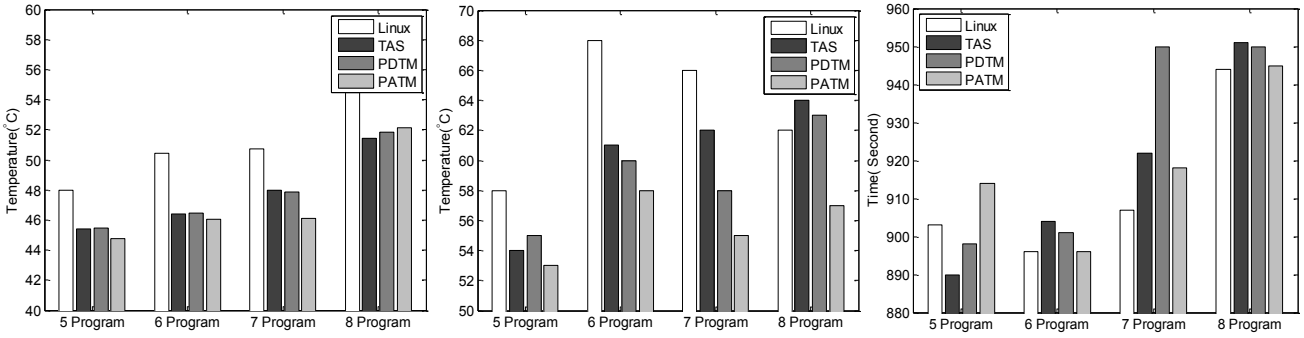

Figure 7 - (a) aveeage temperature, (b) peak temperature and (c) run-time comparison of different algorithms with different number of programs.

running bzip2 benchmark.

E. Thermal management results

Fig. 7 illustrates cores temperature for TAS, PDTM, Linux standard scheduler, and our proposed algorithm on an Intel core i7-3770. The temperatures are sampled every second.

Running a different set of programs on Intel Core i7-3770, our proposed PATM reduces average temperature (average temperature of four cores from begin to end of running five benchmarks simultaneously) about 7.7% (3.6°C), and reduces peak temperature about 13.9% (7.8°C) with 1.7% performance (run-time) overhead compared to standard Linux scheduler. The experimental results also indicate that our proposed algorithm reduces average temperature about 1.1% and 1.3% compared to TAS and PDTM, respectively. ATDTM reduces peak temperature about 8.1% and 5.8% compared to both TAS and PTDM. The overall system performance (run-time) overhead is only about only 1.3% and 0.4% compared to TAS and PDTM. Table X summarizes the comparison results for these four algorithms. The reported results in Table III are mean values extracted from running five to eight benchmarks.

Hence, by comparing with the Linux, PDTM, TAS scheduling method used before, our proposed method indeed lead to more significant peak temperature reduction with only negligible performance overhead.

TABLE III
COMPARISON OF PROPOSED ALGORITHM AGAINST LINUX, TAS, AND PDTM.

DTM Algorithm	Average Temp.	Max Temp.	Run Time(Second)
PDTM	47.9(°C)	59(°C)	924.8(Sec)
TAS	47.8(°C)	60(°C)	916.8(Sec)
PATM	47.3(°C)	58(°C)	928.3(Sec)
Linux	50.9(°C)	64(°C)	912.5(Sec)
Improvement of PATM vs. PDTM	1.3%	1.7%	-0.4%
Improvement of PATM vs. TAS	1.1%	3.9%	-1.3%
Improvement of PATM vs. Linux	7.7%	9.5%	-1.7%

V. Conclusion and Future Work

In this paper, a dynamic thermal management algorithm with a future temperature prediction for multicore SMT-supported processor is presented. The proposed algorithm manages processor temperature in regard to workload and physical feature of cores. As demonstrated, physical feature and application ordering are extremely important in DTM and they have influence on performance and temperature management techniques. Experimental results based on practical benchmarks (SPEC CPU2006) running on a desktop platform (Intel Core i7-3770) indicate that our algorithm can overcome Linux standard scheduler, TAS, and PDTM with negligible performance overhead. For the future work, we will test our schemes in different platforms with various benchmarks such as JBB2005, and WEB2005 to verify their scalability in more general environment.

References

[1] J. Kong, SW Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Computer Survey*, vol. 44, no. 3, pp. 13:1-13:42, 2012.

[2] G. Liu, M. Fan, and G. Quan, "Neighbor-aware dynamic thermal management for multi-core platform," *in DATE*, pp. 187–192, 2012.

[3] Q. Cai, J. Gonzalez, G. Magklis, P. Chaparro, and A. Gonzalez, "Thread shuffling: Combining DVFS and thread migration to reduce energy consumptions for multi-core systems," *in Proc. of ISLPED*, pp. 379-384, 2011.

[4] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, P. Bose, "Thermal-aware task scheduling at the system software level," *in ISLPED*, pp. 213-218, 2007.

[5] Z. Liu, T. Xu, S.X.-D. Tan, and H. Wang "Dynamic thermal management for multi-core microprocessors considering transient thermal effects," *in* ASP-DAC. 2013.

[6] 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 ASPLOS*, pp. 260-270, 2004.

[7] A. Kumar, L. Shang, L. Peh, and N.K. Jha, "HybDTM: a coordinated hardware-software approach for dynamic thermal management," *in DAC*, pp. 548-553, 2006.

[8] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," *in DAC*, pp. 734-739, 2008.

[9] I. Yeo, E. Jung Kim, "Temperature-aware scheduler based on thermal behavior grouping in multicore systems," in DATE, pp. 946-951, 2009.

[10] J. Donald, and M. Martonosi, "Leveraging Simultaneous Multithreading for Adaptive Thermal Control," Workshop on Temperature-Aware Computing Systems, 2005.

[11] J. Donald , M. Martonosi, "Temperature-aware design issues for smt and cmp architectures," In Proceedings of the Workshop on Complexity-Effective Design (WCED). ACM Press, 2004.

[12] J. Donald , M. Martonosi, "Heat-and-run: leveraging smt and cmp to manage power density through the operating system," SIGOPS Oper. Syst. Rev., vol. 38, 2004.

[13]Kursun, Eren, and Chen-Yong Cher. "Variation-aware thermal characterization and management of multi-core architectures," *in* ICCD 2008, pp. 280-285, 2008.

[14] L. Rodgers, Joseph, and W.A. Nicewander. "Thirteen ways to look at the correlation coefficient," The American Statistician 42. no 1, pp. 59-66, 1988.

[15]Lm sensors linux hardware monitoring [Online]. Available: http://www.lm-sensors.org.