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

Bagher Salami †, Mohammadreza Baharani §, Hamid Noori †, Farhad Mehdipour ‡
† School of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
§ School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
‡ E-JUST Center, Graduate School of Information Science and Electrical Eng., Kyushu University, Fukuoka, Japan
Email: bagher.salami@stu-mail.um.ac.ir, m.baharani@ut.ac.ir, hnoori@um.ac.ir, farhad@ejust.kyushu-u.ac.jp

Abstract - This paper presents a task migration algorithm for dynamic thermal management of Simultaneous Multi-Threading (SMT) multi-core processors. The unique features of this algorithm include: 1) considering SMT capability of processors for dynamic thermal management via 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 the 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 the 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, 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 and Dynamic Voltage and Frequency Scaling (DVFS), decrease temperature significantly, while degrades overall system performance due to longer execution time [1]. On the other hand, software-based DTM techniques such as task scheduling [2-3] and task migration [4-9] can reduce the temperature without significant performance degradation and any extra hardware.

Among DTM techniques, some of them are targeting temperature management of SMT processors [2, 3, 6, 7], while others [4, 5, 8, 9] do not leverage SMT capability. DTM techniques for SMT processors can be divided into two categories: a) algorithms that are proposed and evaluated on simulators [2, 6], and b) algorithms that are proposed and evaluated on real platforms [3, 7].

The work of [2] introduces 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. [3] presents a DTM method for commercial single core SMT processors.

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 are other two methods for application thermal categorization [1]. Since application thermal profiling is an offline method, it cannot reflect the real thermal pattern of the processor. 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 model overall core and application temperature with the aid of physical sensor and steady state temperature [8, 9]. Nevertheless, application temperature estimation of off-the-shelf SMT multi-core processors 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.

Recent works predict future temperature of cores to reduce overheat temperature with negligible performance overhead. Their proactive task migration approaches predict the future temperature and manage workload to reduce and balance the temperature before reaching the temperature threshold. PDTM [8] is one of the 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 the processor temperature below a threshold temperature. TAS [9]

categorizes applications according to their thermal behavior and [4] considers the temperature of neighbor cores for improving the accuracy of temperature prediction.

Different cores of a processor do not have similar thermal behavior due to process variation [12], the temperature effect of neighbor components [4], 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 [8]. In this paper, we name these phenomena as *physical features of cores*. It means that the cores of a multi-core 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 processors. 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 unique feature of proposed algorithm is that unlike all other previous 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 considering SMT capability. The experimental results on Intel's Core i7-3770 running five to eight benchmarks indicate that our proposed method, PATM (Physical-Aware Task Migration), outperforms 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 algorithm 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 remainder of this paper is organized as follows: The preliminaries of our algorithm are presented in Section II. Section III describes our proposed algorithm. Implementation and analysis results are shown in Section IV and final conclusions are drawn in Section V.

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 {$core_1$, $core_2$, …, $core_N$ } where each core can execute up to two threads simultaneously (two-context SMT cores). It is assumed that each of N cores has a physical thermal sensor. Since in this paper we focus on SMT feature of processors, the number of tasks should be more than the number of the physical cores, thus it is assumed there are $N+1$ to $2\times N$ tasks for execution. The problem discussed in this paper is how to schedule these tasks among cores dynamically such that the average and peak temperature of the processor can be minimized under minimum performance loss and also temperature does not violate T_{max}. We propose a heuristic method 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 introduce a new temperature prediction method, which predicts the future temperature of a core by considering both *cores physical features* and workload of the processor. 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. Physical features of cores

As mentioned earlier, the temperature of each core of a processor is different from other cores. Table I summarizes our experimental results of running different applications of SPEC2006 benchmark suite, 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 (at a fixed fan speed) while one core executes an application and others are idle. The reported temperature is the maximum temperature among all four cores (e.g. 71°C is the peak temperature among all four cores of Core i7-2600, when core 3 executes the bzip2).

According to Table I, despite all four cores of Core i7-2600 have the same experimental setup, core 3 and core 1 are always the hottest and coolest cores, respectively. We tried the same experiment with Core i7-3770 and 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 features* 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 features* to enhance the thermal management.

TABLE I
Temperature differential between cores.

Benchmark	Executed on core 0	Executed on core 1	Executed on core 2	Executed on core 3
	Intel Core i7-2600			
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 a

temperature that the system would reach if the 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 a core-specific constant. 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 to core activity. w is added to reflect the thermal effects of other cores that are active (running applications) which has not been considered in [9]. The value of c, and w are determined offline using Eq. 2 by trying thermal curves corresponding to SPEC2006 benchmarks on the cores and running different number (one to eight applications) of them, simultaneously.

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. Based on the value of t_r the proposed algorithm decides when to start task migration.

III. Proposed Algorithm

This section discusses the proposed algorithm. In the following subsections, different parts of algorithm are fully explained. The flowchart of the proposed algorithm is depicted in Fig. 1 which briefly illustrates the intuituion behind the algorithm. The main parts of the algorithm are: *Threshold Management*, *Temperature Management*, and *Performance Management*. 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*. If so, the algorithm reschedules and migrates 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.

A. Threshold Management

In all previous proposed task migration algorithms for DTM, T_{thr} is fixed [4-9]. Our experiments show that by having an adaptive T_{thr}, better results can be obtained (subsection IV.D). Therefore, 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 of workload behavior. At first $T_{thr} = T_{max}$. During execution, if the total number of migrations in the last M iterations of the algorithm is higher than $Migration_{limit}$, T_{thr} 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} decreases. The higher migration frequency is, the more overall system performance degrades. Therefore, our proposed T_{thr} management tries to control migration frequency and prevent it from increasing. 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 Management* finds a trade-off between temperature threshold and task migration frequency regarding to workload and cores thermal behavior.

B. Temperature Management

A main challenge in task scheduling of SMT multi-core in

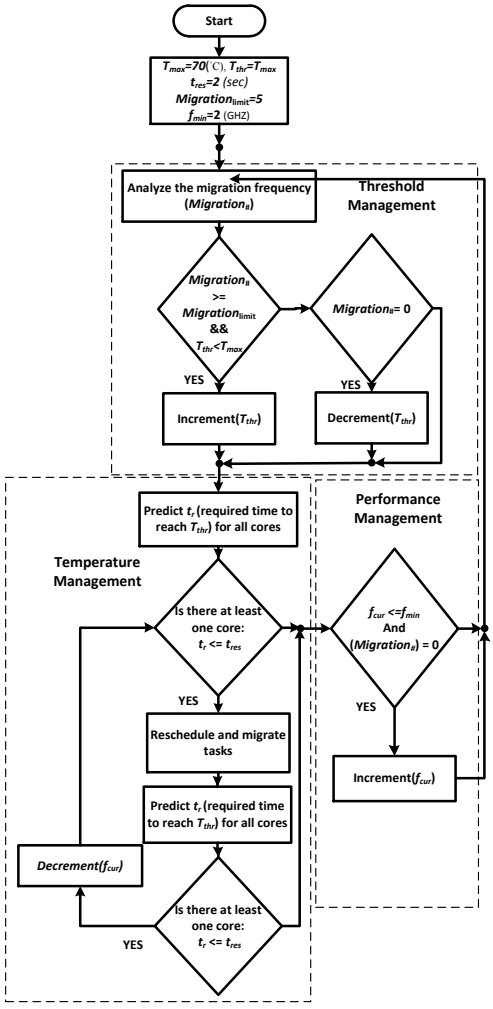


Figure 1- The flowchart of the proposed PATM algorithm.

order to improve performance, is co-scheduling of complementary threads on individual SMT cores [6] to make better use of shared pipeline resources. However, this way of scheduling causes higher heat generation due to more pipeline resources utilization [6]. 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, cores and tasks are sorted according to their temperature. Physical sensors and performance counters are used to measure the temperature of the cores and tasks, respectively. After sorting cores and tasks, hottest and 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 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 *physical features* (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 *physical features*, the first two hottest tasks are co-scheduled to the coldest core. 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. Learning physical features of cores can be done offline and it is needed to be accomplished only once. Our fifth strategy reschedules tasks between only core that is in *critical situation* (the core that has $t_r<t_{res}$) and predicted 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 tasks in hot core will be 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. 2 illustrates selected task scheduling strategy.

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

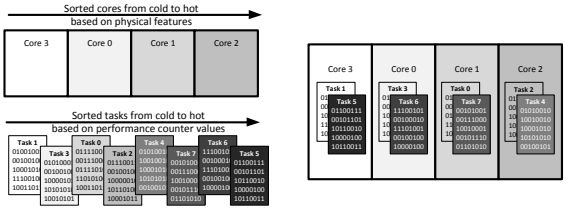


Figure 2 – Second task scheduling strategy.

C. Performance Management

As mentioned in the previous section, if *Temperature Management* cannot improve *critical situation*, it decreases the processor frequency. 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 cores. If the number of migrations is zero in the last *M* iterations and current processor frequency is lower than f_{min}, algorithm increases the global frequency to enhance performance.

IV. Experimental Results

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

A. Experimental Setup

The selected programs from SPEC2006 benchmark suite are summarized in Table II. The processor we use is an Intel Core i7-3770 while the SMT capability of processors is active. The size of the main memory of the system is 8 GB. The Linux kernel version is 3.2.0. The LM sensor [14] 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 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 $T_{thr}=T_{max}$. The other constant is the number of iterations of the algorithm *(M)* for counting $migration_{\#}$ which is set to 10. The temperature threshold that we do not want to violate (T_{max}) is 70°C.

B. Performance counter analysis

Our algorithm uses performance counters to *1)* distinguish the cold thread from the hot thread on a two-context SMT core and *2)* to sort applications according to their temperature. We need to find the performance counter which 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) [13]. It is used as a criterion to measure the correlation between two variables X and Y. The *r* coefficient is calculated using Eq. 5:

$$r = \frac{\sum_{i=1}^{N}(X_i-\overline{X})(Y_i-\overline{Y})}{\sqrt{\sum_{i=1}^{N}(X_i-\overline{X})^2}\sqrt{\sum_{i=1}^{N}(Y_i-\overline{Y})^2}} \quad (5)$$

TABLE II
SPEC CPU 2006 applications 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

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

Acording to Table III *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 III
Correlation between events and applications 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 results. Fig. 3 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 causes peak temperature and average temperature improves 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 III.B) are tried and their results are compared against Linux scheduler. In each strategy we study four cases i.e. we execute from 5 to 8 different benchmarks simultaneously. The average results of four cases are shown in Fig. 4. 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 has been thermal management, we use the second strategy for the following experiments. The results show the effectiveness of considering *physical features* in the algorithm on the results.

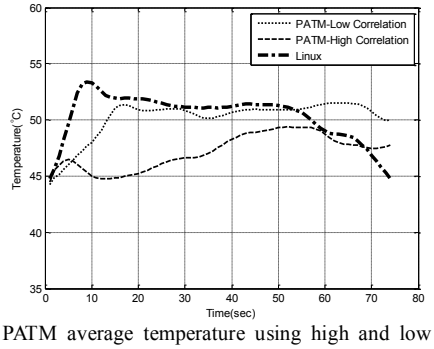


Figure 3 - PATM average temperature using high and low correlation counter for application ordering and Linux standard scheduler.

D. Adaptive threshold analysis

In evaluating the effectiveness of having 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. 5 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 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. 6 illustrates the results of the prediction modell of TAS [9] vs. ours against the real core temperature. Using our predictor, mean absolute error (MAE) is 0.6 °C whereas MAE of TAS predictor is 0.8 °C on running gcc and hmmer programs simultaneously, which shows the effectiveness of new added *w* parameter to our predictor.

F. Thermal management results

Fig. 7 illustrates average and peak temperature and

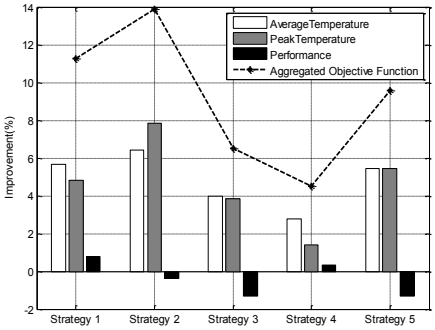


Figure 4 - performance, average, and peak temperature improvement of different strategies compared to Linux standard scheduler.

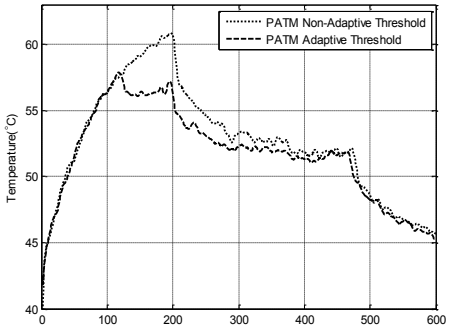


Figure 5 - Comparison of our proposed algorithms for two cases with adaptive and non-adaptive thresholds.

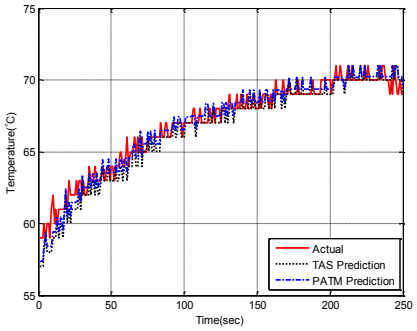


Figure 6 - The comparison of PATM and TAS predictors.

performance overhead for TAS, PDTM, Linux scheduler, and our proposed algorithm with various number of programs on an Intel core i7-3770.

After running a different set of programs on Intel Core i7-3770 processor, it is observable that our proposed technique (PATM) reduces the average temperature by about 7.7% (3.6C), and reduces peak temperature by almost 13.9% (7.8°C) with 1.7% performance (run-time) overhead compared to the standard Linux scheduler, on average. It should be noted that the average temperature is the mean of four cores temperature running programs simultaneously from beginning to the end. The experimental results also indicate that our proposed algorithm reduces the average temperature by about 1.1% (0.5°C) and 1.3% (0.6°C) compared to TAS and PDTM, respectively. Reduction in peak temperature through PATM is about 8.1% (4.5°C) and 5.8% (3.3°C) compared to both TAS and PTDM, correspondingly. The overall system performance (run-time) overhead is only 1.3% and 0.4% compared to TAS and PDTM, respectively. Table IV summarizes the comparison results for these four algorithms that are the mean values extracted at run-time for five to eight attempted applications. Hence, compared to Linux, PDTM, TAS, our proposed method indeed leads to more peak temperature reduction with negligible performance overhead.

V. Conclusion

In this paper, a dynamic thermal management algorithm with a future temperature predictor for SMT multicore processors is presented. The proposed algorithm manages processor temperature taking into account the workload and physical features of cores. As demonstrated, considering SMT, physical features of cores, applications thermal behavior, and adaptive migration threshold ability are extremely important in the DTM and they have significant influence on performance and temperature management. Experimental results obtained by SPEC CPU2006 running on a desktop platform (Intel Core i7-3770) indicate that our algorithm outperforms Linux standard scheduler, TAS, and PDTM in terms of thermal management with negligible performance overhead.

TABLE IV
Comparison results of PATM against Linux, TAS, and PDTM.

DTM Algorithm	Average Temp.	Peak Temp.	Run Time(Second)
Linux	50.9(°C)	64(°C)	912.5(Sec)
PDTM	47.9(°C)	59(°C)	924.8(Sec)
TAS	47.8(°C)	60(°C)	916.8(Sec)
PATM	47.3(°C)	56(°C)	928.3(Sec)
Improvement PATM vs. PDTM	1.3%	5.8%	-0.4%
Improvement PATM vs. TAS	1.1%	8.1%	-1.3%
Improvement PATM vs. Linux	7.7%	13.9%	-1.7%

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] 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 *Low Power Electronics and Design (ISLPED)*, pp. 379-384, 2011.

[3] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," In *Proc. of the 2007 international symposium on Low power electronics and design*, pp. 213-218, 2007.

[4] G. Liu, M. Fan, and G. Quan, "Neighbor-aware dynamic thermal management for multi-core platform," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 187-192, 2012.

[5] Z. Liu, T. Xu, S.X.-D. Tan, and H. Wang "Dynamic thermal management for multi-core microprocessors considering transient thermal effects," In *18th Asia and South Pacific Design Automation Conference*, pp. 473-478, 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 *ACM SIGARCH Computer Architecture News*, pp. 260-270, 2004.

[7] A. Kumar, L. Shang, L. Peh, and N.K. Jha, "HybDTM: a coordinated hardw5are-software approach for dynamic thermal management," In *Proc. of the 43rd Design Automation Conference*, pp. 548-553, 2006.

[8] 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.

[9] 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.

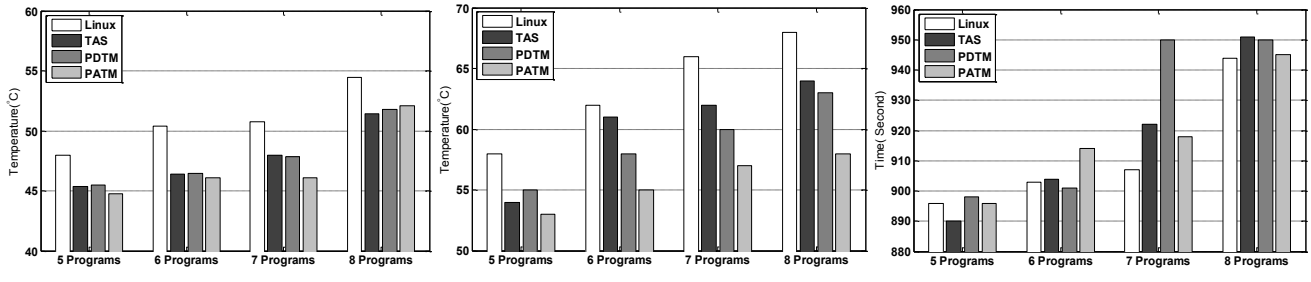
[10] Y. Li, K. Skadron, D. Brooks, and Z. Hu, "Performance, energy, and thermal considerations for SMT and CMP architectures," In *11th International Symposium on High-Performance Computer Architecture*, pp. 71-82, 2005.

[11] J. Donald, and M. Martonosi, "Temperature-aware design issues for SMT and CMP architectures," In *Proc. of the Workshop on Complexity-Effective Design. ACM Press*, 2004.

[12] 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.

[13] 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.

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



(a) (b) (c)

Figure 7 - Comparison of different algorithms with various number of programs in terms of (a) average temperature (b) peak temperature, and (c) run-time.