Neighbor-Aware Dynamic Thermal Management for Multi-core Platform

Guanglei Liu, Ming Fan, Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL 33174
gliu002@fiu.edu, fanming0626@gmail.com, gang.quan@fiu.edu

***Abstract*—With the high integration density and complexity of the modern multi-core platform, thermal problems become more and more significant for both the manufacture and system designer. Dynamic thermal management technique is one effective and efficient way to mitigate and avoid thermal emergences. In this paper, we propose a novel predictive dynamic thermal management algorithm to maximize the multi-core system throughput while satisfying the peak temperature constraints. Different from the conventional approaches, we found that it is not necessarily always a good choice to migrate a hot task to the core with the lowest temperature. Instead, in our algorithm, we develop a new temperature prediction technique and migration scheme that take the local temperature of a core as well as the impacts from neighboring cores into considerations. According to our experiment results on a practical Intel desktop platform, the proposed algorithm can significantly improve the throughput compared with the conventional approach.**

I. Introduction

Fueled by the market need for high computation capability, the size of transistors is continuously shrinking, and more and more transistors are integrated into a single chip to build up more complicated circuit architectures, i.e. chip multiprocessors (CMPs). As a result, the power density within the chip and heat generated by transistors increase rapidly in CMPs. Thus, power and thermal issues become the major challenges for the further improvement of computing performance on CMPs.

The rapidly growing heat generation greatly increases the packaging/cooling costs, and adversely affects the life-span, performance, and reliability of a computing system. The increased heat dissipation can cause thermal failures, even permanent physical damage to the processor. Therefore, an effective thermal management solution is highly desirable, not only to balance the chip's temperature but also to enable the computing system to operate at a high computing performance without exceeding its temperature limit.

The dynamic thermal management technique (DTM) is one of the most effective approaches to address the power and thermal design problem. Many theoretical works have been done by using the dynamic voltage and frequency scaling (DVFS) technique [1] [2] [3], which can control the temperature by dynamically adjusting the processor speed based on the workload. For example, Chantem et al. [4] proposed an algorithm to run real-time tasks under the temperature constraint by switching two avaliable neighboring speed of the ideal speed. However, DVFS techniques sacrifice the performance to cool down the temperature. Task migration is an alternative technique to manage the temperature by balancing the workload among CPU cores without slowing down the processing speed [5] [3] [6]. For example, Gomaa [7] et al. proposed a reactive task migration algorithm, which migrates the task away from overheated core to the coolest core. However, most theoretical thermal management algorithms are derived based on simplified models and assumptions, such as the assumption that the accurate temperatures of processors are readily available, which is not necessarily true on practical platform.

When DTM techniques are applied for real applications, they must deal with important practical details in the practical environment. To this end, many researches have been carried out based on practical hardware platforms [8] [9] [10] [11] [12] [13]. For example, Yefu [14] et al. proposed a chip-level power management algorithm by using control theory and implemented their algorithm on an Intel Xeon desktop. Ahn [8] et al. developed and validated a heuristic to reduce the power consumption and control the temperature on the Intel Centrino Duo and ARM-11 MPCore platforms. The above algorithms rely on the thermal sensor reading to trigger their DTM actions. Since the thermal sensor lacks accuracy due to their placement location and long latency, the effectiveness of the DTM techniques can be severely degraded. Even if the thermal sensor can accurately detect a thermal emergency when temperature reaches the threshold, it still takes 100 to 200 millisecond for the DTM manager to decrease the frequency or migrate the hot task to a different processor [2]. As a result, the temperature would exceed the threshold before the algorithm takes effect. To this end, predicting the potential thermal emergency before thermal failure occuring is a very important feature for the DTM algorithm [6]. In response to this, Inchoon [5] et al. proposed a temperature prediction algorithm, which takes the application's thermal behavior into consideration. Khan [3] et al. developed an alternative thermal management schedule, which combined temperature history based prediction and task

978-3-9810801-8-6/DATE12/©2012 EDAA

migration techniques to efficiently control the CPU temperature under threshold. However, they assumes that at each sampling point, the temperature will increase at the same rate until it reaches the threshold, which is not true for the practical scenario.

In this paper, we develop an on-line predictive thermal management algorithm to maximize the throughput on multi-core systems while satisfying the peak temperature constraint. Compared with the previous work, we make three major contributions in this work:

- We develop a temperature prediction method, which can predict the temperature of a core more accurately by taking its temperature as well as the neighboring impacts into consideration.
- We develop a new task migration strategy. While it has been a common approach to migrate tasks from the hottest to the coolest core, our approach chooses the destination core differently. We choose the destination core not only by its current temperature, but also by the temperature trends as well as the neighboring impacts as well.
- We validate our algorithm on a practical hardware test bed, i.e a desktop workstation with an Intel i5 750 quad core microprocessor. The experimental results show that our proposed algorithm can significantly outperform the conventional approach.

The rest of the paper is organized as follows. In Section 2, we introduce the preliminaries for this paper, and use an example to motivate our research. Section 3 introduces our proposed algorithm. Experimental results are discussed in Section 4, and the conclusion is given in Section 5.

II. Preliminary

A. Problem description

The system considered in this paper consists of N tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ and M identical processors, denoted as $\mathcal{P} = \{P_1, P_2, ..., P_M\}$. The problem discussed in this paper is how to manipulate the scheduler such that the throughput of the system can be maximized under peak temperature constraint. The formal description of the problem is represented below.

Problem Description: Given a task set Γ and a multi-core system $\mathcal{P}$, maximize the throughput of the system under the peak temperature constraint.

For processor P_i, we use a tuple (T_i, t_i) to represent the temperature of P_i at a certain time point t_i. To be more specific, we use T_i^{curr} and T_i^{prev} to denote P_i's current temperature and previous temperature respectively, while t_i^{curr} and t_i^{prev} are the corresponding times.

In this paper, we develop a heuristic 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 processor core by considering both local temperature history and neighbors' effect. Once a potential risk detected under our temperature prediction model, i.e. the predicted temperature is over the threshold, we dynamically manage the executions of corresponding tasks on that core by either migration or DVFS. By considering the neighbors' temperature and their changing trends, we can select a processor among all available candidates to improve the total system performance from a global and long-term perspective.

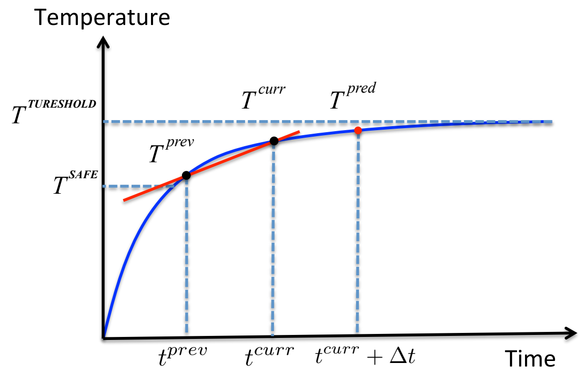


Fig. 1. Temperature history based prediction

III. Algorithm

In this section, we first introduce our temperature prediction method. Then we discuss our approach to select the candidate core to migrate tasks from the core that is having a thermal emergency situation. Finally, we present our overall algorithm.

A. Temperature Prediction

In this subsection, we introduce our approach that can predict the future temperature of a core as well as its future trend more accurately. First, we introduce the following definition to represent the future temperature increment of each processor individually.

Definition 1: Given processor P_i, the *individual increment* factor of P_i, denoted as I_i^{in}, is defined as

$$I_i^{in} = T_i^{curr} - T_i^{prev} \tag{1}$$

This local temperature increment will be used to predict the future temperature at the next sampling point, as shown in Figure 1. (Δt is the sampling period). For a typical desktop computer, the sampling period has been set to 1 second, since this is approximate how long it takes for the thermal sensor to reflect a temperature change [15].

Besides its own power consumption, the temperature of a processor is also affected by other processors on the same chip, especially its neighbors. In this paper, we define the *neighbor processors* of a processor P_i, denoted as $\mathcal{P}_i^{NB}$, as the cores which are adjacent to P_i. When predicting the temperature of a processor, to simplify our algorithm, we consider only the heat transfer impacts from its neighboring processors. By considering the effect of neighbor processors, we define the following concept to represent the neighbors' thermal increment for a given processor.

Definition 2: Given a processor P_i, the *neighbor increment* factor of P_i, denoted as I_i^{nb}, is defined as

$$I_i^{nb} = \frac{\sum_{P_j \in \mathcal{P}_i^{NB}} (T_j^{curr} - T_j^{prev})}{|\mathcal{P}_i^{NB}|} \tag{2}$$

With Definition 1 and 2, we are now ready to introduce our temperature prediction method. Let T_i^{pred} denote the predicted temperature for P_i. We formulate T_i^{pred} as a linear function of its current temperature T_i^{curr}, its temperature increment rate I_i^{in}, and also its neighbor increment factor I_i^{nb}, as shown below:

$$T_i^{pred} = \alpha_i \cdot T_i^{curr} + \beta_i \cdot I_i^{in} + \gamma_i \cdot I_i^{nb} \tag{3}$$

where α_i, β_i and γ_i are weight parameters for P_i.

To determine the values of weight parameters, we resort to the least-square estimation method [5]. Let

$$T^{pred}(t) = \alpha \cdot T^{curr}(t) + \beta \cdot I^{in}(t) + \gamma \cdot I^{nb}(t)$$

where $t = [t_1, t_2, ..., t_n]$ are different sampling points, $T^{curr}(t)$, $I^{in}(t)$ and $I^{nb}(t)$ are corresponding values at t. By collecting a set of training data, $(t_j; T_j^{pred})$ where $j = 1, ..., m$, we have

$$\hat{T}^{pred} = \hat{T} \ \times \hat{W}$$

where $\hat{T}$ is a $3 \times m$ matrix:

$$\hat{T} = \begin{bmatrix} T^{curr}(t_1) & I^{in}(t_1) & I^{nb}(t_1) \\ \vdots & \ddots & \vdots \\ T^{curr}(t_m) & I^{in}(t_m) & I^{nb}(t_m) \end{bmatrix} \tag{4}$$

$\hat{W}$ is a 3×1 unknown weight parameter vector:

$$\hat{W} = [\alpha, \ \beta, \ \gamma]^T \tag{5}$$

and $\hat{T}^{pred}$ is a $m \times 1$ output vector:

$$\hat{T}^{pred} = [T_1^{pred}, \ T_2^{pred}, ..., T_m^{pred}]^T \tag{6}$$

If $(\hat{T}^{pred})^T \hat{T}^{pred}$ is nonsingular, we have

$$\hat{W} = ((\hat{T}^{pred})^T \hat{T}^{pred})^{-1} (\hat{T}^{pred})^T \hat{T}^{pred} \tag{7}$$

Based on the weight parameters calculated from the above equation, we can predict the future temperature of a processor by applying equation (3).

B. Candidate processor for migration

When a processor is in temperature emergency, one solution is to migrate the task to other processors that are cooler. To identify the appropriate destination processor, one common approach such as [7] is to migrate tasks to the processor with the lowest current temperature. In our approach, besides the current temperature of the candidate processor, we also take considerations of its neighboring temperatures, as well as its temperature changing rate.

We first introduce a concept, *heat index*, to quantify how hot a candidate processor (i.e. P_k) is.

Definition 3: Given processor P_k, the *heat index* of P_k, denoted as $\mathcal{H}(P_k)$, is defined as

$$\mathcal{H}(P_k) = \frac{\sum_{P_j \in \mathcal{P}_k^{NB} \bigcup \{P_k\}} T_j}{|\mathcal{P}_k^{NB} \bigcup \{P_k\}|} \tag{8}$$

Intuitively, the smaller the heat index of a processor is, the better candidate processor it can be.

Besides the heat index of a processor, we also consider the temperature changing rates of itself as well as its neighbors. We present the following definition, i.e. the *heat index increasing factor* of a processor P_k, to capture this concept.

Definition 4: Given processor P_k, the *heat index increasing factor* of P_k, denoted as $I(P_k)$, is defined as

$$I(P_k) = \frac{\sum_{P_j \in \mathcal{P}_k^{NB} \bigcup \{P_k\}} \frac{T_j^{curr} - T_j^{prev}}{t_j^{curr} - t_j^{prev}}}{|\mathcal{P}_k^{NB} \bigcup \{P_k\}|} \tag{9}$$

According to Definition 4, $I(P_k)$ indicates how fast the temperature at P_k and its neighbors can increase in average. Thus, the smaller heat index increasing factor, the better that candidate processor can be. From equation (8) and (9), we choose the migration candidate as the one that minimizes

$$\mathcal{H}(P_k) + I(P_k) \cdot \Delta t \tag{10}$$

where Δt is the length of the sampling interval.

Note that, task migration is not always effective in dealing with thermal emergency, especially when the workload is heavy. Given a processor P_k in thermal emergency, it does not help much if the selected target processor (e.g. P_k) for migration has a temperature very close to the peak temperature limit, even if the $\mathcal{H}(P_k) + I(P_k) \cdot \Delta t$ is minimum among all other processors. It may even degrade the throughput performance to frequently migrate tasks among different processors. To avoid this scenario, in our approach, the tasks on processor P_k are only allowed to migrate to processor P_k if

$$\mathcal{H}(P_k) + I(P_k) \cdot \Delta t \leq T^{THRESHOLD} \tag{11}$$

where $T^{THRESHOLD}$ is the given temperature constraint. Otherwise, we will have to adopt an alternative solution, i.e. by reducing the performance of P_k using techniques such as DVFS, to deal with the thermal emergency.

C. Thermal Management Algorithm

In this subsection, we introduce our proposed thermal management algorithm, the Neighbor-Aware Dynamic Thermal Management (NADTM) algorithm, to maximize the throughput of a multi-core system while keeping the temperature under a predefined peak temperature limit.

Algorithm NADTM is presented in Algorithm 1. For processor P_i, we read its temperature sensor to get its current temperature reading and then predict its temperature at the next sampling point based on the method described in section III-A. If the predicted temperature exceeds the temperature constraint, we then search for a candidate processor that we can migrate the tasks to. The candidate processor are selected based on method presented in section III-B. If such a processor

Algorithm 1 Neighbor-Aware Dynamic Thermal Management (NADTM) Algorithm

```
1:  $T_i^{prev} := T_i^{curr}$ // the temperature at previous sampling point ;
2:  $T_i^{curr} :=$ the temperature of $P_i$ from temperature sensor;
3:  $T_i^{pred} :=$ predicted temperature of $P_i$ at next sampling point based on equation (3);
4:  if $T_i^{pred} > T^{THRESHOLD}$ then
5:      $P_k :=$ the processor from $\mathcal{P}$ such that $\mathcal{H}(P_k) + I(P_k) \cdot \Delta t$ is minimum;
6:      if $\mathcal{H}(P_k) + I(P_k) \cdot \Delta t \le T^{THRESHOLD}$ then
7:          migrate current running tasks on $P_i$ to $P_k$;
8:      else
9:          degrade the performance of $P_i$ by setting its speed to the pre-defined safe speed (i.e $S_i^{SAFE}$);
10:     end if
11: end if
```

is not available, we then degrade the performance of the processor to a safe speed to avoid the thermal constraint violation.

We assume that the weights in equation (3) have been identified off-line. The safe speed to run a processor is essentially the maximum processor speed for a processor such that its peak temperature will not exceed the temperature constraint. We assume that this speed is also obtained off line. More details on how to obtain the safe speed in our approach is introduced in the next section.

IV. Experiments and Results

In this section, we first introduce the experiment setup. Then we validate the accuracy of our temperature prediction technique by comparing it with the conventional prediction approach. At last, we verify the performance improvement of our thermal management algorithm by analyzing the efficiency of the neighbor-aware temperature prediction and migration, respectively.

A. *Experiment setup*

Our hardware platform is based on a Dell Precision T1500 Desktop workstation with an Intel i5 750 quad core microprocessor, with a running Linux operating system with kernel version of 2.6.23. The processor has integrated with *Enhanced Intel SpeedStep Technology* (EIST) [16] and supports 12 different working frequency levels. We adopted the *CPUfreq* Linux kernel subsystem to implement the DVFS features.

All experiments were carried out with the same ambient temperature. In addition, the fan speed has been fixed to ensure all experiments were conducted under the same cooling operations. We selected six benchmarks *galgel, parser, ammp, crafty, lucas* and *equake* from the well-known commercial benchmark SPEC CPU2000, including both integer and floating point operation to get credible and comparable experiment results. Those benchmarks have been grouped into three categories, which are *hot*, *warm*, and *cool*, based on their thermal characteristics.

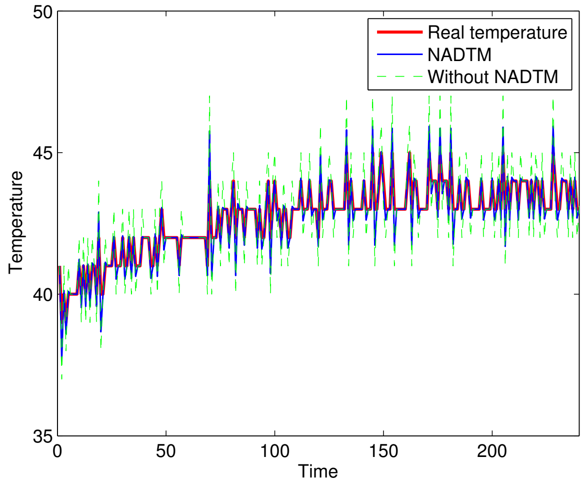


Fig. 2. NADTM compares with the conventional prediction method, which does not take the neighbor effect into consideration

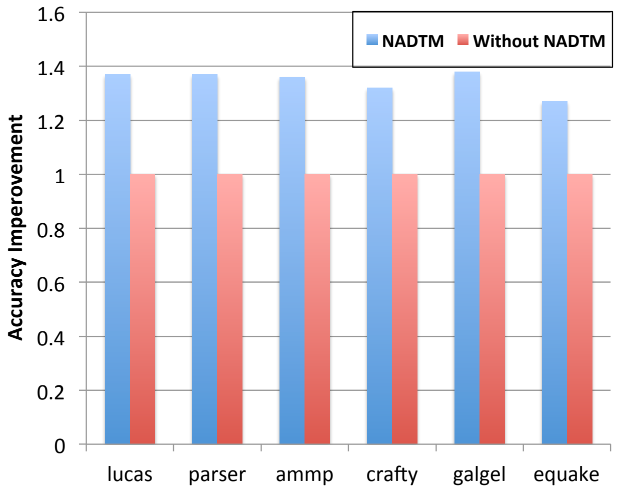


Fig. 3. Prediction accuracy comparison with different benchmarks

To determine the safe speed for a processor, we conducted the off-line thermal profiling analysis by running each benchmark with different CPU speeds. The stable temperature with its corresponding speed level were stored in a lookup table. To ensure the schedule effectiveness, each benchmark was tested with the *hot* benchmark applications running on its neighboring processors. With the lookup table, the safe speed is the maximal speed corresponding to the stable temperature lower than the given temperature constraint.

B. *Temperature prediction analysis*

To evaluate the accuracy of our NADTM temperature prediction technique, we compared our heuristic with the conventional temperature prediction approach, which simply uses the previous and current temperature values of a processor

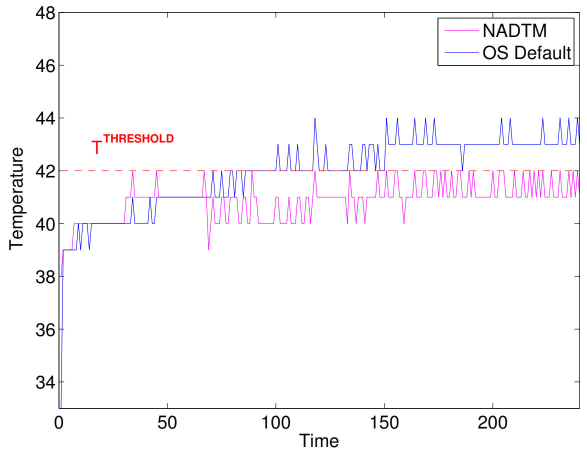


Fig. 4. Temperature trace with different DTM algorithms

to predict the next temperature value without considering the heat transfer from the neighboring processors.

Figure 2 shows the temperature traces when running benchmark *galgel*, as well as the temperature prediction results based on our proposed temperature prediction method and the conventional one. From Figure 2, we can clearly see that the temperature prediction results of using the NADTM approach is much closer to the actual temperature value than the conventional approach. The experiment result shows that the NADTM approach has a smaller maximum prediction error of 1^oC comparing with 3^oC by the conventional approach. The results shown in Figure 2 demonstrate that, by taking consideration of the heat transfer impacts from the neighboring processors, the temperature prediction methods introduced in section III-A can achieve a higher accuracy than the traditional method.

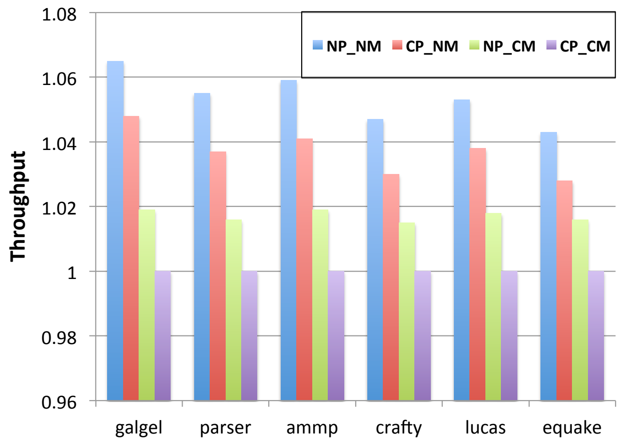
To further validate this conclusion, we ran different benchmark programs on our test platform. First, temperature prediction results will be collected and compare with the actual temperature value. Then the temperature prediction accuracy by using two different prediction methods have been plotted in Figure 3. i.e the prediction accuracy is the number of accurate prediction over total number of prediction. In order to compare the two approaches, both results are normalized to the approach without NADTM. From Figure 3 we can see that our NADTM approach can improve the temperature prediction accuracy by 38% in average compared with the conventional approach. The experiment results prove that our neighboring aware temperature prediction method could effectively improve the prediction accuracy.

C. Performance analysis

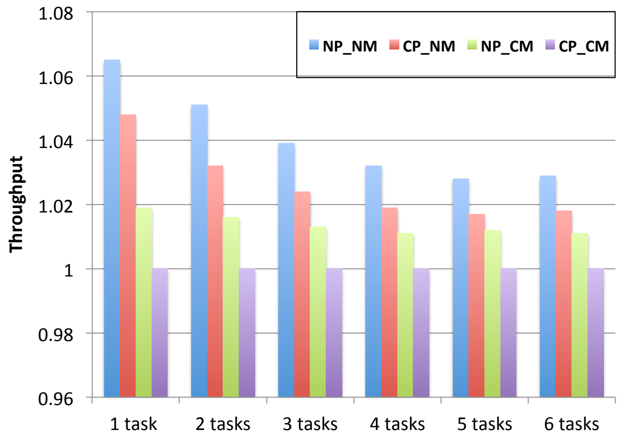
We further study the performance of our proposed algorithm in term of its capability to satisfy the temperature constraints and its throughput under the temperature constraints.

First, we analyze how effective that our NADTM algorithm can maintain the temperature under the threshold. Ideally, a thermal management schedule should push the computing system to the thermal boundary while delivering highest system performance. At the same time, the temperature need to be carefully maintained under the threshold. We test the efficiency of our NADTM algorithm by running a benchmark *galgel*. The temperature traces after implementing the NADTM and original Linux operating system schedule have been plotted in Figure 4. By default, the original operating system scheduler can lead to the temperature to 44^oC. We assume 42^oC is the predefined threshold. From Figure 4 we can see that the NADTM thermal management algorithm can effectively maintain the temperature under the threshold, Meanwhile, it has a small temperature oscillation of 1^oC at the temperature stable state. Since the processor temperature can be maintained at a level that is very close to the temperature limit, our approach can lead to a high throughput.

Next, we study the effectiveness of our new prediction and migration schemes on the throughput improvement. We use *NP, CP* to denote neighbor-aware prediction and conventional prediction, and *NM, CM* for neighbor-aware migration



(a) Execution time comparison by using different single task



(b) Execution time comparison with multiple tasks running on the multiprocessor platform

Fig. 5. Execution time comparison with four different apporaches. *NP and CP represent the neighbor-aware and conventional prediction respectively. NM and CM represent neighbor-aware and conventional migration respectively*

and conventional migration, respectively. The conventional temperature prediction approach refers to the one that predicts the future temperature solely based on its own temperature history. And the conventional migration approach refers to the approach that simply migrates the running tasks from the hottest core to the coolest core. As a result, we have four combinations, *i.e. CP_CM NP_CM, CP_NM* and *NP_NM*.

We first compare the throughput of each approach when running a single task on our hardware platform. In this experiment, six previously used benchmarks have been selected to provide reliable experiment results. The execution times by using different approaches have been recorded for comparison, those experiment results have been normalize and plotted in Figure 5(a). The experiment results show that, with the neighbor-aware prediction algorithm *i.e. NP_CM* can improve the throughput over *CP_CM* as much as 1.7% in average. Since our prediction technique is more accurate than the conventional approach as shown before, it helps to make better scheduling decision and thus improve the performance. Another interesting result is that *CP_NM* improves the throughput over *CP_CM* as much as 3.6%. This is because *CP_NM* can find the appropriate migration candidate rather than simply locate the coolest core. By combining our proposed prediction and task migration algorithm together, *NP_NM* can achieve an average of 5.8% overall throughput improvement when compared with *CP_CM*

To further test our thermal management algorithm, we assigned multiple tasks to the multicore platform. By gradually increasing the number of task running on the multicore processor, their corresponding execution times have been recorded for comparison. The execution times have been normalized and plotted in Figure 5(b). As we can see from the experiment result, the overall throughput decreases as the number of tasks increases. Another important observation is that when the number of tasks is more than the number of core (*i.e. the number of task is more than 4*), the throughput drops significantly. The experiment results show that the throughput for the *NP_CM* decreased by 0.9% while the tasks increased from 1 to 6. The throughput for *CP_NM* decreased by 3%. The throughput decreased by 3.6% for the overall NADTM algorithm. All these results show that the proposed algorithm works better when for a lighter workload than a heavy workload.

V. Conclusion

In this paper, we present a neighbor-aware dynamic thermal management algorithm to maximize the system throughput under peak temperature constraint. Our solution is important because the neighbor effect for temperature prediction on CMPs can be significant, and the processor with the lowest temperature may not be the best choice for migration. Our proposed approach takes the neighbor effect into consideration to make a more accurate temperature prediction and to determine a better migration destination. Experimental results based on practical benchmark and practical desktop platform confirm our conclusions and demonstrate the effectiveness of our approaches. All these results show that the proposed algorithm works better when for a lighter workload than a heavy workload.

Acknowledgment

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, and CNS-1018108.

References

[1] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, 2003, pp. 2 – 13.

[2] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, feb. 2008, pp. 123 –134.

[3] O. Khan and S. Kundu, "Hardware/software co-design architecture for thermal management of chip multiprocessors," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 952 –957.

[4] T. Chantem, X. S. Hu, and R. P. Dick, "Online work maximization under a peak temperature constraint," in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '09. New York, NY, USA: ACM, 2009, pp. 105–110. [Online]. Available: http://doi.acm.org/10.1145/1594233.1594257

[5] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, june 2008, pp. 734 –739.

[6] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, june 2010, pp. 579 –584.

[7] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-run: leveraging smt and cmp to manage power density through the operating system," *SIGOPS Oper. Syst. Rev.*, vol. 38, pp. 260–270, October 2004. [Online]. Available: http://doi.acm.org/10.1145/1037949.1024424

[8] Y. Ahn, Y.-S. Hwang, and K.-S. Chung, "Flexible framework for dynamic management of multi-core systems," in *SoC Design Conference (ISOCC), 2009 International*, nov. 2009, pp. 237 –240.

[9] C. Tianzhou, H. Jiangwei, X. Lingxiang, and S. Qingsong, "Dynamic power management framework for multi-core portable embedded system," in *Proceedings of the 1st international forum on Next-generation multicore/manycore technologies*, ser. IFMT '08. New York, NY, USA: ACM, 2008, pp. 1:1–1:4. [Online]. Available: http://doi.acm.org/10.1145/1463768.1463770

[10] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer, "Dimetrodon: Processor-level preventive thermal management via idle cycle injection," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, june 2011, pp. 89 –94.

[11] L. Guanglei, Q. Gang, and Q. Meikang, "Throughput maximization on intel desktop platform under the maximum temperature constraint," in *The 2011 IEEE/ACM International Conference on Green Computing and Communications*, August. 2011.

[12] T. Wei, X. Chen, and P. Mishra, "Designing a multi-core hard real-time test bed for energy measurement experiments," in *Proceedings of the 2009 ACM symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 1998–1999. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529727

[13] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 93–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956567

[14] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 314–324. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555794

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

[16] Enhanced intel speedstep technology [available online]: http://www.intel.com/support/processors/sb/cs-028855.htm.