2011 IEEE/ACM International Conference on Green Computing and Communications

Throughput Maximization for Intel Desktop Platform under the Maximum Temperature Constraint

Guanglei Liu[1], Gang Quan[1], Meikang Qiu[2]
[1]Electrical and Computer Engineering Department, Florida International University, Miami, FL 33174
[2]Department of ECE, University of Kentucky,Lexington, KY 40506

***Abstract*—As processor power consumption continues to grow, thermal issue is becoming critical in the design of computing systems. While there have been extensive theoretical researches conducted on thermal aware computing, most of these researches are based on idealized theoretical models and, sometimes, unrealistic assumptions. In this paper, we develop a practical thermal aware scheduling algorithm to optimize its throughput under a given temperature constraint. We identify limitations of related theoretical work and implement a practical algorithm on an Intel i5 desktop computing running Linux 2.6.34. We also use SPEC2000 benchmark to validate the effectiveness of the proposed algorithm.**

I. Introduction

Fueled by human's increasing appetite for high computing performance, semiconductor technology has now entered into the deep sub-micron era. The more and more complicated circuit architecture together with the continuously shrinking transistors have resulted in an exponentially increased power density in IC chips. The rapid increase of power consumption makes it challenging to provide enough power supply for computing systems, especially the computing systems with limited power sources such as the portable computing devices. Moreover, it also generates a large amount of heat to the extent that makes the thermal management a major design issue in computing system design today.

The rapid increase of heat generation can greatly increase the packaging/cooling cost, and adversely affect the life span, performance, and reliability of a computing system. It is estimated that [13] the thermal packaging increases total package cost at 1-3 dollar per watt. Moreover, according to Yeh and Chu [21], every 10^oC increase in operating temperature can cut a device's life span by half. From Santarini et al. [12], every 15^oC increase in temperature can lead to as mch as 10-15% in circuit delay. In addition, high temperature also increases leakage power consumption, which is becoming more and more prominent in total power consumption. Based on the research proposed by Liao et al. [10], increasing temperature from 65^oC to 110^oC can increase leakage power by 38% for IC circuits using the 65nm technology. Evidently, high temperature is becoming a more and more critical issue in the design of computing systems.

Thermal aware scheduling, as one of the most effective dynamic thermal management techniques, has been researched extensively in recent years. Some researches (e.g. [2], [19], [7]) take the temperature and leakage interdependency into consideration to optimize the total energy consumption. Some other approaches (e.g. [4], [9]) seek to minimize the peak runtime temperature. There are also some other researches that studied the thermal aware performance maximization problem (e.g. [3], [23], [24]). Many modern processors today have thermal aware self-protect mechanism, which automatically shuts down a processor to avoid physical damage [11] when its temperature exceeds certain threshold. As a result, the processor temperature needs to be carefully monitored and managed to avoid sudden performance disruption. To this end, Zhang and Chatha [23] presented a pseudo-polynomial time speed assigning algorithm based on dynamic programming to minimize the total execution latency. They further developed several heuristics [24] to maximize the throughput of a real-time system by sequencing the execution of a task set consisting of tasks with different power/thermal characteristics for processors with and without *dynamic voltage/frequency scaling* (DVFS) capabilities. Chantem et al. [3] proposed to run real-time tasks by frequently switching between the two speeds which are neighboring to the constant speed whose stable temperature is the given peak temperature limit.

Most theoretical thermal aware scheduling results are derived based on simplified models and assumptions. For example, most of the above research employ the lumped first order RC thermal model to model the temperature dynamics. Also, as shown later in this paper, some of the assumption such as the exact knowledge of the actual temperature may not always be possible. While theoretical researches simplify the research problem and help to uncover the fundamental principles in practical scenarios, practical applications must deal with some important details in the practical environment. To this end, a number of researches were conducted based on more practical computing systems (e.g. [18], [15], [1], [20], [11], [8]). For example, Ahn [1] et al. developed and validated a heuristic to reduce the power consumption and control the temperature on Intel Centrino Duo and ARM-11 MPCore platformss. Erven [11] et al. proposed an algorithm to ensure the system temperature under a pre-defined threshold by adjusting the utilization of the CPU in a Pentium-II desktop. Wang [20] et al. developed a feedback control based on-line temperature control method and implemented their novel algorithm on an Intel Xeon desktop. Amit [8] et al. incorporated hardware and software thermal management technologies and proposed a

978-0-7695-4466-3/11 $26.00 © 2011 IEEE
DOI 10.1109/GreenCom.2011.10

hybrid thermal management algorithm to optimize the heat dissipation in a Pentium-4 system.

In this paper, we study how to maximize the workload on a general desktop computer under certain temperature constraint. Specifically, we are interested in studying the applicability of existing theoretical results for the workload maximization. In this paper, we identify several limitations of existing theoretical algorithms on the practical desktop platform and propose a new *Dynamic Voltage Frequency Scaling* (DVFS) algorithm based on an Intel i5 computing system. We also use SPEC2000 benchmark to validate the effectiveness of the proposed algorithm.

The rest of the paper is organized as follows. Section II introduces the practical environment based on which we develop our algorithm and discusses several related theoretical approaches and their limitations. We then present our approach in Section III. We discuss the experiment results in Section IV and conclude this paper in Section V.

II. Preliminary

In this section, we first introduce the practical desktop environment that we target. We then discuss the related theoretical work on workload maximization and their limitations.

A. The practical working environment

An overview of the practical desktop environment we consider is depicted in Figure 1. The target platform is a Dell Precision T1500 desktop workstation with Intel i5 750 quad core microprocessor. The Intel i5 microprocessor supports 12 different working frequency levels ranging from 2.66GHz to 1.2GHz, as shown in Table I. The processor has integrated with *Enhanced Intel SpeedStep Technology*(EIST) [14] that enables the adjustment of frequency or voltage either separately or simultaneously. We installed the Ubuntu 10.04.1 Linux operating system on top of the hardware platform. To implement the DVFS features, we adopted the *CPUfreq* Linux kernel subsystem, which provides an interface between the user level frequency-controlling policy and the underlying mechanisms. Moreover, CPUfreq subsystem provides different governors which enable us to assign different frequencies in TableTable I to each individual core.

In our tests, we always set the voltage to the lowest level that can support the designated working frequency. In addition, our research in this paper focuses on thermal aware scheduling for single processor systems. Since Intel I5 is a quad core processor, we manually generated four copies of the same program and allocated each copy to each individual core, and the frequency and voltage were always set to the same level at the same time.

A Dell Precision T1500 desktop workstation has two cooling components: the heat sink and cooling fans. The cooling effect of the heat sink depends on its physical characteristic, which does not change when running an application. The fans, on the other hand, can be adjusted dynamically between the maximal of 4500RPM (round per minute) and minimal of 1500RPM. To simplify our test cases, we fixed the fan speed at 1500RPM.

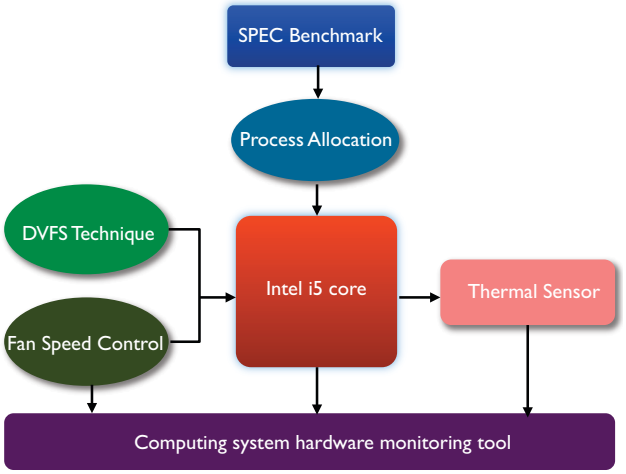


Fig. 1. Structure of hardware platform

TABLE I
Intel i5 available frequency levels (GHz)

1	2	3	4	5	6
2.667	2.533	2.4	2.267	2.133	2.0
7	8	9	10	11	12
1.867	1.733	1.6	1.467	1.333	1.2

A key aspect in our study is to capture the temperature information accurately and timely. Dell Precision T1500 desktop has an external thermal sensor, located underneath the CPU chip. An alternative method is to read temperature value directly from the built-in digital thermal sensor integrated with each core. The temperature is stored in the *Model Specific Register* (MSR), which can be accessed through the *Industry Standard Architecture* (ISA) ports or the *System Management Bus* (SMBus). In our tests, we used the average temperature values of all four on-chip thermal sensors to get the instant temperature of the processor chip.

To ease our implementation and tests, we simply adopted a Linux hardware monitoring tool called *Lm-sensors* to capture the temperature, to set the fan speed, to vary supply voltage and working frequency. For temperature, it can report not only the chip temperature but also the ambient temperature. In our experiments, we found that the time to access the on-chip sensor takes approximately $8ns$; the resolution of the on-chip thermal sensor is only 1^oC; and the minimal time for a temperature sensor to reflect a change in temperature is approximately 1 second [6].

B. Workload optimization under the peak temperature constraint

As discussed in the introduction section, there have been many approaches published to address the thermal aware performance maximization problem, such as [3], [23], [24]. However, most of these approaches require detailed knowledge of processes running on the platform, such as the exact numbers of processes and their execution times, which is not always available on a general computing platform such as the

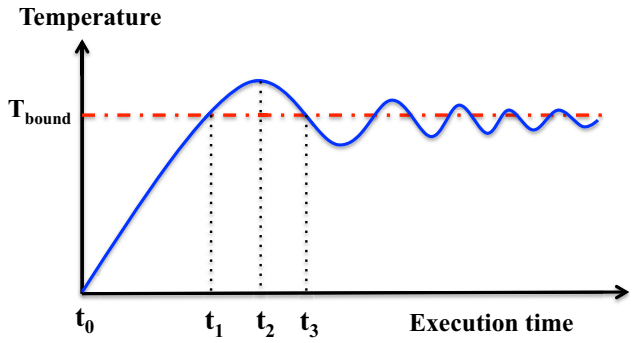


Fig. 2. An example of temperature trace.

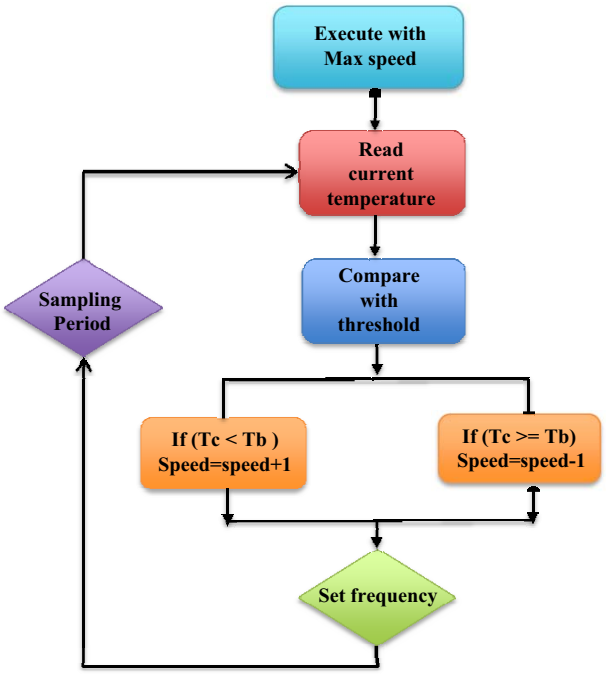


Fig. 3. A simple dynamic approach for throughput maximization.

desktop. Some other information such as the thermal resistance and thermal capacitance, which is essential to build the thermal model for the processor, is also not immediately available.

In addition, some common assumptions used in these approaches may not be applicable. For example, it has been a common practice to assume (e.g. [23]) that power consumption of a processor remains constant as long as it is executed using a constant speed. In reality, power consumption varies with not only temperature but also applications. Also, it is not surprising that a process may consume different power consumptions at different execution stages.

To develop a general thermal aware scheduling algorithm for the desktop computing platform, the less information is required regarding the processes and platform, the more effective can the algorithm be. In this regard, the *reactive two speed* scheduling approach, proposed by Wang et al [16], seems to be a good choice. According to this algorithm, the processor runs at the maximum speed before it reaches the temperature limit and then runs at a speed, so called *the equilibrium speed*, to maintain this temperature. It has been proved [5], theoretically, this is the optimal approach to maximize the workload under a peak temperature constraint.

However, there are a number of problems with this approach. First, since most processors support only discrete levels of working frequencies. The ideal equilibrium speed may not always be available for a given peak temperature constraint. Furthermore, the power consumption the processor varies with not only the processor speeds, but also other factors such as the types of processes, operating temperature, etc. In fact, even a single processor running a single process may have different power consumptions at different times. Therefore, the *the equilibrium speed* is not unique and constant at all, and it is simply not possible to simply set a processor to a constant speed once and for all to maintain a constant temperature.

To deal with these problems, another approach [11], as shown in Figure 3, seems to be more flexible. This approach assumes no a priori knowledge of the applications running on the computing system at all. It monitors the chip temperature regularly and adjusts the processor performance dynamically. At each temperature sampling point, if the current temperature does not reach the threshold, the processor speed is elevated to one level higher. Otherwise, if the current temperature equals or exceeds the temperature limit, the processor speed is changed to one level lower to cool down the temperature.

At the first sight, it seems that this approach solves all the problems mentioned above. It naturally assumes the processor has a discrete working frequency levels. It does not assume any a priori knowledge of the programs running on the processor either. However, there are still a few problems that make this approach less effective in a practical desktop environment. First, this approach assumes that the instant temperature information is available immediately and accurately. Second, updating the frequency level one at a time may not be quick enough to respond to temperature change and meet the temperature constraint.

We use a simple example to explain these two problems. Consider Figure 2. Recall that it takes about 1 second for the thermal sensor in our desktop to reflect any temperature changes. It is possible that even though the system temperature has already reached or surpassed the temperature threshold at t_1, the sensor reading may still be lower than temperature limit. The algorithm continues to increase the performance level of the processor and thus violates the peak temperature constraint. Moreover, even though the algorithm can sense the accurate temperature at t_1 and find that it has already reached the temperature threshold, since it adjusts frequency level one at a time, it may not be able to reduce the temperature fast enough. The temperature continues to rise and violates the peak temperature constraint. To address these problems, in what follows, we develop a new algorithm that can maintain processor temperature under limit while maximizing the system performance.

III. Our approach

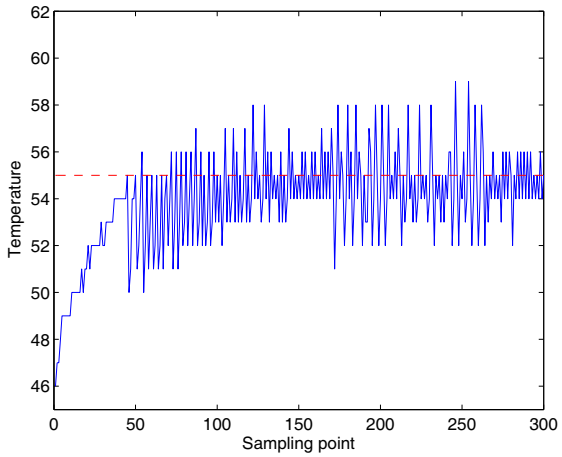
Being able to monitor the temperature change timely and accurately is one of the most critical issues for our approach. Theoretically, it is possible to use interrupt mechanism to monitor thermal sensor readings, and it will be our future work to study the effectiveness and efficiency of using interrupt for this purpose. In this work, as that in [11], we use the simple polling method to monitor thermal sensors for the temperature variations. As a result, defining the appropriate sampling period becomes critical.

A. Identify the sampling period

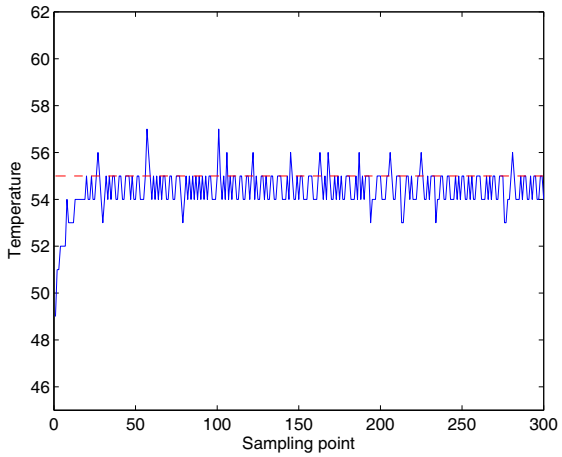
One intuitive idea to define the sampling period is to set the period as small as possible in order to track the temperature change quickly, provided the accumulated overhead can still be negligible. Unfortunately, using a very small sampling period can in fact increase the possibility of temperature constraint violation. Figure 4(a) shows a temperature trace when the sampling period was set to a very small value, i.e. $8ns$. From this figure, we can see that processor temperature violates the given temperature constraint frequently and can exceed the temperature threshold as much as 4^oC. This is mainly due to two factors: First, the thermal sensor cannot keep up with the temperature changes instantly; Second, as mentioned before, the thermal sensor resolution is only 1^oC. Simply relying on the thermal sensor reading can actually mislead the performance setting of the processor. On the other hand, setting the sampling period too large cannot catch the temperature variations timely.

Given the limitations of the temperature sensor in our platform, in our approach, we set the sampling period to be equal to the minimal response time of the thermal sensor for temperature change. To identify the minimal temperature response time, we ran different benchmarks at different speeds with different sample periods. The minimal interval within which the temperature sensor has the same reading is set to be the sampling period. From our empirical work, we found the minimal temperature response time to be 0.98 seconds.

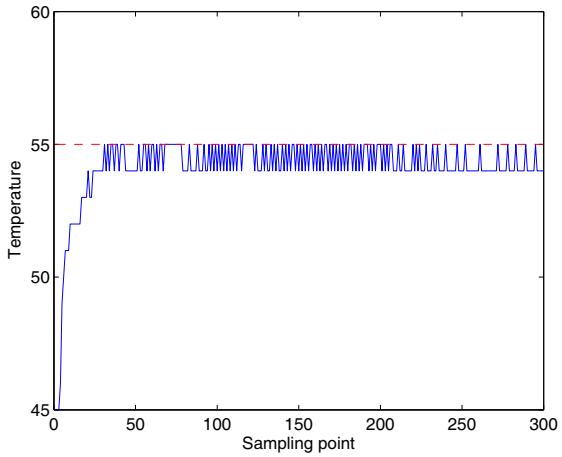
The sampling period defined above can be more effective in avoiding the misjudgement of the temperature changing trend. However, in the worst case scenario, it takes as long as 0.98 seconds to find out that the thermal sensor readings has changed. For example, the thermal reading changes exactly after one sampling point. To further improve the performance, we develop a polling algorithm (as described by Algorithm 1) that uses non-constant sampling periods. Algorithm 1 adopts two sampling speeds T_r and T_s with ($T_r >> T_s$). When the thermal sensor changes its readings, a new processor speed is set accordingly and the sampling period is set to T_r. If the temperature sensor remains the same value, Algorithm 1 changes the sampling period to T_s and the processor performance remains the same. In our case, $T_r = 0.98$ seconds and $T_s = 0.1$ seconds. In comparison with the algorithm using a constant sampling period, this approach catches temperature change and responds to it more timely. Therefore this approach can achieve better performance as shown in Figure 4(b).



(a)



(b)



(c)

Fig. 4. (a) The temperature trace with very small sampling period. (b)The temperature trace with sampling period from Algorithm 1.(c)The temperature trace with sampling period from Algorithm 2.

Algorithm 1 THE POLLING METHOD WITH VARIABLE POLLING PERIODS

```
1: while Process is running do
2:    Read current temperature T_cur;
3:    if T_cur = T_previous then
4:       Wait T_s = 0.1 seconds;
5:    else
6:       if T_cur < T_bound then
7:          Increase processor speed by one level;
8:       else
9:          Decrease processor speed by one level;
10:      end if
11:      Wait T_r = 0.98 second;
12:   end if
13: end while
```

B. Buffer zone and temperature speed lookup table

Figure 4(b) shows clearly that Algorithm 1 greatly reduces the number of "spikes" (i.e. temperature violations) over the simple polling method with a constant sampling period. However, as we can see from Figure 4(b), there are still temperature violations occur. This is due to two reasons: First, even though Algorithm 1 greatly reduces the probability of temperature violations, in the worst case scenario, it still takes as long as 0.98 seconds to confirm a temperature change. This makes temperature violations inevitable for Algorithm 1. Second, Algorithm 1 adjusts the processor speed one level at a time. When temperature is really close to its limit, it is simply not fast enough to slow down the processor speed to cool down the temperature in time. On the other hand, when the temperature is much lower then the temperature threshold, the processor speed is not increased fast enough to maximize the throughput.

To solve these problems, we first introduce a concept called the *temperature buffer zone*. Given a temperature threshold T_{bound}, the temperature buffer zone is defined as the interval of $[T_{safe}, T_{bound}]$, where T_{safe} is determined by the following equation

$$T_{safe} = T_{bound} - \triangle T, \tag{1}$$

where $\triangle T$ is the maximum possible temperature increase within one sampling period. $\triangle T$ can be determined empirically. Using SPEC2000 benchmark, we found that $\triangle T = 4^oC$. When the temperature is lower than T_{safe}, we say that the temperature is located in the *temperature safe region*.

When the processor temperature is located in the safe region, we can safely use the highest possible speed to maximize the throughput before temperature enters into the buffer zone. The problem is how to define the safe speed to run the program and make sure the temperature does not exceed T_{bound}.

Consider the commonly used thermal model as follows

$$\frac{dT(t)}{dt} = aP(s) - bT(t), \tag{2}$$

where $T(t)$ is the temperature at time t, $P(s)$ is the power consumption with processor speed of s, and a,b are the cooling constants. To ensure that temperature does not exceed T_{bound}, we only need to make sure when temperature is located in the buffer zone, we have

$$\frac{dT(t)}{dt} \mid_{T(t)\in[T_{safe},T_{bound}]} = 0, \tag{3}$$

By combining equation (2) and (3), analytically we can solve for processor speed s. However, it can be extremely challenging if not possible to determine the analytical function of $P(s)$ and the cooling constants a, b. Hence, we determine the safe speed empirically from SPEC2000 benchmark. Specifically, let benchmark application set be $\mathcal{T} = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}$. Let $T_{stable}(\tau_i, s_j)$ be the stable temperature when running τ_i using processor speed s_j. Let S_i be the speed such that

$$S_i = \max\{s_j \text{ such that } T_{stable}(\tau_i, s_j). \leq T_{bound}\}. \tag{4}$$

And the safe speed S_{safe} is determined as follows.

$$S_{safe} = \min_{\tau_i \in \mathcal{T}} S_i. \tag{5}$$

Our improved algorithm is depicted in Algorithm 2. When a process is running, the processor uses the highest possible speed to improve its throughput if the temperature is located in the safe region (line 5-6). Otherwise, it adopts the safe speed to make sure the temperature constraint is not violated. With the establishment of the safe region, Algorithm 2 eliminates temperature violations as demonstrated in Figure 4(c).

It is worth mentioning that if the running applications are known a priori, we can further improve the performance of our algorithm by building up a lookup table. The lookup tables list the applications and their specific safe speeds under different temperature constraints, as those defined in equation (4). In that case, we can use the corresponding safe speed depending on the current running process to further maximize its throughput.

Algorithm 2 THE ENHANCED ALGORITHM USING THE SAFE SPEED

```
1: while Process is running do
2:    if T_cur = T_previous; then
3:       Wait T_s = 0.1 seconds;
4:    else
5:       if T_cur ≤ T_safe then
6:          Set processor speed to the maximum speed;
7:       else
8:          Set processor speed to S_safe.
9:       end if
10:      Wait T_r = 0.98 seconds;
11:   end if
12: end while
```

IV. EXPERIMENTS AND RESULTS

In this section, we use experiments to investigate the effectiveness of our newly developed algorithms. The experimental platform is the same as presented in section II. All experiments were carried out with the same ambient temperature and initial chip temperature. We selected several benchmark from SPEC CPU2000, including $gzip, vpr, gcc$ from integer operation and

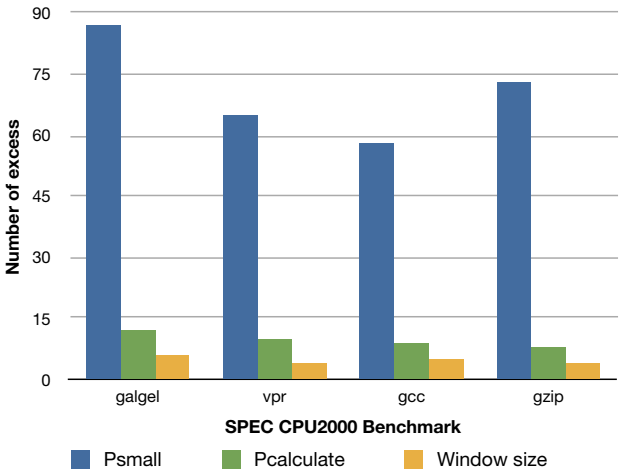


Fig. 5. Temperature violations for different algorithms

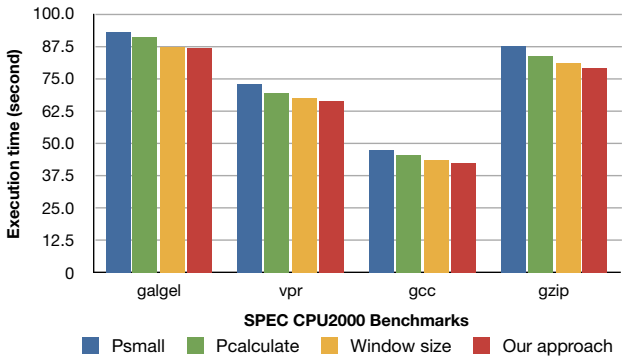


Fig. 6. Performance comparison of different algorithms

$galgel$ from floating operation, to get credible and comparable experiment results. We set the temperature constraint at 55 ^oC.

We first created the lookup table, shown in Table II, as discussed in section III. Table II shows running different applications can result in distinctly different stable temperature, even with the exactly the same speeds. This clearly demonstrates the limitations of many theoretical research results that assume processor's power changing only with its working frequency.

TABLE II
LOOKUP TABLE FOR SPEC 2000 BENCHMARKS

	Frequency levels (GHz)				
SPEC benchmark	2.6GHz	2.4GHz	2.2GHz	2.0GHz	1.8GHz
galgel	64^oC	59^oC	57^oC	54^oC	52^oC
vpr	60^oC	54^oC	53^oC	50^oC	48^oC
gcc	61^oC	55^oC	53^oC	51^oC	47^oC
gzip	61^oC	54^oC	52^oC	51^oC	46^oC

We next studied the performance of our approach in terms of the total number of times that the processor violates the temperature constraint. Four approaches are compared: the approach using very small constant sampling period (namely *Psmall*), the approach using a constant sampling period that is equal to the thermal sensor's shortest response time (namely *Pcalculte*), the approach using the variable sampling periods (namely *Window size*), and the approach that adopts the safe speed and buffer zone (namely *our approach*). The results are plot in Figure 5. In Figure 5, our final algorithm (i.e. *our approach*) guarantees the temperature constraints and there is no temperature violation shown in the figure. In addition, Figure 5 shows that our approach using the variable sampling periods can also significantly reduces the temperature violations times than other two approaches. From the experiment results, we clearly prove that our approach can perfectly control the chip temperature under the threshold.

We next compare the throughput for the same four approaches. The execution times of four benchmarks are plot in Figure 6. From Figure 6, we can see that the first approach always takes the longest execution time for all four benchmarks. By properly increasing the sampling period, the redundant overhead has been significantly reduced and the execution time has been improved as much as 2.4% on average. When using variable sample periods, the throughput performance is further improved because the unnecessary speed switching overhead was reduced. Our final algorithm has the shortest execution time. When comparing with the first algorithm, the execution time could be reduced as much as 7%. The improvement comes from the fact that we use maximum speed to run the benchmark file if the temperature is in the safe region, which outperform the cases when the processor speed has to be increased every sampling period. Also, when the temperature is located in the temperature buffer region, we can use the safe speed to further maximize its throughput and ensure the temperature constraint at the same time.

Overall, the experiment results show that our algorithm can not only precisely control processor temperature under a predefined temperature limit but also significantly improve the system performance.

V. CONCLUSIONS

Dynamic thermal management is becoming more and more critical as more transistors are integrated into one single chip. While there have been extensive theoretical researches conducted on this subject, most of these researches are based on idealized theoretical models and assumptions. In this paper, based on a desktop environment, we study the limitations of the existing theoretical work. We also present a DVFS-based thermal aware scheduling algorithms to maximize the throughput under a given peak temperature constraint. We also use SPEC2000 benchmark to validate the effectiveness of the proposed algorithm.

ACKNOWLEDGEMENT

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, CNS-1018108, NSFC 61071061 and the University of Kentucky Start Up Fund.

REFERENCES

[1] 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*, pages 237 –240, nov. 2009.

[2] M. Bao, A. Andrei, P. Eles, and Z. Peng. Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 21 –26, march 2010.

[3] 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*, ISLPED '09, pages 105–110, New York, NY, USA, 2009. ACM.

[4] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximum temperature minimization for periodic hard real-time systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1802 –1809, 29 2010-july 1 2010.

[5] A. Cohen, F. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy. On estimating optimal performance of cpu dynamic thermal management. *IEEE Comput. Archit. Lett.*, 2:6–, January 2003.

[6] L. L. hardware monitoring. In *http://www.lm-sensors.org*.

[7] H. Huang and G. Quan. Leakage aware energy minimization for real-time systems under the maximum temperature constraint. In *DATE*, 2011.

[8] A. Kumar, L. Shang, L.-S. Peh, and N. Jha. System-level dynamic thermal management for high-performance microprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(1):96 –108, 2008.

[9] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 123 –128, jan. 2011.

[10] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(7):1042 – 1053, 2005.

[11] E. Rohou and M. D. Smith. Dynamically managing processor temperature and power. In *In 2nd Workshop on Feedback-Directed Optimization*, 1999.

[12] M. Santarini. Thermal integrity: a must for low-power-ic digital design. 2005.

[13] 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*, pages 2 – 13, 2003.

[14] E. I. S. Technology. http://www.intel.com/support/processors/sb/cs-028855.htm.

[15] 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*, IFMT '08, pages 1:1–1:4, New York, NY, USA, 2008. ACM.

[16] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10 pp. –170, 0-0 2006.

[17] X. Wang, K. Ma, and Y. Wang. Adaptive power control with online model estimation for chip multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1, 2011.

[18] 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*, SAC '09, pages 1998–1999, New York, NY, USA, 2009. ACM.

[19] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 9 –14, march 2010.

[20] K. M. Yefu Wang and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA '09 Proceedings of the 36th annual international symposium on Computer architecture*, 2009.

[21] L.-T. Yeh and R. C. Chu. Thermal management of microelectronic equipment: Heat transfer theory, analysis methods and design practices. In *ASME Press, New York, NY*, 2002.

[22] H. Yu, B. Veeravalli, and Y. Ha. Leakage-aware dynamic scheduling for real-time adaptive applications on multiprocessor systems. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 493 –498, june 2010.

[23] S. Zhang and K. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 281 –288, nov. 2007.

[24] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 585–590, New York, NY, USA, 2010. ACM.