Dynamic Thermal Management for Multimedia Applications Using Machine Learning*

Yang Ge, Qinru Qiu
Department of Electrical and Computer Engineering, Binghamton University
{yge2, qqiu}@binghamton.edu

ABSTRACT

Multimedia applications are expected to form the largest portion of workload in general purpose PC and portable devices. The ever-increasing computation intensity of multimedia applications elevates the processor temperature and consequently impairs the reliability and performance of the system. In this paper, we propose to perform dynamic thermal management using reinforcement learning algorithm for multimedia applications. The proposed learning model does not need any prior knowledge of the workload information or the system thermal and power characteristics. It learns the temperature change and workload switching patterns by observing the temperature sensor and event counters on the processor, and finds the management policy that provides good performance-thermal tradeoff during the runtime. We validated our model on a Dell personal computer with Intel Core 2 processor. Experimental results show that our approach provides considerable performance improvements with marginal increase in the percentage of thermal hotspot comparing to existing workload phase detection approach.

Categories and Subject Descriptors

C.4 [**Perfomance of Systems**]: Reliability, availability, and serviceability

General Terms

Algorithms, Management, Performance

Keywords

Dynamic thermal management, reinforcement learning, multimedia application

1. INTRODUCTION

The ever-increasing computation demands and escalating computation speed significantly increase the power consumption and operating temperature of current computing systems. Current CMOS technology already hits the "power wall" which blocks the performance improvement of a chip and limits the growth of its clock frequency due to the high temperature. Elevated peak temperatures also reduce life-time of the chip, deteriorate its performance, affect the reliability [10] and increase the cooling cost.

Dynamic Thermal Management (DTM) techniques have been widely studied and employed to control the temperature for different computing platforms, from servers [9], general purpose computers [10], to embedded systems [17]. These works consider applications of various characteristics, including web applications [6], standard benchmarks [5][7] and multimedia applications [11][16]. Among these, multimedia applications are expected to form the largest portion of workload in general purpose personal computers and portable computing devices like smart phones [11]. In spite of their popularity, the computation intensity of multimedia applications is likely to produce high temperature in these platforms [16]. A thermal safe solution is to run the applications at lower speed or reduce the computation by decreasing the *quality of service* (*QoS*). However these solutions impact the user satisfactory.

In this paper, we consider the problem of dynamic thermal management for multimedia applications. We utilize the processor's dynamic voltage and frequency scaling (DVFS) ability to control the operating temperature under a threshold while maximizing the system performance, i.e. minimizing the CPU time of the multimedia applications. Our DTM technique does not require to pre-characterize the system for its thermal and power model neither does it need any prior knowledge of the workload information. It relies on machine learning algorithms to find the best management policy during the runtime. Compared to the existing DTM techniques [5][11] it provides considerable performance improvements with marginal increase in the percentage of thermal hotspot.

We model the dynamic thermal management problem as a stochastic control process and adopt the reinforcement learning algorithm to find the optimum policy during runtime. We consider the processor's DTM controller as a learning agent and model the rest of the system, e.g. the operating temperature, the hardware and the application status as the environment. After the learning agent, i.e. the DTM controller, takes an action (i.e. switching to a new operating frequency), it observes the environment and estimates the reward or penalty caused by this action. The agent learns from this experience and tries to improve its future action selections to maximize the reward (or minimize the penalty).

Most of the multimedia applications such as MPEG movie clips or MP3 files are naturally arranged into frames. The computation load of processing each frame has high temporal correlation. We exploit this property and perform the learning based DTM at a single frame granularity. The application status is characterized by its frame level computation load which can be obtained from the processor's performance counter.

The characteristics of the proposed work are summarized as the following:

- This is the first work that applies reinforcement learning algorithm to solve the problem of dynamic thermal

* This material is based upon work supported by the National Science Foundation under Grant No. CNS-0845947

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC'11, June 5-10, 2011, San Diego, California, USA
Copyright © 2011 ACM 978-1-4503-0636-2/11/06...$10.00

management. The proposed approach is truly adaptive. The learning agent does not require having any prior knowledge of the environment or the system power/thermal characteristics. It learns from the experience and adjusts the policy online. Therefore, it works robustly in different computing systems.

- The proposed learning algorithm explores the frame level temporal correlation in multimedia applications. The environment observation and policy adaptation are performed at single frame granularity.
- The proposed learning model has very small run time overhead. It only incurs a few table lookups and some simple arithmetic operations.
- Instead of simply minimizing the thermal violation, our goal is to maximize the performance without increasing the thermal violation.
- The proposed learning model is validated on a Dell Dimension 9200 desktop PC with Intel Core 2 processor. All the experimental results data reported in this paper are gathered with the consideration of the real implementation and control overhead.
- Compared to running the application without dynamic thermal management and the existing DTM techniques that utilize the same event counter information, the learning based DTM provides better performance-thermal tradeoff.

The rest of the paper is organized as follows: Section 2 reviews the related work. We discuss how to apply the reinforcement learning model in detail in Section 3. Experimental results are reported in Section 4. Finally, we conclude the paper in Section 5.

2. RELATED WORKS

Dynamic thermal management (DTM) has been studied for different types of applications from general purpose industry standard benchmarks to multimedia applications. Reference [5] and [7] focus on thermal management techniques for SPEC benchmarks. The authors of [7] propose to dynamically adapt some micro-architecture parameters, such as instruction window size, issue width, and fetch gating level, to the application characteristics and hence control the processor temperature. The authors of [5] propose to utilize the processor's performance counter readings to detect the phase changes of application at run time and adjust the operating frequency accordingly to avoid thermal violations. Both these works perform the thermal management at a regular time interval. Although this is effective for standard benchmarks, as we will show in Section 3.2, it might not be the same for multimedia applications.

There have also been a number of studies of thermal management for multimedia applications. Reference [16] proposes to profile the number of cycles to decode each frame. With this information and a temperature prediction model, an operating frequency is selected for a group of frames to guarantee the QoS while minimizing the temperature. Our work is different from [16] as we try to maximize performance while stratifying the thermal constraint. The thermal management scheme in [11] is based on the observation that the same type of frames has similar average IPC and power consumption, so the same configuration is applied to these frames.

One common drawback for the aforementioned works is that they rely on certain system models or profiled information. For example, [7] utilizes a neural network model to predict future temperature for a set of architecture parameters and application characteristics while [5] uses a linear regression model to predict future temperature. Even though those prediction models are carefully characterized, they suffer from lack of adaptability. Once the model has been trained or obtained, it cannot be modified. Any change in the system or environment (e.g. room temperature variation, chip aging) will invalidate the model and thus hit the performance of the thermal management scheme.

Model free online learning techniques should be a solution to this problem. Previous work [13] has successfully applied the online learning model to dynamic power management (DPM) problems, but few works have applied the learning techniques to the DTM problem.

3. REINFORCEMENT LEARNING FOR THERMAL MANAGEMENT

In this section, we will first give the formulation of Q-learning in its standard form in Section 3.1 and then discuss how to apply this technique to the thermal management problem in Sections 3.2~3.5.

3.1 The Reinforcement Learning Model

Reinforcement learning [12] is an unsupervised machine intelligence approach that has been applied in many different areas. The general learning model consists of

- An agent, with a finite action set $\boldsymbol{A}$.
- The environment that has a finite state space $\boldsymbol{S}$. The actions of the agent will affect the future states of the environment, which in turn affects the options and opportunities available to the agent at later times.
- A policy $\boldsymbol{\pi}$ that defines the behavior of the learning agent at any given time. It is a mapping from the set of environment states to the set of actions, i.e. π: $\boldsymbol{S} \rightarrow \boldsymbol{A}$.
- A reward function $\boldsymbol{r}$: $\boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{R}$ which maps each state-action pair to a single real number, ($\boldsymbol{R}$ is the real number set). A *reward* indicates the intrinsic desirability of taking an action at one particular state.

The process of reinforcement learning is divided into multiple decision steps. We refer to each decision step as *decision epoch*. At each *decision epoch*, the agent takes an action according to current environment state. It then observes the environment for the reward/penalty caused by this action. The goal of the agent is to maximize the average long-term reward by trial-and-error interaction with a dynamic environment. It is achieved by learning the policy π, i.e. a mapping between the states and the actions. The agent might not select the optimum action at beginning, but its performance can be improved over time.

The Q-learning algorithm is one of the most popular reinforcement learning algorithms. In Q-learning, the agent keeps a value function $Q^{\pi}(s,a), s \in \boldsymbol{S}, a \in \boldsymbol{A}$, for each state-action pair and stores it in a Q-table. The value function represents the expected long-term reward if the system starts from state s, taking action a, and thereafter following policy π. Based on this value function, the agent decides which action should be taken in current state to achieve the maximum long-term rewards. An optimum policy $\boldsymbol{\pi}^*$ is a policy which achieves the maximum value function denoted as Q^*, i.e. $Q^* = Q^{\pi *}(s,a) = \max_{\pi} Q^{\pi}(s,a)$.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \\ \times \left[r_{t+1} + \gamma \times \max_{a} Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

The core of the Q-learning algorithm is to iteratively update the value function $Q^{\pi}(s, a)$ as above [13]. In this equation, α is the learning rate which determines how fast the Q value will adapt. The discounted factor γ is between 0 and 1.

3.2 Frame Based Decision Epoch

How frequent an agent observes its environment, updates the policy and issues DTM command is determined by the length of the decision epoch. An appropriate decision epoch can help the system to learn and control effectively. Within a decision epoch, the system should have consistent behavior. Across multiple decision epochs, the system behavior should exhibits repeated patterns. Here we define a *workload phase* to be an execution interval during which the application has near identical power, temperature and performance characteristics [5]. It is obvious that the workload phase of an application determines the decision epoch of its DTM agent.

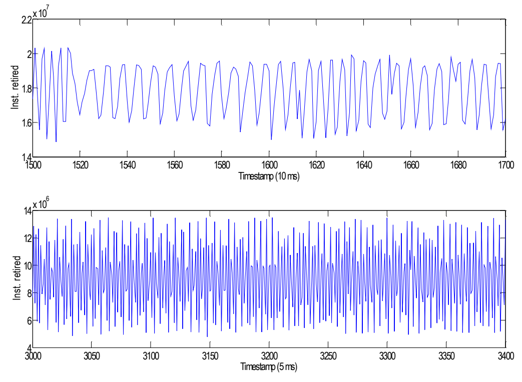

Figure 1 Variation of retired instructions for every 10 and 5 ms for MPEG4 decoder

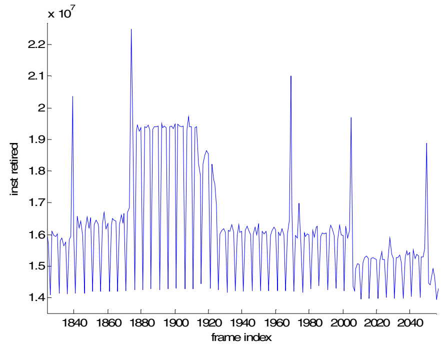

Figure 2 Variation of retired instructions at frame granularity

For a multimedia application, the decision epoch can be divided in two ways: frame based decision epoch and equal time step decision epoch. An equal time step decision epoch at the granularity of 100 *ms* has been used for the DTM of SPEC CPU2006 benchmarks [5]. This is because the workload phase change of the applications in SPEC CPU2006 benchmarks can be detected at this granularity. (Those applications will stay in the same workload phase for several to tens of seconds before moving to next phase [5].) However, the same equal time step decision epoch is not suitable for multimedia applications. Figure 1 shows a segment of trace of retired instructions for the MPEG4 decoder from the MediaBench [1]. The upper part and lower part of the figure show at time interval of every 10 ms and every 5 ms respectively. The number of retired instructions has been reported as one of the architectural events that contribute most for the temperature change [5]. From this figure, we can see that instruction retired number is constantly vibrating and does not show obvious phase change. This is because equal time epoch smoothes out the inner workload phase change of the application. On the other hand, Figure 2 demonstrates the trace of the retired instructions every frame for a total of about 200 frames. As shown in this figure, phase change can be clearly observed with a regular pattern. Therefore in this work we choose frame based learning epoch.

3.3 Interactions Between Agent and Environment

Figure 3 presents the system model of our learning based DTM agent. The actions that the agent takes are the available frequency levels of the processor. At each decision epoch, the agent observes the current state of the environment and chooses the frequency level for the next frame according to the Q-values in the Q-table. The Intel's speedstep technology [2] or AMD's PowerNow technology [3] can be used to dynamically switch the voltage and frequency of a processor.

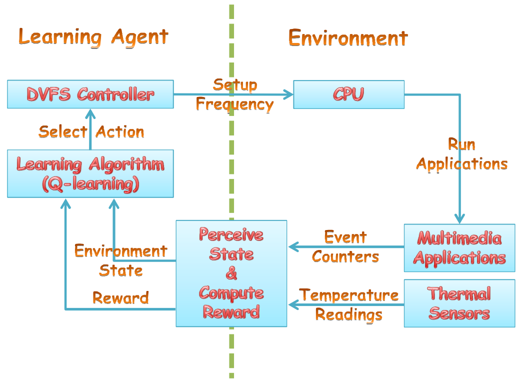

Figure 3 System model

Everything outside the agent is considered as environment, including the processor, the applications and the thermal monitoring system (thermal sensors). The environment and the agent are closely coupled. For instance, the power consumption of the processor is linearly dependent on the operating frequency, which in turn determines the processor's temperature. As another example, because the agent only changes the clock frequency of the processor not the clock frequency of the memory subsystem, its actions will affect the instruction per cycle (IPC) of the system. On the other hand, the environment status limits the agent's actions in some cases by changing the reward/penalty value. For example, the agent is not allowed to run at full speed when the processor is approaching the temperature threshold.

3.4 Classification of Environment States

The environment can be characterized by many features, from the temperature and the power consumption of the processor, to the cache miss rate and IPC. The learning agent works under a discrete state space. How to map this huge and sometimes continuous feature space into a finite set of discrete state space has direct impact to the effectiveness of the learning algorithm. Here, two problems are involved.1) Which features should be selected to represent the environment; 2) How to discretize the selected feature space into the state space.

The rules of thumb of selecting features to represent the environment are: first, those features should closely relate to our problem, i.e. performance optimization with thermal constraint; second, they could be observed easily.

It is obvious that the processor's temperature should be one of the features to represent the environment state as it is directly related to our optimization problem. This information can be obtained by reading the temperature sensors that are equipped on most state-of-the-art processors.

We may also want to use the processor power consumption as one of the features for environment state, because it directly contributes to the temperature change of the processor. Unfortunately, this information is not easy to obtain. Therefore, we decided to use the readings of performance counters as a proxy for the power consumption as they reflect the processor's switching activities. Besides power consumption, the performance counter readings, such as the execution cycles, instruction retired rate and cache miss rate, etc., also reflect the performance of the application programs, which is what we want to optimize.

Our test platform, the Intel's Core 2 Duo processor has 5 performance counters which could record 130 architectural events [4]. Among them, we select those events that contribute most to the temperature change and are most relevant to the system performance. Reference [5] utilizes the principle component analysis to find the contribution of each architectural event to the temperature change and suggests that 3 of them, i.e. the "instruction retired" event, the "floating point instructions executed" event and the "conditional instructions executed" event, play the most important roles in temperature change. However, our analysis shows that, for a typical multimedia application such as an MPEG-4 decoder, the variation of floating point instructions executed among different frames is very small. We also found that the number of conditional instructions executed has high correlation with the number of retired instructions and thus does not provide much additional information.

We analyzed the correlations between different events and retired instructions that were recorded during an MPEG decoding process. Table I gives the selected results. The results indicate that the "last level cache miss" event has the least correlation with the "instruction retired" event and hence provides the most additional information about the system. Therefore we use the "last level cache miss" event together with the "instruction retired" event to represent the environment state. The former represents the memory activities while the latter specifies the computation activities. Combined together they provide a complete picture of the system. This choice also agrees with what is suggested in reference [8].

Table I Correlation between different events and retired inst.

UOPS RETIRED	BR_INST RETIRED	BR_CND EXEC	FP_INST EXEC	BUS _MEM TRANS	LAST_LEVEL $_MISS
0.997	0.977	0.964	0.763	0.762	0.669

Based on the above analysis, we will use the feature set $(T, \boldsymbol{P})$ to characterize the environment, where T is the reading from the thermal sensors and $\boldsymbol{P}$ is a vector of selected event counter readings. In the next, we will discuss how to map the feature space into a finite state space to apply the reinforcement learning model.

Note that T can be any real number within the working range, usually from 0°C ~ 100°C. Because it is a continuous variable, we have to discretize it to get a finite set of states. We divide the temperature working range into a set of disjoint intervals, i.e. $(0,\ T_0]$, $(T_0,\ T_1]$, $(T_1, T_2]$, … , $(T_{N-1}, T_N = T_{threshold}]$, (T_N, ∞), each interval $(T_{i-1},\ T_i]$ corresponds to a state T_i. Note that the (N+1)th interval covers all temperatures beyond the threshold. Although the temperature level T_0, … T_{N-1} can be set arbitrarily, we would divide the region near the threshold temperature in finer granularity while leave the other region in coarse granularity. In this way when the temperature is approaching the threshold, the agent might take better control at finer resolution.

In order to classify the space of the event counter readings (i.e. $\boldsymbol{P}$) we use the k-means clustering method as in [5]. We took 5 representative video clips and collect the retired instructions number and cache miss number for each frame, and classify them by the standard k-means algorithm [14]. To find the optimum number of states (i.e. number of clusters), we start from a small number and gradually increase it until the classification error is less than 5%, which is defined as the ratio between square sum of all points' within cluster distance and the square sum of their distance to the origin. Based on k-means clustering, we divide the event counter values into K states $\{P_0, \cdots, P_{K-1}\}$. Together with the N temperature states, the size of the resulting learning space is $|N|\times|K|$. In our implementation, we set N=11 and K=10.

3.5 Design of the Reward Function

The state of the ith decision epoch is denoted as (T^i, p^i), where T^i give the temperature at the beginning of the ith epoch and p^i gives the number of cache misses and the number of instruction retired during the ith epoch. The action taken by the agent during the ith epoch is denoted as a^i. At the end of the ith epoch (or the beginning of the (i+1)th epoch, the agent calculates the reward caused by the action and update the Q-function of state action pair (T^i, p^i, a^i),The reward function is defined as the following:

$$r\big((T^i, p^i), a^i\big) = \begin{cases} -PT, \ \ if\ T^{i+1} = N+1 \\ A \cdot Inst(p^{i-1}) \cdot Freq(a^i) \\ +B \cdot T^{i+1}, \ \ if\ T^{i+1} \neq N+1 \end{cases}$$

where T^{i+1} is the temperature state at the end of the ith epoch (or the beginning of the (i+1)th epoch), $Inst(p^i)$ is the number of retired instructions, $Freq(a^i)$ is the processor frequency selected by action a^i, and state (N+1) is the temperature state that covers all temperatures beyond the threshold. In this function, P, A and B are constants. The upper part of the reward function is the thermal violation penalty, which is a negative number. If at the end of the ith epoch, we reach a thermally unsafe state (i.e. $T^{i+1} = N+1$), then a negative reward will be received. A negative reward will decrease the Q-value of state action pair (T^i, p^i, a^i) so that this action will be avoided at this state in the future.

The lower part of the reward function is used when the system is thermally safe at the end of the ith epoch. In this scenario, we would like to increase the performance of the system. The first part of the reward function is the performance. We use the product of the number of retired instructions and the processor frequency to represent the performance reward. This encourages the agent to select high frequency for frames with high computation demand. The intuition behind this is that use high frequency for complex frames can reduce more execution time than using it for simple frames. The second part of the reward function represents the thermal award. The parameters A and B provide tradeoff between temperature and performance.

4. EXPERIMENTAL RESULTS

4.1 Experiment Setups

We carried out our experiments on a Dell Dimension 9200 desktop with Intel Core 2 Duo E6400 processor which has 2MB

L2 cache and 1333 MHz FSB. The processor supports 8 frequency levels: 267 MHz, 533 MHz, 800 MHz, 1.07 GHz, 1.33 GHz, 1.63 GHz, 1.87 GHz and 2.13 GHz. The operating system is Fedora 11 with Linux kernel 2.6.29.

To monitor the temperature change of the processor, we utilize the thermal sensors available in each core [15]. The *coretemp* driver in the Linux kernel generates an interface under */sys/devices/ platform/coretemp.[X]* directory (X is the index of each core), and the current temperature will be reported when the file *temp1_input* is read each time. The default driver only updates temperature readings once every 1 second. This granularity is too coarse for our frame based management scheme, because the decoding time for a frame is about tens of milliseconds. We modified the driver so that it could update its readings every 10 ms.

We utilize Intel Enhanced SpeedStep [2] technology to adjust the frequency level. Similar to temperature readings, the Linux kernel provides the *cpufreq* driver for users to read and modify the operating frequency. Processors equipped with DVFS ability will have an interface under */sys/devices/system/cpu/cpu[X]/cpufreq/* directory (X is the index of each core). It has been reported in [15] that the overhead for each frequency adjustment is about 20 us, therefore the overhead for DVFS at every frame is under 2%.

To collect the performance counter readings, we utilize the pfmon 3.9 hardware monitoring tool [4]. Two event counters are monitored in our program, i.e. the instruction retired event and cache miss event. The monitoring is trigger at the end of each frame.

We choose the MPEG-4 decoder from the MediaBench benchmark [1] as our application. Please note that the proposed method can be readily applied to other multimedia applications, such as MPEG-2 decoder, H.264 decoder etc, because they have similar characteristics. We apply MPEG-4 decoder on 5 video clips extracted from recent movies of different genres, e.g. drama, action, animation.

We compare our reinforcement learning based DVFS thermal management policy with the Phase-Aware dynamic thermal management policy proposed in [5]. In Phase-Aware DTM, performance counter values are collected every 100 ms. Then the readings are classified into different phases. Based on a linear temperature prediction model, the max frequency which is guaranteed to be thermal safe under current phase will be selected. It is important to point out the effectiveness of the phase aware DTM heavily relies on the accuracy of the temperature prediction model. We also compare our policy with two scenarios that run the entire application at 1.63 GHz and 1.87 GHz clock frequencies without any dynamic thermal management.

4.2 Results Analysis

In the first set of experiments, we compare the run time and thermal violations among our reinforcement learning based policy, Phase-Aware policy, and two single operating frequencies and the results are shown in Figure 4 and Table II. The thermal violation is defined as the percentage of time that the processor temperature is above the given threshold. We use 1.63 GHz as the base line frequency and set the peak temperature under this frequency as the thermal threshold. The reason that we use a floating temperature threshold instead of a fixed temperature threshold is to cancel the impact of the ambient temperature, which cannot be controlled by us. Using the next available higher level frequency could reduce the run time significantly; however, without any thermal management, it also incurs large thermal violations. The average thermal violation for 1.87 GHz is 36.65%. On the other hand, learning based policy provides large performance improvement with very small thermal violation. For example, our Learning policy improves the run time by 22.15% and 7.53% over the 1.63 GHz policy and the Phase-Aware policy, while only incurs 2.38% and 1.9% more thermal violation respectively. And compared to the 1.87 GHz clock frequency, the Learning policy reduces thermal violation by 34.27% while maintaining the similar run time. The reason that the Learning policy has marginal higher thermal violation than the 1.63 GHz and the Phase-Aware policy is because, unlike the Phase-Aware policy, it does not employ a temperature prediction model and has to try frequency settings at different states. Therefore, the Learning policy will make some mistakes "on purpose" in order to learn the optimum policy.

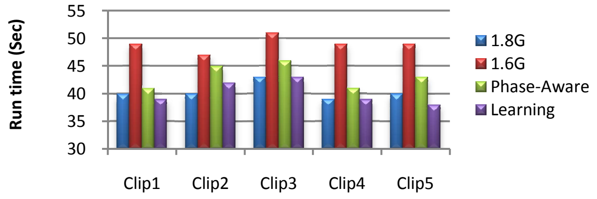

Figure 4 Comparison of run time for different policies

Table II Comparison of thermal violations in % of time

Policy	Clip1	Clip2	Clip3	Clip4	Clip5
1.6G	0	0	0	0	0
Phase-Aware	0.49	0.4	0.65	0.25	0
Learning	1.78	3.31	2.83	1.26	2.7
1.8G	33.66	60	21.95	42.65	24.98

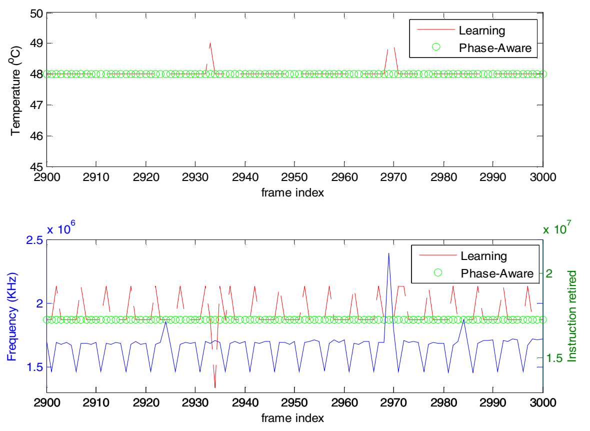

Figure 5 Temperature and frequency comparison between Learning and Phase-Aware

We observed in our experiments that Learning based policy is more aggressive than the Phase-Aware policy. This is illustrated in Figure 5, which shows the temperature variation (upper part) and operating frequency (lower part) for an interval of 100 frames. The green circle line shows the data for the Phase-Aware policy while the red dashed line shows the data for the learning policy. We also plot the number of instruction retired for each frame in the lower figure (the blue line). As shown in the figure, both control policies run at the thermal threshold 48^{o}C for most of the time, while the Learning policy incurs minor thermal violation. The Phase-Aware policy fix the frequency at 1.87 GHz. while Learning based algorithm is able to perceive the state change at the frame granularity and learns a control policy that alternates the clock frequency between 2.13 GHz and 1.87 GHz.

Figure 6 shows the percentage of time that Learning and Phase-Aware policies run at each frequency. As shown in the figure, learning policy was able to run at the highest frequency for more than half the time, while the Phase-Aware policy is more conservative and runs at the second highest frequency for most of the time.

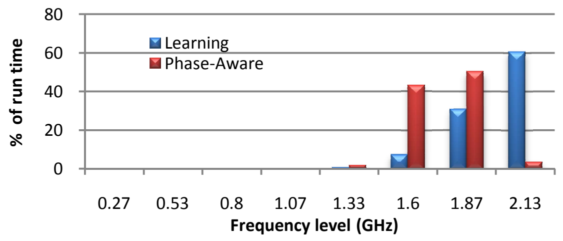

Figure 6 Percentage of time of each frequency for Learning and Phase-Aware policy

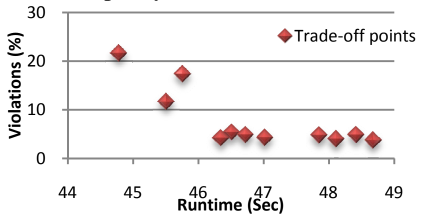

Figure 7 Trade-off between run time and thermal violation

Our Learning policy is very flexible. We can achieve different performance-thermal violation trade-offs by changing the parameters in the reward function. In the second set of experiments, we vary the thermal violation penalty *PT* in the reward function from 500, 1000 ~ 10000 by a step of 1000 and obtained a set of trade-off points which are shown in Figure 7. When the thermal violation penalty is small, the learned control policy tends to be more aggressive. On the other hand, when the penalty is large, the learned control policy tends to be more conservative. This trade-off is a unique property of our Learning policy and could not be achieved by the Phase-Aware policy. This property could be useful when the reliability is considered as a resource that can be used to trade for performance improvement.

Table III Comparison of different state representation

Clip #	State Rep.	Inst + $	Inst only	Impr. (%)	$ only	Impr. (%)
Clip 1	Violations (%)	3.80	4.65	0.85	5.58	1.78
	Run time (Sec)	40.70	42.61	4.71	43.53	6.96
Clip 2	Violations (%)	4.90	5.16	0.26	4.94	0.04
	Run time (Sec)	41.02	40.90	-0.28	42.81	4.37
Clip 3	Violations (%)	5.24	6.30	1.06	5.47	0.23
	Run time (Sec)	42.81	44.79	4.63	47.49	10.95
Clip 4	Violations (%)	3.66	5.90	2.24	6.25	2.59
	Run time (Sec)	45.32	47.40	4.60	45.20	-0.27
Clip 5	Violations (%)	2.53	2.99	0.46	3.43	0.90
	Run time (Sec)	37.35	39.38	5.45	38.77	3.79

In the last set of experiments, we tested the impacts of different environment state representations to our learning policy. We tested three kinds of state representations: retired instructions with cache misses, retired instructions only and cache misses only. We applied the same Q-learning algorithm using these state representations and compared the run time and thermal violations. Table III shows that combining these two variables together could result in optimum performance. Compared to systems using only retired instructions or cache misses, in average, it reduces the run time by 3.82% and 5.16% respectively and also has 0.974% and 1.108% lower thermal violations respectively. The results indicate that the learning agent receives more information about the environment when monitoring these two variables, and hence makes better control decisions.

5. CONCLUSIONS

In this paper, we proposed reinforcement learning based dynamic thermal management method for multimedia applications. The agent learns the workload pattern of the application based on the performance counter readings, and adjusts the processor's operating frequency at the beginning of each frame to optimize the performance while ensuring thermal safety. We implemented our learning based DTM policy on a personal computer and tested it using real application. Our experimental results show big performance improvement with only marginal thermal violations compare to a thermal management policy that also based on workload phase detection.

6. REFERENCES

[1] MediaBench: http://euler.slu.edu/~fritts/mediabench/

[2] "Enhanced Intel SpeedStep® Technology - How To Document", http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm

[3] AMDPowerNow:http://www.amd.com/us/products/technologies/power-management/Pages/power-management.aspx

[4] Perfmon2: http://perfmon2.sourceforge.net/pfmon_usersguide.html

[5] R. Cochran and S. Reda, "Consistent Runtime Thermal Prediction and Control Through Workload Phase Detection," *In Proc. Design Automation Conference*, June 2010.

[6] A. Coskun, T. Rosing and K. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *In IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp 1503-1516, Oct. 2009.

[7] R. Jayaseelan, T. Mitra, "Dynamic Thermal Management via Architectural Adaption", *In Proc. Design Automation Conference*, pp 484-489, Jul. 2009.

[8] H. Jung, P. Rong and M. Pedram, "Stochastic Modeling of a Thermally-Managed Multi-Core System," *In Proc. DAC*, June 2008.

[9] E. Pakbaznia, M. Ghasemazar, and M. Pedram,"Temperature Aware Dynamic Resource Provisioning in a Power Optimized Datacenter," *In Proc. of Design Automation and Test in Europe,* Mar. 2010.

[10] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Trans. on Architecture and Code Optimization*, Vol. 1 Issue 1, pages 94-125,Mar. 2004.

[11] J. Srinivasan and S. Adve, "Predictive dynamic thermal management for multimedia applications," *In Proc. ICS*, 2003

[12] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", MIT Press, Mar. 1998.

[13] Y. Tan, W. Liu and Q. Qiu, "Adaptive Power Management Using Reinforcement Learning," *In Proc. ICCAD*, Nov. 2009.

[14] P. Tan, M. Steinbach, V. Kumar, "Introduction to Data Mining", Addison-Wesley, May. 2005.

[15] Y. Wang, K. Ma and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," *In Proc. ISCA*, 2009, pp. 314-324.

[16] I. Yeo, E. Kim, "Hybrid Dynamic Thermal Management Based on Statistical Characteristics of Multimedia Applications," *In Proc. ISLPED*, Aug. 2008.

[17] S. Zhang and K. Chatha, "Thermal aware task sequencing on embedded processors," *In Proc. DAC*, June 2010.