Learning Based DVFS for Simultaneous Temperature, Performance and Energy Management

Hao Shen†, Jun Lu* and Qinru Qiu†

†EECS Department, Syracuse University
Syracuse, NY, USA
{hshen01, qiqiu}@syr.edu

*ECE Department, Binghamton University
Binghamton, NY, USA
jlu5@binghamton.edu

Abstract

Dynamic voltage and frequency scaling (DVFS) has been widely used for energy reduction in the modern processors. How to select the optimal frequency that minimizes energy dissipation for the given performance constraint at runtime is a nontrivial problem. The problem becomes more complicated if temperature needs to be constrained (or minimized) simultaneously. The temperature, performance and energy have different nonlinear relationships with frequency/voltage scaling ratio and this relationship is closely related to the characteristics of hardware and applications. In this paper, we design a reinforcement learning algorithm to tackle the problem of simultaneous temperature, performance and energy management. The proposed approach allows continuous tradeoff among these three quality measurement of a computer system. It also enables us to set two of the measurements as constraints and optimize the third one. The proposed approach is validated on an Intel Core 2 processor running Linux system.

Keywords

DVFS, dynamic voltage and frequency scaling, temperature, performance, energy, enhancement learning

1. Introduction

Dynamic voltage and frequency scaling (DVFS) has been widely used in modern processors for energy reduction or temperature control. Traditionally, reducing the voltage and clock frequency of a digital IC is considered to give cubical reduction in its power consumption and linear reduction in its performance. However this trend has started to change. As semiconductor technology keeps scaling down, leakage power becomes more and more dominant in modern processors [1][3]. Although the DVFS technique effectively reduces the dynamic energy, it also increases the leakage energy because the system has to be kept active for a longer time [1][4]. On the other hand, as the CPU speed increases, the limited memory bandwidth has become the performance bottleneck for many applications with intensive memory access. For those memory bound applications, the DVFS technique incurs less performance penalty because the memory subsystem still works under a constant frequency [1][2][4][5].

DVFS technique is supported by many modern computing hardware and operating systems. For example, the Enhanced Intel SpeedStep [11] technology allows the user to dynamically adjust processor voltage and frequency. Modern OS usually provides several power management options to meet different user performance requirement. For example, the Linux OS provides several processor frequency governors including Performance, Powersaver, Ondemand, and Conservative governors [19]. Similarly, the Windows OS also provides power management options such as "High performance", "Power saver" and "Balanced" power management. The IBM Thinkpad [10] performs power management based on provided energy profiles. The above mentioned power management options give users certain flexibility to find trade-off between performance and energy saving. However the trade-off space is limited to those provided policies.

Many research works have been proposed to find the optimal DVFS scheduling for energy and temperature reduction. Reference [5] uses runtime information on the statistics of the external memory access to perform CPU voltage and frequency scaling. Its goal is to minimize the energy consumption while translucently controlling the performance penalty. The authors of reference [1] take different frequencies of the processor as different experts. These experts are dynamically selected based on their weight, which is a function of the energy dissipation and performance penalty and is updated online. Reference [7] first presents a workload prediction model for MPEG decoder and the predicted workload is further used to guide the voltage and frequency scaling. Reference [6] presents a set of new job scheduling and power management policies for chip multiprocessors. Their impact on chip lifetime is evaluated. The authors of reference [8] use machine learning to adaptively change the frequency of the processor for the thermal management of multimedia applications. Reference [9] considers processors as producers and consumers and tunes their frequencies in order to minimize the stalls of the request queue while reducing the processors' energy.

All of the previously mentioned work focus on either energy or temperature control. In this paper, we target at simultaneous management of temperature, performance and energy (TPE) management. A low-overhead machine learning approach is proposed to dynamically select the processor's voltage and frequency to achieve the required balance among energy, performance and temperature of the processor. The proposed controller is capable of achieving almost continuous tradeoff in the temperature, performance and energy space.

The rest of the paper is organized as follows: Section 2 discusses the impact of processor frequency/voltage scaling on its energy, performance and temperature. We will show the non-simple tradeoff space among energy, performance and

*This work is supported in part by NSF under grant CNS-0845947

temperature, which gives the motivation of learning based detail and Section 4 provides the experimental results. We conclude the paper in Section 5.

2. Impact of DVFS on processor energy, temperature and performance

In this section we will investigate the relation between processor's voltage/frequency and its energy, performance and temperature. First order analysis of the tradeoff space among these three parameters is provided afterward.

In the following analysis, we use V_{max} (F_{max}) and V (F) to represent the maximum voltage (frequency) and the scaled voltage (frequency) of the processor. We use v and f to represent normalized voltage and frequency, i.e. $v = V/V_{max}$ and $f = F/F_{max}$. Obviously, v and f are also the scaling ratios of CPU voltage and frequency.

We adopt the convention of reference [1] and refer the time when the CPU is actively executing as T_{cpu} and the time when the CPU stalls (due to memory access and etc.) as T_{stall}. During T_{cpu}, the CPU has both dynamic power consumption (P_D) and leakage power consumption (P_L), while during T_{stall} there is only leakage power consumption. The total energy of the processor can be calculated as the following:

$$Ene = (P_D + P_L)T_{CPU} + P_L T_{stall} \quad (1)$$

Our study on many commercial processors' voltage and frequency settings [2][4][5][20] shows that the scaling ratio of the voltage is usually less than the scaling ratio of frequency. In other words, the scaled voltage V is usually an increasing and concave function of scaled frequency F. In order to facilitate our following discussion, we use $V = aF^{\alpha}$ to approximate the relation between F and V, where V is the minimum supply voltage that achieves frequency F, a and α are hardware related parameters and $\alpha < 1$. It is easy to derive the following relation between v and f:

$$v = {}^{V}/_{V_{max}} = ({}^{F}/_{F_{max}})^{\alpha} = f^{\alpha} \quad (2)$$

As in [1], we define ρ as the percentage contribution of leakage power to the total power consumption at the highest v-f setting when processor is running, so $(1-\rho)$ is the percentage contribution of dynamic power consumption. We define μ as the percentage of time that the application spends on CPU when the processor is configured at the highest voltage and frequency, so $(1-\mu)$ is the percentage of time that the application spends on memory access and other CPU stall events. According to reference [1], the normalized dynamic power (p_D) and leakage power (p_L) can be calculated as: $p_D = P_D/P_{max} = (1-\rho)v^2 f$ and $p_L = P_L/P_{max} = \rho v$, where P_{max} is the total power consumption of the processor when it is running at the highest clock frequency.

Using Equation(1), the normalized energy (*ene*) can be calculated as the following:

$$ene = \frac{Ene}{P_{max}T_{max}} = \left((1-\rho)v^2 f + \rho v\right)\frac{\mu}{f} + \rho v(1-\mu) \quad (3)$$

where T_{max} is the total execution time when the processor running at the maximum frequency. Note that the T_{CPU} can be reduced by increasing the frequency f while the leakage power during T_{stall} has nothing to do with the CPU frequency.

exploration. Section 3 introduces the proposed controller in

We replace v with f^{α} in Equation(2) and rewrite the normalized energy as a function of frequency scaling ration f:

$$ene = \left((1-\rho)f^{2\alpha} f + \rho f^{\alpha}\right)\frac{\mu}{f} + \rho f^{\alpha}(1-\mu)$$
$$= (1-\rho)\mu f^{2\alpha} + \rho(1-\mu)f^{\alpha} + \rho\mu f^{\alpha-1} \quad (4)$$

The normalized execution time t can be calculated as:

$$t = T/T_{max} = \frac{\mu}{f} + (1-\mu) \quad (5)$$

The performance is inversely proportional to the execution time. Hence it can also be written as a function of frequency scaling ratio f:

$$perf = \frac{1}{t} = \frac{f}{(1-\mu)f+\mu} \quad (6)$$

In order to obtain a close form expression of the relation between temperature and normalized frequency f, we use a rough approximation that assumes the temperature to be proportional to the average power [18] . Therefore, the temperature and clock scaling ratio satisfy the following relation:

$$temp \propto p = \frac{ene}{t} = \frac{(1-\rho)\mu f^{2\alpha} + \rho(1-\mu)f^{\alpha} + \rho\mu f^{\alpha-1}}{\frac{\mu}{f} + (1-\mu)}$$
$$= \frac{(1-\rho)\mu f^{2\alpha+1} + \rho(1-\mu)f^{\alpha+1} + \rho\mu f^{\alpha}}{(1-\mu)f+\mu} \quad (7)$$

Equations (4)(6)(7) gives the relations between the normalized frequency f and energy, performance and temperature. For example, given the parameters ρ=0.3, μ = 0.75 and a=0.45, Figure 1 shows how these three design metrics change as clock frequency scales.

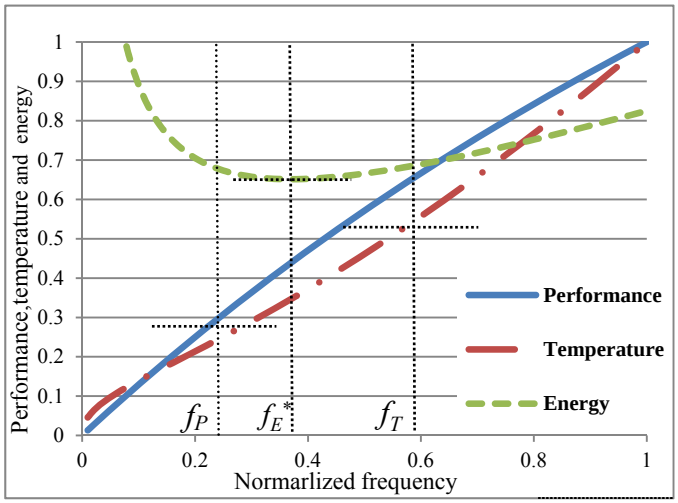

Figure 1 T, P and E versus clock frequency

Given equations (4)(6)(7) and the condition that ρ, μ and α are in the range (0, 1), the following can be proved:

1. The processor energy is a convex function of normalized frequency. It first decreases and then increases as the frequency increases.
2. The performance is a concave and increasing function of normalized frequency.
3. The temperature is a convex and increasing function of normalized frequency.

4. The optimal frequency for minimal energy increases when μ increases.

5. The performance curve will shift to right when μ increases.

6. The temperature curve will shift to left when μ increases.

In the previous discussions, for simplicity, we didn't consider the fact that leakage power will increase super-linearly with the increasing of temperature [12]. Intuitively, such relation between leakage power and temperature will shift the energy curve to the right and increase the curvature of the temperature curve. It will not affect the performance curve.

The relation among temperature, performance and energy determines the possible tradeoff space of the TPE management. From property 1 we know that there is an optimal frequency that minimizes the energy for the given ρ, μ and α parameters. We denote this frequency as f_E^*. The user imposed performance and temperature constraints determine if such optimal frequency is achievable. Let f_P denote the minimum frequency that satisfies the given performance constraint, and f_T denote the maximum frequency that satisfies the given temperature constraint. A valid clock frequency f must be greater than f_P and less than f_T, i.e. $f_P \le f \le f_T$. Based on the relation among f_P, f_T and f_E^*, the following 4 TPE management scenarios may happen.

1. $f_P > f_T$: In this scenario, the performance and temperature constraints could never be fulfilled.

2. $f_P \le f_T \le f_E^*$: In this scenario, selecting $f = f_T$ gives the minimum energy while satisfying the performance and temperature constraints.

3. $f_E^* \le f_P \le f_T$: In this scenario, selecting $f = f_P$ gives the minimum energy while satisfying the performance and temperature constraints.

4. $f_P \le f_E^* \le f_T$: In this scenario, selecting $f = f_E^*$ gives the minimum energy while satisfying the performance and temperature constraints.

The above analysis shows that the relative position of f_P, f_T and f_E^* determines how the optimal frequency will be selected. Such relation is specified by the user constraints. Furthermore, based on property 4~6 we know that the relative position of f_P, f_T and f_E^* will change as μ changes. This means that for different software applications and even for the same software application but at different time, different TPE management scenarios may occur. Finally, as different computing devices have different ρ and α, the same set of performance and temperature constraints and the same software program will result in different relative positions of f_P, f_T and f_E^* when running on different hardware systems, hence requires different TPE management policy. It is important to have an adaptive technique that automatically searches for the optimal clock frequency for TPE control when hardware and software changes.

3. Learning based temperature, performance and energy (TPE) management

3.1 General Q-learning algorithm

Q-learning [21] is one of the most widely used reinforcement learning algorithms. It consists of a state space S and a set of actions A. Selecting an action $a \in A$ at state $s \in S$, will lead the system to another state and result a reward (or penalty). A policy π is a mapping from the set of environment states to the set of actions, i.e. π: $\boldsymbol{S} \rightarrow \boldsymbol{A}$. For each state action pair (s, a), it maintains a value function $Q^{\pi}(s,a)$ 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.

The core of the Q-learning algorithm is the iterative update of the Q-value function. The Q-value for each state-action pair is initially chosen by the designer and later, it is updated each time an action is issued and a penalty is received, based on the following expression:

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{old\ value} + \underbrace{\varepsilon_t(s_t,a_t)}_{learning\ rate} \times \left[\overbrace{\underbrace{p_{t+1}}_{Penalty} + \underbrace{\gamma}_{Discount\ Factor} \underbrace{min_a\, Q(s_{t+1},a)}_{Min\ Future\ Value}}^{Expected\ Discounted\ Penalty} - \overbrace{Q(s_t,a_t)}^{Old\ Value}\right] \quad (8)$$

In the above expression, s_t, a_t and p_t are the state, action and penalty at time t respectively, and $\varepsilon_t(s,a) \in (0,1)$ is the learning rate. The discount factor γ is a value between 0 and 1 which gives more weight to the penalties in the near future than the far future. The next time when state s is visited again, the action with the minimum Q-value will be chosen, i.e. $\pi(s) = min_{a \in A}\, Q(s,a)$. The value of $Q(s_t,a_t)$ is updated at the beginning of cycle $t+1$, i.e., the Q-value for the state-action pair of the previous cycle is updated at the beginning of current cycle.

3.2 States and actions of TPE management agent

The environment state is a vector of four components, (f, T, IPS, μ), they represent the clock frequency, the temperature, the instructions per second (IPS) and the CPU intensiveness respectively.

Since our goal is to tune the processor frequency in order to find a balance among energy, performance and temperature, the current processor frequency f and temperature T must be selected as part of the state vector. Let N be the total number of clock frequencies supported by the processor, we use f_i to denote the ith clock frequency, with f_0 representing the minimum frequency. Similarly, we discretize the possible range of temperature into M levels, with T_0 representing the ambient temperature and T_{M-1} representing the maximum temperature threshold.

Although processor's power consumption is also a critical information pertaining to the environment state, this information is usually not easy to obtain for many of the commercial processors. However, it can be inferred from other system information, such as IPS (instructions per second), clock frequency, and the workload CPU intensiveness[22].

IPS (instructions per second) is an important value related to processor's performance, power and temperature [8]. It is recorded by the performance counter of many commercial processors. Even if they are running under the same clock frequency, different programs usually have significantly

different *IPS* due to events such as cache miss, branch prediction miss, etc. Therefore, the *IPS* does not fully represent the system performance, which in our definition, is the slowdown ratio of a system running at scaled frequency compared to the same system running at the maxim clock frequency. In the Q-learning algorithm, we use both *IPS* and clock frequency to infer the state of system performance.

Workload CPU intensiveness μ is another critical feature that classifies environment states. As we can see from Equations (4)(6)(7), it directly determines the relation between *T*, *P, E* and the normalized frequency.

The value of μ can be calculated as the following [1]:

$$\mu = 1 - \frac{cycles\_l1i\_stalled + cycles\_l1d\_stalled}{total\ number\ of\ cycles} \quad (9)$$

$cycles\_l1i\_stalled$ and $cycles\_l1d\_stalled$ are the number of cycles during which instruction and data fetches are stalled. They can be recorded periodically in many commercial processors. Though there are other architectural events that are related to μ, such as the cycle of stalls due to TLB miss, branch prediction miss and etc., they are less dominant and cannot be monitored at the same time with the cache miss events in our experiment system.

The value of *IPS* and μ will also be discretized in order to obtain a finite state space. In general, the more frequency levels supported by the hardware, the more tradeoff we can achieve and the finer should the state space be partitioned. This is attributed to the fact that executing the same program using different frequency levels generally will lead to different levels of die temperature and CPU *IPS*. Hence the more frequency levels supported by the hardware, the more detailed temperature and performance states should be divided.

The actions of our Q-learning controller are decisions switching to each frequency level supported.

3.3 Penalty function of learning based TPE controller

In order to satisfy different user requirements in TPE management, here we propose two working modes of the learning based TPE controller: *free mode* and *constrained mode*. The TPE controller in free mode is designed for users that are interested in exploring the tradeoffs among temperature, performance and energy, while the TPE controller in constrained mode is designed for users that have specific constraints in two of these three quality measurements and would like to optimize the third one. The main difference of these two modes is in the calculation of the cost function (p_{t+1}) of each state action pair, which will be introduced in the next.

Cost function in free mode

In this mode, in order to update the Q-value for the previous state-action pair $(Q(s_t, a_t))$, we calculate the penalty p_{t+1} (in Equation (8)) in Q-learning as follows :

$$pnlt = w_E * pnlt_E + w_P * pnlt_P + w_T * pnlt_T \quad (10)$$

$pnlt_E$, $pnlt_P$ and $pnlt_T$ are penalties associated to energy, performance and temperature respectively. Their definition will be given in the following paragraphs. w_E , w_P and w_T are the weight coefficients of the penalties which are exposed to the users. As we can see, $pnlt_E$, $pnlt_P$ and $pnlt_T$ are in different units and their values might not even be in the same order. It is very difficult to select the appropriate weight coefficients to achieve the desired tradeoffs. To conquer this problem, we normalize values of $pnlt_E$, $pnlt_P$ and $pnlt_T$ with respect to the largest energy, performance and temperature penalties that can be achieved. Hence the weight coefficients are confined to the range (0, 1). It is easy to know that, the minimum temperature, maximum performance, or minimum energy policy can be found when the weight coefficient vector (w_T, w_P, w_E) is set to (1, 0, 0), (0, 1, 0), or (0, 0, 1) respectively.

From the discussion in Section 2 we can see that the normalized energy of an application is a function of *f*, μ and ρ. For the given μ, ρ and α there is an optimal normalized clock frequency f_E^* that minimizes the energy. Deviating frequency from f_E^* will increase energy dissipation. Because ρ and α are fixed for a given computing hardware, the value of f_E^* only depends on the CPU intensiveness μ, which is an application specific parameter. We define the $pnlt_E$ as the difference between *f* (i.e. current normalized frequency) and f_E^*. For discrete DVFS system, f_E^* is rounded to the nearest frequency level supported by the system. Equation (11) gives the normalized energy penalty:

$$pnlt_E = \frac{|f - f_E^*|}{f_{\max} - f_{min}}, \quad (11)$$

where $f_{\max}$ and f_{min} are normalized maximum and minimum clock frequency.

Similar to energy, the performance is also a function of *f*, μ andρ. In general, if the CPU intensiveness μ is low, decreasing the CPU frequency will not degrade the performance a lot since more time will be spent on the memory subsystem whose frequency is fixed. We define the performance penalty as the extra execution time compared to the system running at the highest clock frequency, i.e. $(f_{max} - f)\mu$. It is normalized with the respect of the maximum possible execution delay, which happens when $f = f_{\min}$ and $\mu = 1$. The normalized performance penalty is calculated as:

$$pnlt_P = \frac{(f_{max} - f)}{(f_{max} - f_{min})} * \mu \quad (12)$$

We need to point out that, today's commercial computing system has architectural advances that enable out-of-order execution and minimize CPU stall. In all of our experiments, the performance penalty is almost equal to the ratio of frequency change.

Lastly, we use the change of the temperature as temperature penalty:

$$pnlt_T = \frac{T - T_{old}}{T\_range_{intervals}}. \quad (13)$$

If $T > T_{old}$, a positive temperature penalty will be given and if $T < T_{old}$, a negative temperature penalty, in other words, a reward will be given. $T\_range_{intervals}$ is the maximum temperature change in two adjacent time intervals. It is about 2°C in our experiment system.

Cost function for constrained mode

In the free mode, the users has the freedom to tune the weight coefficients w_E, w_P and w_T from 0 to 1 to explore the tradeoffs in energy, performance and temperature. In the constrained mode, there are user constraints on energy, performance or temperature which must be met. Previous work [15] tries to solve the performance constrained energy optimization problem by dynamically adjusting the weight coefficient in the penalty function to find minimum energy policy that exactly meet the performance constraint. The rationale of this approach is that energy is a decreasing function of performance. However, this is not true if we consider the leakage power. Furthermore, this approach will not work if we have more than one constraint. For example, as shown in Figure 1，assume that the minimum frequency that satisfies the performance constraint is f_P and the maximum frequency that satisfies the temperature constraint is f_T. Our goal is to constrain the performance and temperature, while at the same time minimizing the energy. As we can see, it is not possible to find a frequency that satisfies both performance and temperature constraints exactly. To make things worse, because we keeps increasing the value of w_P and w_T to ensure the satisfaction of performance and temperature constraints, the value of w_E will become relatively smaller and smaller compared to w_P and w_T. Hence, we may find a frequency/voltage setting that satisfies both performance/temperature constraints, but it is not guaranteed to be energy optimal (f=f_E^*).

In this work, we modify the penalty function to decouple the two constraints. To simplify the discussion, we use the average T, P, and E penalties of a system to constrain its temperature, performance and energy. The constraints are denoted as con_E, con_P, con_T. We use Δ_E, Δ_P, and Δ_T to denote the difference between the constraint and the actual average penalty during a history window for energy, performance and temperature respectively. The value of Δ will be positive if the system has been outperforming the user constraint during the history window, otherwise it will be negative. Because we are interested in constraining only the average performance (energy or temperature), we consider the system to be bounded when $pnlt_{P(E,T)} \le cont_{P(E,T)} + \Delta_{P(E,T)}$, otherwise, the system is unbounded. This basically says that if the system has been outperforming the user constraint during the past, then the penalty of the current cycle can be a little higher than the constraint.

We consider the general scenario that the user may set constraints on any two of the T, P and E and try to optimize the third one, and the problem can be written as:

$\min pnlt_3$, s.t.

$pnlt_1 \le con_1$, and $pnlt_2 \le con_2$.

Here the subscripts 1, 2, and 3 can be any permutation of the symbols T, P and E. We refer $pnlt_3$ as the *objective penalty* and $pnlt_1$ and $pnlt_2$ as *constrained penalties*.

The modified penalty function considers 4 scenarios:

$$pnlt = \begin{cases} pnlt_3 & if \ \le con_2 + \Delta_2 \,\&\, pnlt_1 \le con_1 + \Delta_1 \\ pnlt_3 + C \cdot pnlt_2 & if\ pnlt_2 > con_2 + \Delta_2 \ and\ pnlt_1 \le con_1 + \Delta_1 \\ pnlt_3 + C \cdot pnlt_1 & if\ pnlt_2 \le con_2 + \Delta_2 \ and\ pnlt_1 > con_1 + \Delta_1 \\ pnlt_3 + C \cdot pnlt_1 + C \cdot pnlt_2 & otherwise \end{cases} \quad (14)$$

In the above equation, C is a large constant. Based on the modified penalty function, when the system is bounded, the Q-learning algorithm will search for policies that minimize the objective penalty. As soon as the system becomes unbounded in one or both of the constrains, the penalty function will be modified and the Q-learning algorithm will search for policies that balances the objective penalty and constrained penalty (or penalties). During this procedure, it puts more emphasis on improving the constrained penalty.

We need to point out that, if the objective is to minimize the energy dissipation with the performance and temperature constraint, it can be proved that as long as the performance and temperature constraints are feasible, scenario 4 in the penalty function is not reachable. In other words, performance and temperature constraints will not be violated at the same time. However, if the constraints are in any other two metrics, all of the 4 scenarios are reachable.

Another point need to be mentioned is that the temperature penalty is calculated based on temperature changes as shown in Equation (13) rather than the absolute temperature. So when the temperature constraint is violated, the action leading to the reducing of the temperature will lead to a negative penalty (i.e. reward) to reinforce the corresponding action.

4. Experimental results

4.1 Experiment setup

We carried experiments on Dell Precision T3400 workstation with Intel Core 2 Duo E8400 Processor. The processor supports 4 frequency levels: 2GHz, 2.33GHZ, 2.67GHz, 3GHz. The Linux kernel we use is version 2.6.29.

We used *coretemp* driver in the Linux kernel to read the temperature sensor of the processors. The default driver updates temperature readings once every second and we modified it to be every 10ms to achieve our required granularity. We used *cpufreq* driver in Linux based on Enhanced SpeedStep[11] technology of Intel Core 2 processor to adjust the processor's frequency. We used Perform2 tool [13] to monitor performance events of the processors.

We used benchmarks from MiBench [17] and MediaBench [16] to form the workload of the evaluation system. Our goal is to generate workloads with changing CPU intensiveness. The benchmarks we selected are: bitcount_small, basicmath_small, qsort_large, tiff2rgba, mpeg4dec, and jpeg200dec together with a simple custom application with only CPU busy loops. Their CPU intensiveness varies from 11% to almost 100% with an average of 82% according to our measurement. Each benchmark running a little more than 0.2s under minimum frequency is a running unit. We serialized 100 running units of different benchmarks in 4 different random orders to construct 4 'workloads'. Every experiment result reported here is the average of the 4 'workloads'.

Since the proposed algorithm considers the TPE management of only one core, we ran the experiments on one core and fixed the frequency of the other core to be the minimum. The Q-learning controller was triggered every 20ms. Empirically, this interval will not exert too much overhead to the processor

while still capable of tracking the change of workload. The overhead of frequency change is only about 20us.

As mentioned in Section 3, we represent the environment state using a state vector (f, T, IPS, μ). Since the processor supports 4 frequency levels, f has 4 states. We further partition the CPU intensiveness μ into 4 states, so that f_i is corresponding to the ideal frequency f_E^* when $\mu = \mu_i$, $1 \le i \le 4$. Such partition enables us to measure the energy penalty using the deviations from the ideal frequency. Our goal is to evaluate our TPE controller's ability of tracking the CPU intensiveness for the energy reduction. The temperature and *IPS* are also empirically partitioned into 4 states as there are 4 frequency levels.

4.2 Results and analysis

4.2.1 Free mode TPE management

In the first set of experiments, we swept the value of w_E, w_P, and w_T from 0 to 1 and recorded the average temperature, performance and energy penalty. In order to compare the performance of the proposed learning algorithm with the state-of-art approach, we modified the expert-based algorithm in [1] based on our best understanding to achieve tradeoff among T, P and E. We also ran the Linux *Ondemand* governor for comparison. We swept the *up_threshold* of Ondemand governor [19] to achieve tradeoff between performance and energy.

Since we have three features, including energy, performance and temperature, two figures are presented in the next to show the relation between energy and performance (Figure 2) as well as the relation between temperature and performance (Figure 3).

In both figures, the X-axis gives the normalized execution time (performance) of the workloads. The Y-axis in Figure 2 gives normalized energy while the Y-axis in Figure 3 gives the normalized temperature. Our experiment data show that, the test case that has the best performance is 7.5 seconds faster than the test case that has the worst performance. In general, the former executes the entire workload at the highest clock frequency while the latter at the lowest frequency. In terms of the die temperature, the test case that has the lowest temperature is 7°C cooler than the test case that has the highest temperature. Because we cannot measure the real energy dissipation, we use Equation (11) to estimate the average energy penalty during the workload execution. Its value varies from 0.13 to 0.58 in best case and worst case respectively.

The results shown in Figure 2 and Figure 3 in general agree with our analysis in Section 2. When the execution time increases (i.e. the performance reduces), the energy will first decrease then increase. The minimum energy is achieved when the normalized execution time is around 0.4. We believe that this is approximately the average f_E^* of the testing workloads under the given hardware characteristics (i.e. ρ and α) and the workload's average CPU intensiveness (i.e. μ). We also need to point out that the TPE controller sometimes gives policy with higher energy and longer execution time compared to the energy optimal policy. These policies are corresponding to the data points located in the upper right corner in Figure 2. This is because, for those test cases, the weight coefficient of temperature is much larger than the weight coefficients of performance and energy. Therefore, the TPE controller slows down the performance to reduce temperature. As the clock frequency drops below f_E^*, the energy dissipation increases.

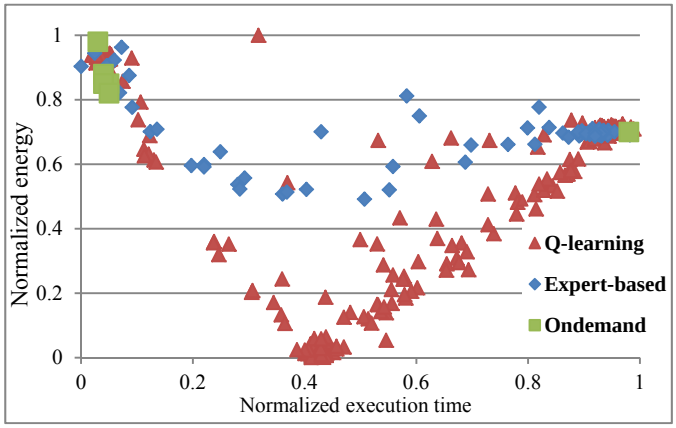

Figure 2 Energy versus performance tradeoffs

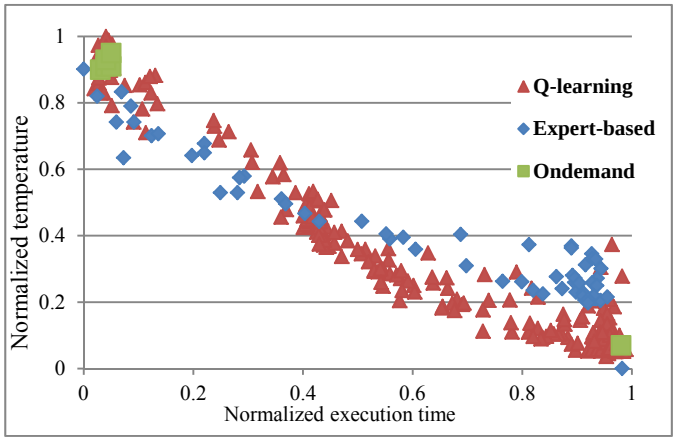

Figure 3 Temperature versus performance tradeoffs

As we can see from the figures, the proposed TPE controller generally achieves lower energy dissipation than the expert-based controller for the same performance level. And it performs better on the temperature control at low performance level.

The Linux default Ondemand governor can only achieve very limited tradeoffs. The rightmost point in Figure 2 and Figure 3 is achieved by setting *up_threshold* to *100*. This makes the processor running at the lowest frequency all the time. Setting *up_threshold* to any other values results in very similar energy performance tradeoffs. The corresponding energy delay points are located in the upper left corner in Figure 2 and Figure 3. This is because the Ondemand governor will provide the highest frequency as long as there are applications running, no matter what kind of application it is.

Figure 4, Figure 5 and Figure 6 show the how the temperature, delay and energy change when the corresponding weight coefficients vary from 0 to 1. In all experiments, the other two weight coefficients are both fixed to 0, 0.4 and 1 separately. The points when (w_T, w_P, w_E) equals to (0, 0, 0) are

ignored in the figure because those weight coefficient settings give random policies. As we can see, in general, increasing the T, P, and E weight coefficient, the learning algorithm will find policy with smaller energy, higher performance and lower temperature respectively. However, the actual performance, energy and temperature are determined by the relative value of the weight coefficients. For example, Figure 5 shows that increasing w_P from 0 to 1 with (w_T, w_E) setting to (0, 0) does not reduce the execution time because the relative value of w_P (compared to w_T and w_E) does not change. On the other hand, increasing w_P from 0 to 1 with (w_T, w_E) setting to (0.4, 0.4) can significantly reduce the execution time because now the relative value of w_P increases notably. However, if (w_T, w_E) is set to (1,1)，then the potential of performance improvement reduces again because with energy and temperature being the first optimization priority, there is much less flexibility for performance optimization. The same trend can be observed for the temperature and energy as shown in Figure 4 and Figure 6.

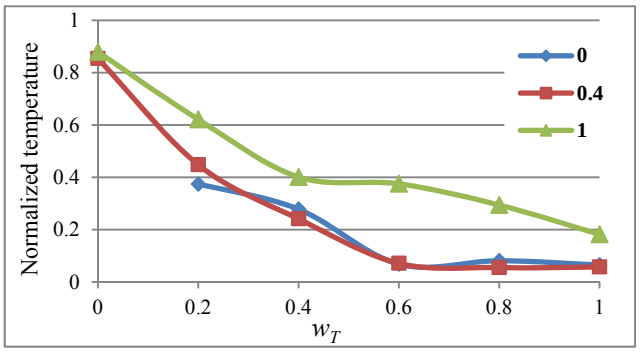

Figure 4. Controlling temperature using w_T

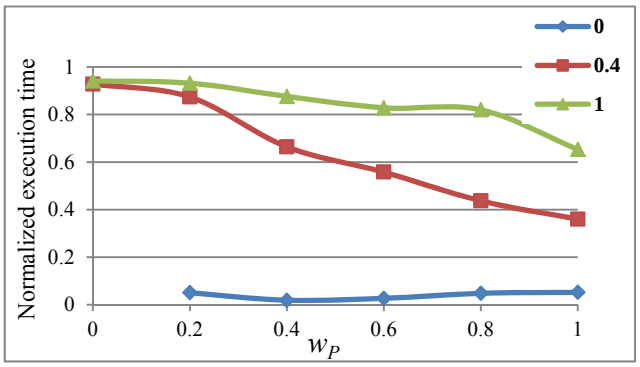

Figure 5. Controlling performance using w_P

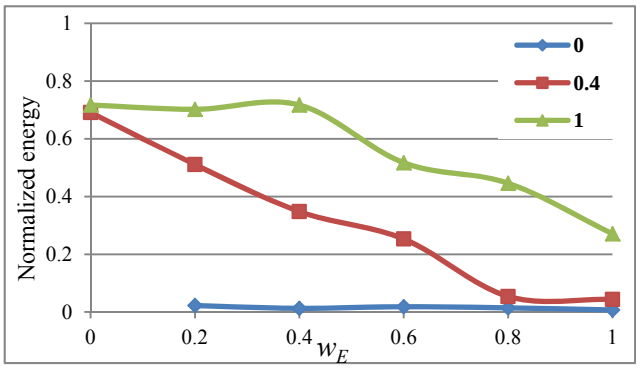

Figure 6. Controlling energy using w_E

4.2.2 Constrained mode TPE management

In the second set of experiments, we set two constraints among temperature, performance and energy, and try to optimize the third one.

Temperature \ Performance	0.34 (constraint)	0.67 (constraint)	1 (constraint)
0.34(constraint)	0.27 / 0.45	0.42 / 0.31	0.64 / 0.25
0.67(constraint)	0.24 / 0.48	0.37 / 0.42	0.61 / 0.31
1 (constraint)	0.26 / 0.49	0.38 / 0.44	0.60 / 0.35

Table 1 Constraining performance and temperature

Performance \ Energy	0.34 (constraint)	0.67 (constraint)	1 (constraint)
0.34(constraint)	0.41 / 0.30	0.40 / 0.33	0.41 / 0.30
0.67(constraint)	0.31 / 0.53	0.41 / 0.59	0.43 / 0.59
1 (constraint)	0.21 / 0.69	0.35 / 0.90	0.45 / 0.99

Table 2 Constraining energy and performance

Temperature \ Energy	0.34 (constraint)	0.67 (constraint)	1 (constraint)
0.34(constraint)	0.38 / 0.36	0.41 / 0.37	0.46 / 0.40
0.67(constraint)	0.39 / 0.43	0.42 / 0.46	0.51 / 0.55
1 (constraint)	0.37 / 0.44	0.42 / 0.49	0.56 / 0.65

Table 3 Constraining energy and temperature

The results are shown in Table 1,Table 2 and Table 3. In these tables, the performance is represented by average normalized performance penalty calculated based on Equation (12). Therefore, a large number corresponds to poor performance. The energy is the average energy penalty calculated based on Equation (11). The temperature is the average normalized temperature of the CPU. Each row and column in those tables associates to a user constraint in either temperature,

performance or energy. Because our platform only supports 4 frequency levels and the frequency increases linearly at an equal step from level 1 to level 4, the corresponding normalized temperature and performance for those frequency levels should also change at an equal step. To better show our results, we set the constraints to be 0.34, 0.67 and 1 as shown in the tables. Each cell in those tables has two fields. They give the actual achieved value in those two quality measurements with user constraints. For example, the cell in row 1 and column 1 of Table 1 shows that the actual normalized temperature and performance of the system is 0.45 and 0.27 respectively when the performance and temperature constraint are both set to 0.34.

Each cell is shaded differently according to the results of the third metric (i.e. the objective penalty). The lighter the cell is, the better the optimization we achieve. As we can see, in all 3 tables, the upper left cell has the darkest color because it corresponds to the most stringent user constraints and hence leaves almost no room for the optimization of the 3rd metrics. On the contrary, the bottom right cell has the lightest color because it corresponds to the most relaxed constraints.

We can see that sometimes TPE controller cannot find a policy that satisfies both user constraints. For example, the entries (1,1) in all 3 tables have constraint violations. Sometime, the TPE controller finds policies that exactly satisfies one of the constraints and outperforms the other. These cases include: entry (1, 2) in Table 1, entries (1, 2), (1, 3), (2, 1), (3, 1), (3, 2) and (3, 3) in Table 2. For the rest of times, the TPE controller finds policies that outperform both user constraints. This clearly shows that the relation among T, P and E are not monotonic. We cannot optimize one metric by setting the other (one or two) metrics exactly to the given user constraints. For example, consider cell (2,2) in Table 1. The user set a loose performance and temperature constraint (con_P=con_T=0.67) in order to optimize the energy. However the result shows that the policy that minimizes the energy actually does not have to work so slowly and will not generate so much heat. Clearly in this test case, we have $f_P \leq f_E^* \leq f_T$ which corresponds to the last category presented in Section 2. The experimental results also show that, generally without the prior knowledge of the values of ρ, α, and μ, (which correspond to the knowledge of hardware and software), our TPE controller can correctly learn the TPE tradeoff space and give effective control policies. The only information we need to know related to the hardware is the mapping of different workload CPU intensiveness to the ideal working frequency f_E^* for the energy optimization purpose. This requirement can be removed if there is a way to measure the processor's power online.

5. Conclusion

In this paper, we first discuss the impact of voltage and frequency scaling on processor's temperature, performance and energy. And then we present a Q-learning based controller to dynamically adjust the processor's clock frequency to get the desired tradeoff among temperature, performance and energy. The proposed TPE controller works at two modes. At the *free mode*, it controller allows the user to explore the tradeoff by tuning the weight coefficients in the penalty function. At the *constrained mode*, the controller allows the user to set constraints to two out of the three parameters in T, P and E, while optimizes the third one. The experimental results show that the proposed controller learns the temperature, performance and energy tradeoff space of the experiment system and performs effect control without priori knowledge of hardware and software.

6. References

[1] G.Dhiman and T.S.Rosing, "System-Level Power Management Using Online Learning," *TCAD*,vol28,pp.678-698, Apr.2009.

[2] P.Langen and B.Juurlink, "Leakage-Aware Multiprocessor Scheduling for Low Power*," IPDPS'06*, pp.60,2006

[3] ITRS: http://www.itrs.net/

[4] R.Jejurikar,C.Pereira and R.Gupta, "Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems," *DAC'04*, pp.275-280,2004

[5] K.Choi, R.Soma and M.Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ration of Off-chip Access to On-chip Computation Times," *TCAD'2004*, vol.24,pp.18-28,Dec.2004.

[6] A.Coskun, R.Strong, D.Tullsen and T.Rosing, "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessor," *SIGMETRICS '09*, pp.169-180,2009.

[7] Y.Tan, P.Malani,Q.Qiu and Q.Wu "Workload Prediction and Dynamic Voltage Scaling for MPEG Decoding," *ASP-DAC '06*,pp.911-916,2006

[8] Y.Ge and Q.Qiu, "Dynamic Thermal Management for Multimedia Application Using Machine Leaning," *DAC'11*, Jun.2011.

[9] P.Choudhary and D.Marculescu, "Power Management of Voltage/Frequency Island-Based System Using Hardware-Based Methods," *TVLSI*, vol.17, issue3, pp.427-438,2009

[10] Lenovo : http://www.lenovo.com/us/en/

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

[12] Y.Liu, R.Dick, L.Shang, H.Yang, "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," *DATE'07*, pp.1526-1531,2007

[13] Perfmon2: http://perfmon2.sourceforge.net/pfmon_userguide.html

[14] Y.Tan, W.Liu and Q.Qiu, "Adaptive Power Management Using Reinforcement Learning," *ICCAD'09*,pp461-467,2009

[15] W.Liu, Y.Tan, Q.Qiu, "Enhanced Q-learning Algorithm for Dynamic Power Management with Performance Constraint," *DATE'10*, pp602-605, 2010

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

[17] Mibench: http://www.eecs.umich.edu/mibench/

[18] "Intel® Core™2 Duo Processor E8000 and E7000 Series": http://download.intel.com/design/processor/datashts/318732.pdf

[19] V.Pallipadi, A.Starikovskiy, "The Ondemand Governor: Past, Present and Future," *Ottawa Linux Symposium*, 2006

[20] B.Lin,A.Mallik,P.Dinda,G.Memik, R.Dick,"User- and Process-Driven Dynamic Voltage and Frequency Scaling," *ISPASS'2009*,pp.11-22,Apr.2009

[21] Q-learning: http://en.wikipedia.org/wiki/Q-learning

[22] R.Joseph and M.Martonosi, "Run-time Power Estimation in High Performance Microprocessors", *ISLPED'01*,pp.135-140,2001