Throughput Maximization for Periodic Real-Time Systems Under the Maximal Temperature Constraint

Huang Huang, Gang Quan, Jeffrey Fan, *Meikang Qiu
Department of ECE, Florida International University, Miami, FL 33174
*Department of ECE, University of Kentucky, Lexington, KY 40506
E-mail: hhuan001, gaquan, fanj@fiu.edu, *mqiu@engr.uky.edu

Abstract

We study the problem on how to maximize the throughput for a periodic real-time system under the given peak temperature constraint. We assume that different tasks in our system may have different power and thermal characteristics. Two algorithms are presented in this paper. The first one is built upon processors that can be either in active or sleep mode. By judiciously selecting tasks with different thermal characteristics as well as alternating the processor active/sleep mode, our approach can improve the throughput upon the existing techniques by 21% in average. We further extend this approach for processors with dynamic voltage/frequency scaling (DVFS) capability. Our experiments show that an improvement of 24% can be achieved when compared with the existing methods.

Categories and Subject Descriptors

D.4.7[**Operating Systems**]: Organization and Design–*Real-time systems and embedded systems*

General Terms

Algorithms, Performance

Keywords

Thermal aware real-time scheduling, Dynamic voltage/frequency scaling, Task sequencing

1. Introduction

The continued scaling of semiconductor technology has resulted in the exponential increase of power density in the IC circuit, and as a result, the rapidly elevated temperature. High temperature not only increases the packaging/cooling cost, but also shortens the life span of the computing system and degrades its performance, robustness, and reliability [12, 15]. In addition, high chip temperature dramatically increases the leakage power, which is becoming a major component of the overall power consumption in the deep sub-micron domain. High power consumption leads to high temperature, and high temperature in turn increases the leakage power and thus the overall power consumption. Evidently thermal awareness is becoming a more and more critical issue and researchers have incorporated the thermal issues into almost every abstraction level in the design of electronic computing systems [13].

We are interested in the problem of employing the scheduling technique to maximize the throughput for a real-time system under the peak temperature constraint. There are a few existing works (e.g. [8, 3, 6, 16, 17]) that are closely related to our research. Specifically, Liu and Qiu [8] proposed a hybrid approach based on job sequencing and DVFS for nondeterministic applications. 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. Zhang and Chatha [16] presented a pseudo-polynomial time speed assigning algorithm based on the dynamic programming approach followed by a scheme to minimize the total execution latency. To guarantee the peak temperature constraint, this approach requires the the ending temperature of each iteration not to exceed the starting temperature which can be very pessimistic. In addition, the two approaches above assume that all tasks have the same power characteristics, i.e. they consume the same amount of power as long as they run at the same processor speed, which might not be true in practice. As shown in [6], the power and thus the thermal characteristics for different real-time tasks can be significantly different. With this fact in mind, Jayaseelan and Mitra [6] proposed to construct the task execution sequence to minimize the peak temperature. Zhang and Chatha [17] further developed several new algorithms to maximize the throughput of a real-time system by sequencing the task executions for processors with and without dynamic voltage/frequency scaling (DVFS) capability.

In this paper, we propose two novel approaches to maximize the throughput for a periodic real-time system under the given peak temperature constraint, one for processor with simple active and sleep mode and the other for more complicated processors with DVFS capability. There are several distinct differences between our approaches and the existing works. First, while several existing techniques (e.g. [3, 6, 17]) take the leakage/temperature dependency into consideration, they assume that the leakage changes only with temperature. In fact, leakage power changes not only with temperature but also supply voltage [7]. As evidenced in [5], the leakage model ignoring the effect of supply voltage can lead to results deviated far away from the actual values and thus produce potentially inaccurate solutions. Second, both the existing approaches [6] and [17] assume that the task sequencing and the processor mode change can only occur at the task boundary. In contrast, we employ the same principle as implied in the m-oscillating approach [4] to sequence tasks and change processor modes within the task execution. Our experimental results, based on parameters drawn from the 65nm technology, show that our methods consistently outperform the existing approaches [17] by over 21% in average.

The rest of this paper is organized as follows. Section 2 introduces the system models followed by the motivational examples in Section 3. Our proposed scheduling algorithms are discussed in Section 4. Experimental results are presented in Section 5 and Section 6 concludes this paper.

2. Preliminaries

In this section, we briefly introduce the system models used in this paper, including the task model, processor and its thermal/power model, followed by the problem definition at the end.

Thermal Model: The thermal model used in this paper is similar to the one that has been used in the similar researches (e.g. [8, 4, 17, 3]) that $RC\frac{dT(t)}{dt} = RP(t) + (T(t) - T_{amb})$, where $T(t)$ and T_{amb} are the chip temperature and ambient temperature respectively. $P(t)$ denotes the power consumption at time t, and R, C are the chip thermal resistance and thermal capacitance respectively. We can scale T such that T_{amb} is zero and then we have $\frac{dT(t)}{dt} = aP(t) - bT(t)$, where $a = 1/C$ and $b = 1/RC$.

The Processor: We assume that the processor has one *sleep* mode and N *active* modes. Each *active* mode is characterized by a supply voltage/frequency pair (v_k, f_k). A task can only be executed when the processor operates in *active* mode. We assume that the processor can be switched from one mode to another at any time. However, such a switching will cause a timing penalty of t_{sw} during which no computation can take place.

Real-Time Tasks: The task model considered in the paper is a periodic task set consisting of independent heterogeneous tasks. The heterogeneous nature of the tasks are manifested in the fact that the power consumptions of different tasks vary significantly even running under the same speed level and at the same temperature. This is because the power consumptions are strongly depending on the circuit activity factor [10] and the usage pattern of different functional units when executing different tasks. Specifically, we introduce a parameter μ, called *activity factor*, to capture the different switching factors of different tasks. For a given task, the activity factor μ (ranging in $(0,1]$) defines how intensively the functional units have been used. For common benchmark tasks, the activity factors can be obtained by using any architectural level power analysis tools such as *Wattch* [2]).

Power Model: With the activity factor, the dynamic power consumption of the processor when executing task τ_i at the kth speed levels can be formulated as $P_{dyn}(i,k) = \mu_i C_2 v_k^3$, where v_k is the supply voltage level, C_2 is a constant. As leakage current changes super linearly with temperature [9], the leakage power of the processor when executing the task τ_i can be effectively estimated as $P_{leak}(i,k) = \mu_i(C_0(k)v_k + C_1(k)Tv_k)$, where $C_0(k)$ and $C_1(k)$ are curve fitting constants. Therefore, the overall power consumption when executing task τ_i at the kth speed level can be modeled as

$$P(i,k) = \mu_i(C_0(k)v_k + C_1(k)\cdot Tv_k + C_2 v_k^3). \tag{1}$$

Accordingly, the temperature dynamics when executing task τ_i can be formulated as

$$\frac{dT(t)}{dt} = A(i,k) - B(i,k)T(t) \tag{2}$$

where $A(i,k) = a\mu_i(C_0(k)v_k + C_2 v_k^3)$ and $B(i,k) = b - a\mu_i C_1(k)v_k$ (we use B_s for $B(sleep)$). Hence, for a given time interval $[t_0, t_e]$, if the initial temperature is T_0, by solving equation (2), the ending temperature after executing task τ_i can be formulated as below:

$$\begin{aligned} T_e &= \frac{A(i,k)}{B(i,k)} + (T_0 - \frac{A(i,k)}{B(i,k)})e^{-B(i,k)(t_e - t_0)} \\ &= T_{ss}(i,k) + (T_0 - T_{ss}(i,k))e^{-B(i,k)(t_e-t_0)}. \end{aligned} \tag{3}$$

where $T_{ss}(i,k)$ is the steady state temperature of the task τ_i at the kth speed level. For a given task, if $T_{ss}(i,k) > T_{max}$ (the maximal temperature limit), we call it a *hot task*, or *cool task* otherwise. Apparently, for the sleep mode, we have $T_{ss} = T_{amb}$.

Based on the models introduced above, the throughput of a real-time system can be maximized when the latency for executing the task in one period is minimized. Our research problem can be formulated as follows.

PROBLEM 1. *Given a tasks set $\Gamma = \{\tau_1(t_1,\mu_1), \tau_2(t_2,\mu_2), ... ,\tau_n(t_n,\mu_n)\}$, where t_i and μ_i are the execution time and activity factor of task τ_i respectively, develop a feasible schedule such that the latency to execute one iteration of Γ is minimized under the thermal steady state while ensuring the satisfaction of the peak temperature constraint T_{max}.*

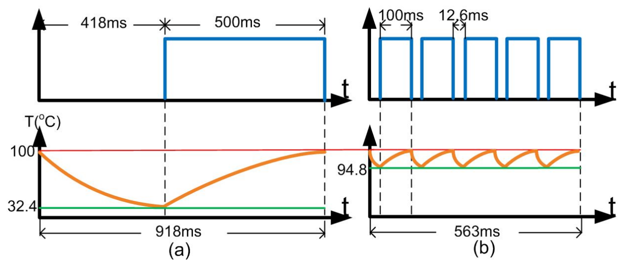


Figure 1. Motivation: Reduce Latency by Sleep Time Distribution

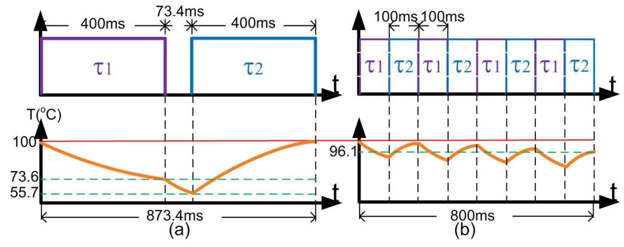


Figure 2. Motivation: Reduce Latency by Task Switching

3. Motivational Examples

Consider a task τ with execution time t of $500ms$ and steady state temperature T_{ss} of 115^oC. Let $T_{max} = 100^oC$. Assume that the processor has already reached this threshold before task τ starts to run. Then the temperature constraint will definitely be violated if the execution of τ starts immediately.

To avoid temperature exceeding T_{max}, we can turn the processor to *sleep* mode and let the processor cool down, as illustrated in Figure 1(a). Based on a processor and power model detailed later in Section 5, the processor has to stay in the *sleep* mode for $418ms$ to make sure its temperature dropped to a *safe* temperature. With this safe temperature, if we continue to execute τ, the peak temperature constraint, i.e. T_{max}, will not be violated.

Alternatively, as shown is Figure 1(b), we can divide the execution of τ equally into 5 sections and distribute the *sleep* mode before each of them. In this case, only $12.6ms$ is required for the processor to stay in the *sleep* mode to cool down to the *safe* temperature (e.g.94.8^oC) for each sub-interval. Consequently, the processor only needs to spend a total of $12.6 \times 5 = 63ms$ in the *sleep* mode to ensure the maximal temperature constraint. Shorter idle time implies that the latency of the tasks can be reduced and therefore improving the system throughput. This example indicates that it is more effectively to insert multiple idle intervals *inside* the task than only at the task boundary to improve the throughput under the peak temperature constraint.

Now consider another example as shown in Figure 2 with a *cool* task τ_1 ($T_{ss} = 68^oC$) and a *hot* task τ_2 ($T_{ss} = 115^oC$). Assume both tasks have the same execution time ($400ms$) at its peak performance. Similarly, we still assume that both the initial temperature and the peak temperature constraint are set as 100^oC. To reduce the execution latency without violating the peak temperature constraint, one approach is to run the *cool* task τ_1 first followed by the *hot* task. However, in this example, running the *cool* task alone cannot bring the temperature low enough such that τ_2 can immediately start to run without exceeding the peak temperature. Therefore, a sleep period of $73.4ms$ has to be inserted before τ_2 to further cool down the processor which results in a total latency of $873.4ms$ as shown in Figure 2(a).

In contrast, another approach is to divide the execution of τ_1 and τ_2 into 4 sub-intervals and run them alternatively. The schedule and the corresponding temperature curve are shown in Figure 2(b). Note that, both τ_1 and τ_2 are successfully executed under the peak temperature without inserting any idle interval at all. Moreover, the temperature at the end of the execution is reduced to 96.1^oC . This saved temperature budget (i.e.3.9^oC) could be utilized to further improve the throughput [17] if the processor has a more complicated DVFS mechanism.

The two examples clearly indicate that, by splitting tasks with different power/thermal characteristics into multiple sections and execute them alternatively, the throughput can be significantly improved. Several questions immediately rise. First, how effective this approach can be, especially when considering the switching overhead for alternating the task executions? Second, how can we choose the optimal section that a task needs to be split to achieve the best performance, i.e. throughput? We answer these questions in the next section.

4. Our Approach

In this section, we discuss our approach in detail and present our scheduling algorithms. We first consider the processor with only one *active* mode, i.e. $N = 1$ and assume all tasks are *hot* tasks with respect to the given peak temperature constraint. We then consider the task set consisting of both *hot* and *cool* tasks. Finally, we introduce our approach for processor with multiple *active* modes, i.e. $N > 1$.

4.1 Sleep Mode Distribution for Hot Tasks

We begin our discussion by assuming that the processor has only one *active* mode and one *sleep* mode. We further assume that all tasks in Γ have $T_{ss} > T_{max}$ (i.e. "*hot*" tasks). Since Γ is periodic and it is shown [3, 17] that the throughput of Γ is maximized when the temperature at the end of each period equals T_{max}, we can conveniently make the initial temperature of Γ to be the same as the ending temperature of each iteration, and set them to be T_{max}.

Since all tasks are *hot* tasks, starting at T_{max}, we can only bring down the temperature by inserting idle intervals. The question is how long we should insert the interval. The shorter the total idle interval, the smaller the overall latency and thus the larger the throughput. To quantify the effectiveness of different choices, we use a metric called the *idle ratio* (Θ) which is defined as the ratio between the time that the processor stays in *sleep* mode and the *active* mode within one period. It is not difficult to see that the smaller the Θ, the larger the throughput.

From the first motivational example above, we can see that the length of overall idle interval can be reduced by splitting each task into multiple—i.e. $m(m > 1)$—sections, and inserting the idle interval in between. In fact, we found that, when the switching overhead is negligible, the larger the m is, the smaller the overall idle interval time is needed. The observation is formulated in the following theorem.

THEOREM 1. *Given a task τ_i, a processor with only one active and one sleep mode, the maximal temperature constraint T_{max}, assume that $T_{ss}(i) > T_{max}$. Let $\Theta(m)$ represent the idle ratio for the feasible schedule when τ_i is evenly split into m sections. Then the idle ratio Θ is a monotonically decreasing function of m.*

Proof: Assume that when $m = 1$, to cool down the processor, t_s seconds of sleep period has to be added before τ_i. To find t_s,

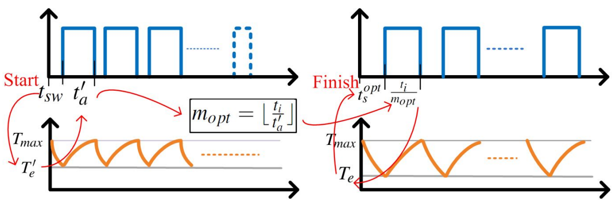


Figure 3. Sleep Time Distribution

we first need to find the *safe* temperature $T_{safe}(i)$ which can be calculated as

$$T_{safe}(i) = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}, \tag{4}$$

Thus, we have

$$t_s = -\frac{1}{B_s} \cdot \ln(\frac{T_{safe}(i)}{T_{max}}). \tag{5}$$

Therefore

$$\Theta(1) = -B_s t_i \cdot \ln \frac{T_{ss}(i)(e^{-B(i)t_i} - 1) - T_{max}}{T_{max} e^{-B(i)t_i}}. \tag{6}$$

Replace t_i with t_i/m in equation (6), we can formulate the idle ratio Θ as a function of m as

$$\Theta(m) = -B_s \frac{t_i}{m} \cdot \ln \frac{\hat{K}_2}{\hat{K}_1}, \tag{7}$$

where $\hat{K}_1 = T_{max} e^{-\frac{B(i)t_i}{m}}$ and $\hat{K}_2 = T_{ss}(i)(e^{-\frac{B(i)t_i}{m}} - 1) - T_{max}$. Since both $\Theta(m)$ and $\frac{B_s t_i}{m}$ are positive numbers, we have $\hat{K}_1 > \hat{K}_2 > 0$.

With the above result, it is not difficult to show that the first order derivative of $\Theta(m)$, i.e. $\frac{d\Theta(m)}{dm} < 0$. Therefore $\Theta(m)$ monotonically decreases with m. $\square$

Theorem 1 implies that the smaller we divide a task, the shorter the idle interval is needed. However, since

$$\lim_{m->\inf} \Theta(m) = \frac{B(i,k)}{B_s} \frac{T_{ss}(i,k) - T_{max}}{T_{max}}, \tag{8}$$

the impact of dividing the task becomes saturated as m increases. Furthermore, as $m \to \inf$, the context switching overhead cannot be ignored anymore, no matter how smaller it can be. The question then becomes how to determine the optimal "m" for each *hot* task.

Assuming the overhead for each context switching is t_{sw}, we develop a heuristic to search for the optimal value of m as illustrated in Figure 3. Starting from T_{max}, the corresponding ending temperature of the sleep period can be obtained from equation (3),

$$T'_e = T_{amb} + (T_{max} - T_{amb})e^{-B_s t_{sw}}. \tag{9}$$

Based on T'_e, the duration of the subsequent active mode is thus

$$t'_a = -\frac{1}{B(i)} \ln(\frac{T_{max} - T_{ss}(i)}{T'_e - T_{ss}(i)}). \tag{10}$$

Accordingly, we have $m_{opt} = \lfloor \frac{t_i}{t'_a} \rfloor$ (using floor function to make sure m_{opt} is an integer). Once m_{opt} is available, the duration of each active sub-interval can be readily determined, i.e. $\frac{t_i}{m_{opt}}$. The ending temperature of the corresponding sleep period can also be obtained as

$$T_e = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)\frac{t_i}{m_{opt}}}} \tag{11}$$

Finally , the minimized sleep time per sub-interval t_s^{opt} can be solved from equation (5) by replacing $T_{safe}(i)$ with T_e in equation (11)

$$t_s^{opt} = -\frac{1}{B_s} \ln(\frac{T_{ss}(i)(e^{-B(i)\frac{t_i}{m_{opt}}} - 1)}{e^{-B(i)\frac{t_i}{m_{opt}}}} - 1) \tag{12}$$

Although the sleep distribution method is targeted for single task, it is applicable to a task set consisting of multiple hot tasks, since the optimization procedure can be conducted on each individual task separately.

4.2 Improving Throughput by Task Switching

In this subsection, we extend our discussion to the task set consisting of both *hot* and *cool* tasks. Consider only two tasks, one *hot* task and one *cool* task for a given T_{max}. Recall that as implied by the second motivational example, dividing both tasks into $m(m > 1)$ sections, and alternating the execution of both tasks helps to improve the throughput. In fact, a similar theorem as Theorem 1 can be established for a task set consisting both *hot* and *cool* tasks as follows.

THEOREM 2. *Let $\Gamma = \{\tau_i(t_i,\mu_i), \tau_j(t_j,\mu_j)\}$ with τ_i, a cool task (i.e. $T_{ss}(i) \leq T_{max}$) and τ_j, a hot task (i.e. $T_{ss}(j) \geq T_{max}$). Assume both τ_i and τ_j are equally divided into m sections and alternatively executed (with τ_i first). Let $T'_j(i,j,m)$ represent the temperature when completing the first subsection of τ_j. Then $T'_j(i,j,m)$ monotonically decreases with m.*

Proof: Given the initial temperature T_{max}, based on equation (3), the ending temperature of task τ_i after $\frac{i}{m}$ seconds of execution can be expressed as

$$T'_i(i,j,m) = T_{ss}(i) + (T_{max} - T_{ss}(i))e^{-B(i)\frac{t_i}{m}}. \tag{13}$$

Similarly, the ending temperature of task τ_j after $\frac{j}{m}$ seconds can be formulated as

$$T'_j(i,j,m) = T_{ss}(j) + T'_i(i,j,m) - T_{ss}(j))e^{-B(j)\frac{t_j}{m}},. \tag{14}$$

After replace $T'_i(i,j,m)$ using equation (13), we have

$$T'_j(i,j,m) = T_{ini}\hat{K}_3 + T_{ss}(i,k)(e^{-B(j)\frac{t_j}{m}} - \hat{K}_3) + T_{ss}(j,k)(1 - e^{-B(j)\frac{t_j}{m}}) \tag{15}$$

where $\hat{K}_3 = e^{-B(i)\frac{t_i}{m} - B(j)\frac{t_j}{m}}$ and T_{ini} is the starting temperature of the schedule which is T_{max} in this case. Then, we can prove the conclusion by showing that $\frac{dT'_j(i,j,m)}{dm} < 0$. □

Now the problem becomes how to judiciously choose the appropriate m to maximize the throughput, especially under the assumption that the context switching overhead is not negligible. Theorem 2 implies that given a *cool/hot* tasks pair, one can always try to find a feasible schedule by increasing m as it always reduces the $T'_j(i,j,m)$. Thus, the optimal m can be found by sequentially search from 1 to +inf (among positive integers). The searching is stopped when we have $T'_j(i,j,m) \leq T_{max}$. However, the total amount of switching time associated with m task switching is $m \cdot t_{sw}$. If the original sleep period is t_{cu}, we must have $m \cdot t_{sw} < t_{cu}$ in order to further improve the latency. Therefore, we set the upper bound for m that $m_{max} = \lfloor \frac{t_{cu}}{t_{sw}} \rfloor$. If the ending temperature at $m = m_{max}$ still cannot meet the temperature constraint, the task switching scheme fails. Thus, we first test if $T'_j(i,j,m_{max}) < T_{max}$, the searching process will only start if the result is true. Our simulation results show that $T'_j(i,j,m)$ decreases drastically at the first several steps (small m), usually the optimal m can be obtained within 5 iterations.

We are now ready to present our algorithm, i.e. Algorithm 1, for a task set consisting of more than two tasks. Our algorithm works as follows: First, the tasks are classified into *cool* tasks and *hot* tasks based on their power/thermal characteristics and the given peak temperature constraint. Then, we put *cool* tasks in a queue Q_c and sort them in the increasing order of their ending temperature (assuming the execution starts at temperature T_{max}). The *hot* tasks are put in Q_h and sorted in the increasing order of their *safe* temperatures. The output of the algorithm is the final schedule Q_f. For instance, the Nth element in Q_f is denoted as $Q_f(N) = \{\tau_i, \tau_j, m\}$

Algorithm 1 Improving Throughput by Task Switching

```
 1: Input: $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$, $T_{max}$
 2: Output: $Q_f$; (the optimal task paring and the associate number of switching)
 3: Initialization: classify all tasks into cool/hot tasks based on $T_{ss}$
 4: Find $T_{safe}$ for all hot tasks
 5: Sort cool tasks into $Q_c$ in increasing order of $T_{end}$
 6: Sort hot tasks into $Q_h$ in increasing order of $T_{safe}$
 7: $T_{ini} = T_{max}$
 8: $Q_f = empty$
 9: for i=1:length($Q_c$)
10:   for j=1:length($Q_h$)
11:     Find the upper bound of cooling time $t_{cu}$ from eq.5
12:     Find the upper bound of $m$: $m_{max} = \lfloor \frac{t_{cu}}{t_{sw}} \rfloor$
13:       if $(T'_j(i,j,m_{max})) \leq T_{max}$
14:         Find $m_{opt}$ by sequential search from $m = 1$
15:         Move $\{Q_c(i), Q_h(j), m_{opt}\}$ to the end of $Q_f$
16:         Update initial temperature: $T_{ini} = T_{end}(i,j,m_{opt})$
17:       endif
18:   endfor
19: endfor
20: if ($Q_c$ is not empty)
21:   Move tasks in $Q_c$ to the end of $Q_f$ by decreasing order of $T_{ss}$
22:   Update the initial temperature
23: if ($Q_h$ is not empty)
24:   Move tasks in $Q_h$ to the end of $Q_f$ by sleep distribution
25: Return: $Q_f$
```

which specify the index of the two tasks being paired (τ_i and τ_j) as well as the optimal number of task switching (m).

Starting from the initial temperature T_{max}, the task at the beginning of Q_h attempts to pair with the head task in Q_c. If this task pair is feasible, the m_{opt} can be obtained by a sequential search. Then this task switching schedule specified by tuple $\{Q_c(i), Q_h(j), m_{opt}\}$ is added to the beginning of Q_f. After that, the ending temperature of $\{Q_c(i), Q_h(j), m_{opt}\}$ needs to be set as the initial temperature of the next attempted task pairing.

To find the ending temperature, we derived a closed-form formula based on [11], that if there exists m times of task switching between τ_i and τ_j, the ending temperature of the schedule can be formulated as $T_{end}(i,j,m) = T_{ini} + \frac{(T'_j(i,j,m) - T_{ini}) \cdot (1-\hat{K}_4)}{1-\hat{K}_3}$, where $\hat{K}_4 = e^{-B(i)t_i - B(j)t_j}$.

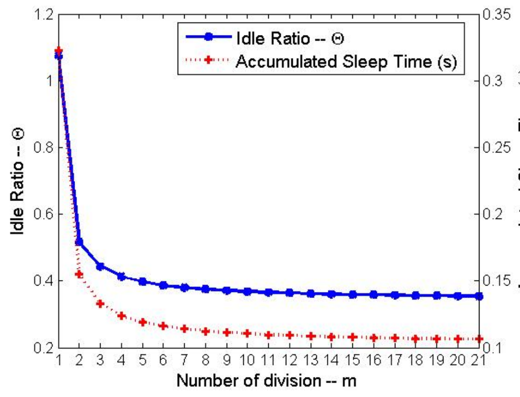
Based on the above equation, the ending temperature after m_{opt} time task switching can be obtained by replacing m with m_{opt}

However, if the task pairing fails, it is still possible to make a feasible combination with jth ($i \neq j$) *cool* task. Because the temperature profiles depend not only on the T_{ss} of the two task but also on the duration of the tasks. Therefore, the *hot* task is left in the Q_h until the end of the iteration to get chance to be matched with all *cool* tasks. Finally, after the double-loop task pairing, if there are still tasks left in Q_c/Q_h, the cool tasks are simply attached to the end of Q_f followed by the hot tasks executed with sleep time distribution method introduced in Section 4.1.

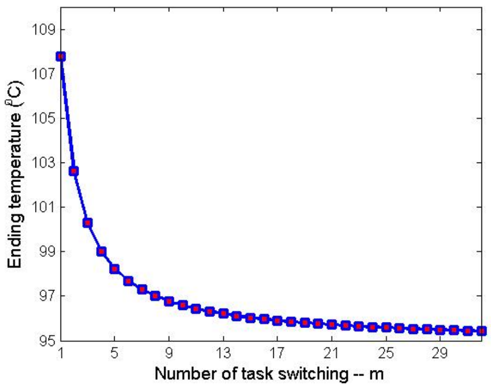
4.3 Improving Throughput by DVFS

In the previous discussion, we assume that the processor has only one *active* mode. In this subsection, we adopt a more complex processor model, i.e. the processor with $N(N > 1)$ different *active* modes, and the processor can change its working mode dynamically. Employing DVFS is a double-edged sword in terms of throughput maximization. On one hand, reducing the supply voltage slows down the task execution and reduces the throughput. On the other hand, reducing the supply voltage helps to reduce the power consumption and thus the thermal pressure. How to make an appropriate tradeoff needs careful analysis.

Consider the pros and cons of DVFS in throughput maximization, we utilize DVFS only under certain scenario. Specifically,



(a) Latency Reduction by Increasing m



(b) Ending Temperature Reduction by Task Switching

Figure 4. Theorem Validation

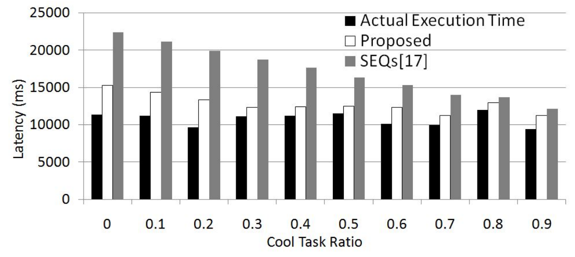
for a give task set, we first employ Algorithm 1, assuming the processor work at its highest speed. DVFS mechanism is only used when there are hot tasks left in the Q_h, and we try to scale down the supply voltages of those tasks.

Given a *hot* task τ_i, if the execution time at the highest speed level k is $t_i(k)$, then the corresponding execution time at $(k-l)$th mode is calculated by $t_i(k-l) = \frac{f_k}{f_{k-l}} t_i(k)$, where f_k and f_{k-l} are the working frequencies associated with V_{dd} level k and $k-l$ respectively. From equation (5), the required sleep time of task τ_i under supply voltage levels $k-l$ can be obtained and expressed as

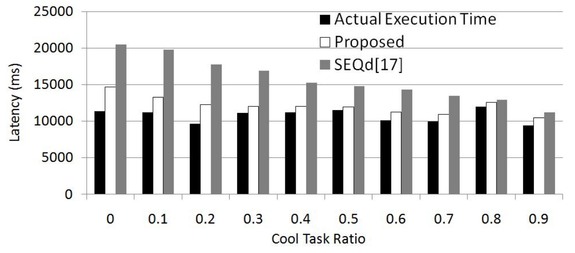
$$t_s(i,k-l) = -\frac{1}{B_s} \ln\left(\frac{T_{ss}(i,k-l)(e^{-B(i,k-l)\frac{f_k}{f_{k-l}}t_i(k)} - 1)}{e^{-B(i,k-l)\frac{f_k}{f_{k-l}}t_i(k)}} - 1\right) \quad (16)$$

then the total latency of τ_i at speed level $k-l$ is formulated as $t(i,k-l) = t_s(i,k-l) + \frac{f_k}{f_{k-l}} t_i(k)$. Here we calculate the latency for a given task under all supply voltage levels by increasing l from 1 to $k-1$. And the speed level that leads to the minimized latency will be selected as the optimal speed k_{opt}.

After we find the k_{opt} for all *hot* tasks left in Q_h, the pairing routine (line 9-19 in algorithm 1) is called again for a final round of attempted pairing between tasks in Q_h and Q_c. Now, the *hot* tasks (if speed is scaled) might have a better chance to be combined with the tasks left in Q_c (if not empty) due to the reduced steady state temperature. Finally, if Q_h and Q_c are still not empty, line 20-24 in Algorithm 1 is executed to attach these tasks at the end of Q_f with minimum sleep time. The detailed algorithm is omitted due to page limit.



(a) Processor without DVFS



(b) Processor with DVFS

Figure 5. Simulation Results: Latency Comparisons

5. Experimental Results

In this section, we validate the theorems and show the performance of the proposed approach through a set of simulations. We adopted the processor model similar to the one in [11]. The mode switching overhead is assumed to be 5ms [14]. The thermal model is obtained from [1]. We let the ambient temperature and the maximal temperature limit to be 25^oC and 100^oC [17] respectively unless otherwise specified.

5.1 Theorem Validation

To validate the conclusions made in Theorem 1, we run a hot task τ_1 ($t_i = 300ms$, $T_{ss} = 138^oC$ @$V_{dd} = 1.2v$). The idle ratio Θ as well as the total sleep time is plotted in Figure 4(a) while we increase m from 1 to m_{max}. The result conforms to the conclusion in Theorem 1 that Θ monotonically decreases with m. When $m = 1$ the schedule is identical to the one proposed in [17] that $322ms$ sleep time is required to cool down the processor before τ_1 starts to running (thus $\Theta = 1.07$). As m increases, the total sleep time and thus the idle ratio are minimized when $m = m_{max} = 21$. The final latency is $310ms$ compared with $622ms$ by the approach in [17], a 50% reduction is achieved.

We next validate Theorem 2 by running a *cool* task τ_2 ($T_{ss} = 77^oC$) followed by a *hot* task τ_3 ($T_{ss} = 112^oC$) without introducing the sleep interval in between. The execution time of both tasks are $500ms$. By using the proposed task switching method, the ending temperature of the first hot task sub-interval is plotted in Figure 4(b) as m increases from 1 to m_{max}. Again, the results agree with Theorem 2 that $T_j'(i,j,m)$ is a monotonically decreasing with m. From Figure 4(b) we can also see that $T_j'(i,j,m)$ drops drastically when $m < 5$ and becomes relatively stable when m is further increased. It implies that, in this particular case, m_{opt} can be found within 4 rounds of evaluation (since $T_{max} = 100^oC$ and $T_j'(i,j,4) < 100^oC$).

5.2 Latency Minimization

In this subsection, we evaluate the performance of the proposed method in terms of latency minimization. We created 10 representative task sets each of which is composed of 20 randomly generated tasks with execution time (at the highest speed level) and utility factor μ uniformly distributed within $[100, 1000ms]$ and $[0.4, 1]$ respectively. Based on our thermal model, the steady state temperature of these tasks are ranging in between 62^oC and 145^oC. For each task set, we specify the ratio of the number of the cool tasks versus the total number of tasks (i.e. 20), and vary this ratio from 0 to 0.9 with 0.1 increment.

For processor without DVFS, we compare our approach with SEQ_s proposed in [17]. In this case, we set the processor to the highest available speed level. We recorded the total latency achieved by both methods and plotted in Figure 5(a). We also provide the actual time that the processor spends in task execution as references. As shown in Figure 5(a) the proposed method consistently outperforms SEQ_s in all 10 task sets. Compared with SEQ_s, on average our method improves the latency by 24% (up to 35%) and reduces the idle time by 64% (up to 79%). For DVFS enabled processor, the same task sets are scheduled by the proposed approach and SEQ_d [17]. As shown in Figure 5(a), clearly, the proposed method still achieves better performance. Specifically, compared with SEQ_d, our method reduces the latency and idle time by on average 21% and 69% respectively.

From Figure 5(a) and 5(b) we also notice that the performance of the proposed approach is relatively stable compared with SEQ_s/ SEQ_d. This is because the proposed approach can rely on sleep time distribution to reduce the latency even without *cool* tasks

5.3 Feasibility Improvement

We next investigate the performance of the proposed method in terms of feasibility improvement for a processor without DVFS. Recall that a task is defined as infeasible if the required *safe* temperature is below ambient temperature. We used the same parameters to randomly generate task sets without specifying the number of cool tasks in each task set, instead, we set the ambient temperature as the variable and vary it from 25^oC to 65^oC with the step size of 10^oC. Under each ambient temperature condition, we generated 10 task sets each including 100 tasks. The averaged feasibility ratio (number of feasible task divided by 100) achieved by the proposed method and SEQ_s are recorded and plotted in Figure 6. The proposed method completes all tasks without infeasible task and thus improves the feasibility ratio by 21% on average compared with SEQ_s.

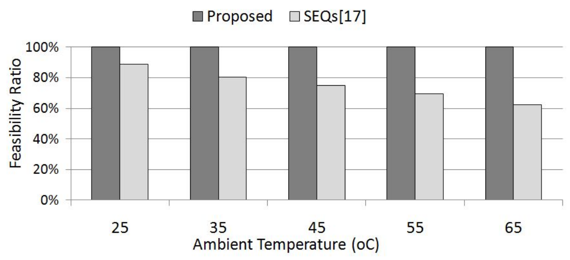


Figure 6. Simulation Results:Feasibility Comparisons

6. Conclusions

We study the problem on how to maximize the throughput for a periodic real-time system under the given peak temperature constraint. We incorporate the interdependency between the leakage, temperature and supply voltage into analysis and assume that different tasks in our system may have different power and thermal characteristics. Two algorithms are presented in this paper. The first one is built upon processors that can either be in active or sleep mode. By judiciously selecting tasks with different thermal characteristics as well as alternating the processor active/sleep mode, our approach can improve the throughput upon the existing techniques by 21% in average. We further extend this approach for processors with dynamic voltage/frequency scaling (DVFS) capability. Our experiments show that an improvement of 24% can be achieved when compared with the existing approaches.

Acknowledgment

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] Hotspot 4.2 temperature modeling tool. *University of Virgina*, page http://lava.cs.virginia.edu/HotSpot, 2009.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, 2000.

[3] T. Chantem, X. S. Hu, and R. Dick. Online work maximization under a peak temperature constraint. In *ISLPED*, pages 105–110, 2009.

[4] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximal temperature minimization for periodic hard real-time systems. In *ICESS*, 2010.

[5] H. Huang, G. Quan, and J. Fan. Leakage temperature dependency modeling in system level analysis. In *ISQED*, pages 447–452, 2010.

[6] R. Jayaseelan and T. Mitra. Temperature aware task sequencing and voltage scaling. In *ICCAD*, pages 618 – 623, 2008.

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

[8] S. Liu, M. Qiu, W. Gao, X.-J. Tang, and B. Guo. Hybrid of job sequencing and DVFS for peak temperature reduction with nondeterministic applications. In *ICESS*, pages 1780–1787, 2010.

[9] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *DATE*, pages 1526–1531, 2007.

[10] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang. Thermal vs energy optimization for DVFS-enabled processors in embedded systems. In *ISQED*, pages 204–209, 2007.

[11] G. Quan and Y. Zhang. Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks. *ECRTS*, pages 207–216, 2009.

[12] M. Santarini. Thermal integrity: A must for low-power ic digital design. *EDN*, pages 37–42, 2005.

[13] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. *ICSA*, pages 2–13, 2003.

[14] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *DATE*, pages 9–14, 2010.

[15] L.-T. Yeh and R. C. Chu. *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices*. ASME Press, New York, NY, 2002.

[16] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *ICCAD*, pages 281–288, 2007.

[17] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. In *DAC*, pages 585 – 590, 2010.