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

HUANG HUANG, Florida International University
VIVEK CHATURVEDI, Florida International University
GANG QUAN, Florida International University
JEFFREY FAN, Florida International University
MEIKANG QIU, University of Kentucky

In this paper, we study the problem on how to maximize the throughput of a periodic real-time system under a given peak temperature constraint. We assume that different tasks in our system may have different power and thermal characteristics. Two scheduling approaches are presented in this paper. The first one is built upon processors that can be in either *active* or *sleep* mode. By judiciously selecting tasks with different thermal characteristics, as well as alternating the processor's *active*/*sleep* mode, the sleep period required to cool down the processor is kept at a minimum level and, as the result, the throughput is maximized. We further extend this approach for processors with dynamic voltage/frequency scaling (DVFS) capability. Our experiments on a large number of synthetic test cases as well as real benchmark programs show that the proposed methods not only consistently outperform the existing approaches in terms of throughput maximization, but also significantly improve the feasibility of tasks when a more stringent temperature constraint is imposed.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Thermal management, task scheduling, dynamic voltage scaling, leakage/temperature dependency.

ACM Reference Format:
Huang, H., Chaturvedi, V., Quan, G., Fan, J., and Qiu, M. 2013. Throughput maximization for periodic real-time systems under the maximal temperature constraint. ACM Trans. Embedd. Comput. Syst. 1, 1, Article 1 (June 2013), 20 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The continued scaling of semiconductor technology has resulted in an exponential increase of the transistor density, which causes the escalading power consumption and the rapidly elevated temperature in the IC circuits. High temperature not only increases the packaging/cooling cost, but also shortens the life span of a computing system and degrades its performance, robustness, and reliability [Santarini 2005]. It is reported in [Yeh and Chu 2002] that a temperature increase of $10-15^oC$ can reduce the chip's lifespan by half.

In addition, due to the strong leakage temperature dependency, high chip temperature also dramatically increases the leakage power, which is becoming a major component of the overall power consumption in the deep sub-micron domain digital IC. As shown in [Liao et al. 2005] that the leakage power consumption can be 2-3 times higher than the dynamic power for a 65nm design. 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 needs to be incorporated into every abstraction level in the design of electronic computing systems [Skadron et al. 2003].

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

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

To tackle the high temperature problem, advanced packaging and cooling solutions are developed. These solutions are devised at the end of the design cycle based on the nominal peak power consumption of a chip. Thus, they can be too expensive to be designed for the worst-case scenarios, e.g. 1-3 dollar/watt [Skadron et al. 2003], given the exponentially increased power density. A less expensive method is to design the packaging and cooling solutions for the average case and use advanced hardware mechanisms to dynamically manage the chip temperature during run-time. Several approaches are proposed such as dynamic voltage/frequency scaling (DVFS), dynamic power down (DPD), task migration [Powell et al. 2004], fetch throttling [Brooks and Martonosi 2001], clock gating and etc. These mechanisms are triggered at run time when processor temperature reaches a predefined threshold to control the temperature. However, due to the reactive nature and the high execution overhead, these techniques often incur unexpected performance penalties and cannot guarantee that all tasks meet their deadlines, especially under the circumstances that the thermal emergencies are no longer infrequent events.

In this paper, we are interested in the problem of employing the scheduling technique to maximize the throughput of a real-time system under a given peak temperature constraint. Two novel approaches are proposed, one for processors 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. [Chantem et al. 2009; Jayaseelan and Mitra 2008; Zhang and Chatha 2010]) 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 with supply voltage [Liao et al. 2005]. As evidenced in [Huang et al. 2010], the leakage model ignoring the effect of supply voltage can lead to results deviated far away from actual values and thus produce potentially inaccurate solutions. Second, the existing approaches, e.g. [Jayaseelan and Mitra 2008] and [Zhang and Chatha 2010], assume that the task sequencing and the processor mode change can only occur at task boundaries. In contrast, we employ the same principle as implied in the *M-Oscillating* approach [Chaturvedi, Huang, and Quan Chaturvedi et al.] to sequence tasks and change processor modes during the task execution. Our experimental results, based on parameters drawn from the 65nm technology, show that our methods consistently outperform the existing approaches [Zhang and Chatha 2010] by over 23.3% and 5.3% in average for processor without and with DVFS capability, respectively.

The rest of this paper is organized as follows. Section 3 introduces the system models followed by the motivational examples in Section 4. Our proposed scheduling algorithms are discussed in Section 5. Experimental results are presented in Section 6 and Section 7 concludes this paper.

2. RELATED WORK

Due to the increased leakage power consumption and its strong dependency with the temperature in the DSM domain, recently, thermal-aware scheduling problems have attracted many research attentions. Existing research on thermal-aware scheduling, based on their objectives, can be largely grouped into the following three categories: (*i*) energy minimization under the timing or temperature constraint, e.g. [Chen et al. 2006; Jejurikar et al. 2005; Yuan et al. 2006; Bao et al. 2010, 2009; Huang and Quan 2011; Yang et al. 2010]; (*ii*) temperature minimization under the timing constraint, e.g. [Bansal et al. 2007; Chaturvedi, Huang, and Quan Chaturvedi et al.; Jayaseelan and Mitra 2008]; and (*iii*) throughput maximization under the peak temperature constraint, which the proposed work falls into.

The desire of high performance computing system together with the awareness of many negative effects of the high system temperature have driven the researchers into the thermal-aware throughput maximization problem. Significant efforts have been spent to address the thermal-constrained throughput maximization problem for single-processor platforms, e.g. [Wang and Bettati 2006; Quan et al. 2008; Chantem et al. 2009; Zhang and Chatha 2007, 2010], as well as multi-processor platforms, e.g. [Hanumaiah et al. 2009a,b; Liu et al. 2010b; Zhou et al. 2008; Lung et al. 2011; Zhu et al. 2008; Sun et al. 2007; Coskun et al. 2009].

For single-processor platforms, Wang et al. [Wang and Bettati 2006] proposed an effective two speed reactive scheme that runs the processor at the maximum speed until it reaches the temperature threshold, then uses an equilibrium speed to maintain the temperature. Quan et al. [Quan et al. 2008] proposed a closed-form formula for feasibility analysis for a given peak temperature threshold. For multi-processor

platforms, Liu et al. [Liu et al. 2010b] proposed a thermal-aware job allocation algorithm which assigns hot tasks to the cores close to the heat sink by taking advantage of its better heat removing capability. Cool tasks are assigned to cores far away from heat sink. By doing so, better thermal condition can be achieved such as a lower peak temperature and less temperature variations. In [Zhou et al. 2008], Zhou et al. proposed a task scheduling technique based on the observation that the vertically adjacent cores have strong thermal correlations. This method then jointly considers vertically adjacent cores and forms them into super-cores. Lung et al. [Lung et al. 2011] presented a fast thermal simulation method, based on which, a task allocation algorithm is proposed by assigning a given task to a core that can lead to a minimized peak temperature. In [Coskun et al. 2009], Coskun et al. proposed a thermal-aware scheduling method called Adapt3D that takes the thermal history into account to reduce the number of hot spot occurrences. Most of these works do not consider the leakage/temperature dependency in the analysis. They either ignore the leakage power consumption or simply treat it as constant.

There are some closely related works that have considered the leakage/temperature dependency when dealing with throughput maximization problems, e.g. [Chantem et al. 2009; Zhang and Chatha 2007, 2010]. Specifically, Chantem et al. [Chantem et al. 2009] proposed to run real-time tasks by frequently switching between the two speeds which are neighboring to a constant speed. This work targets only at the scenario when the processor reaches its thermal steady state. Zhang and Chatha [Zhang and Chatha 2007] presented a pseudo-polynomial time speed assignment 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 ending temperature of each period not exceeding 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 real world. As shown in [Jayaseelan and Mitra 2008], the power and thus the thermal characteristics of different real-time tasks can be significantly different. With this fact in mind, Jayaseelan and Mitra [Jayaseelan and Mitra 2008] proposed to construct the task execution sequence to minimize the peak temperature. Zhang and Chatha [Zhang and Chatha 2010] further developed several algorithms to maximize the throughput of a real-time system by sequencing the task execution for processors with and without DVFS capability. Both works ([Jayaseelan and Mitra 2008] and [Zhang and Chatha 2010]) assume that the processor can only change its speed at boundaries of task execution.

In this paper, we address the single-processor throughput maximization problem by considering both the leakage/temperature and leakage/supply voltage dependencies. Different from [Chantem et al. 2009] and [Zhang and Chatha 2007], we assume that different tasks have different power/thermal profiles, which are more practical in real world applications. In addition, by employing more complex intra task scheduling policies, our methods can greatly improve the throughput performance than those proposed in [Zhang and Chatha 2010]. In what follows, we first introduce the system models we used in this paper. Then, we present our scheduling algorithms for processors with and without DVFS feature, respectively.

3. PRELIMINARIES

In this section, we briefly introduce our system models, 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 closely related works (e.g. [Liu et al. 2010a; Chaturvedi, Huang, and Quan Chaturvedi et al.; Zhang and Chatha 2010; Chantem et al. 2009]) that

$$RC\frac{dT(t)}{dt} = RP(t) - (T(t) - T_{amb}), \tag{1}$$

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 $T(t)$ such that T_{amb} is zero (i.e. by replacing $T(t) - T_{amb}$ with $T(t)$) and then we have

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

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 a processor operates in the *active* mode. We assume that a processor can be switched from one mode to another at any time. However, such a switching can 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 a way that the power consumptions of different tasks vary significantly even running with the same speed level and at the same temperature. This is because the power consumptions are strongly depending on the circuit activity [Liu et al. 2007] 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 functional units are being used. For common benchmark tasks, the *activity factors* can be obtained by using architectural-level power analysis tools such as *Wattch* [Brooks et al. 2000]. Similarly, we also define the *leakage factor* ξ, which can be used as the scaling factor of the leakage power of a processor when different tasks are executed.

Power Model: The dynamic power consumption is independent to the temperature and can be formulated as $P_{dyn} \propto C_{load} f v^2$, where C_{load} is the equivalent load capacitance, f is the clock frequency and v is the supply voltage. Assuming the working frequency is in proportion to the supply voltage and with the *activity factor* taken into consideration, the dynamic power consumption of a processor when executing the task τ_i at the kth speed levels can be formulated as

$$P_{dyn}(i,k) = \mu_i C_2 v_k^3, \tag{3}$$

where v_k is the supply voltage level, C_2 is a constant and μ_i is the *activity factor* of task τ_i.

The leakage power is temperature dependent and can be calculated as $P_{leak} = N_{gate} \cdot I_{leak} \cdot v_k$, where N_{gate} represents the number of gates and I_{leak} is the leakage current, which can be formulated by a non-linear exponential equation as [Liao et al. 2005]

$$I_{leak} = I_s \cdot (\mathcal{A} \cdot T^2 \cdot e^{((\alpha \cdot V_{dd} + \beta)/T)} + \mathcal{B} \cdot e^{(\gamma \cdot V_{dd} + \delta)}), \tag{4}$$

where I_s is the leakage current at certain reference temperature and supply voltage, T is the operating temperature, V_{dd} is the supply voltage, $\mathcal{A}, \mathcal{B}, \alpha, \beta, \gamma, \delta$ are empirically determined technology constants. As leakage current changes super linearly with temperature [Liu and Yang 2010], the leakage power of the processor when executing the task τ_i can be effectively estimated as

$$P_{leak}(i,k) = \xi_i (C_0(k) v_k + C_1(k) T v_k), \tag{5}$$

where ξ_i is the leakage factor, $C_0(k)$ and $C_1(k)$ are curve fitting constants. Therefore, the overall power consumption of a processor when executing the task τ_i at the kth speed level can be modeled as

$$P(i,k) = \xi_i (C_0(k) v_k + C_1(k) \cdot T v_k) + \mu_i C_2 v_k^3. \tag{6}$$

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

$$\frac{dT(t)}{dt} = A(i,k) - B(i,k) T(t), \tag{7}$$

where $A(i,k) = a(\xi_i C_0(k) v_k + \mu C_2 v_k^3)$ and $B(i,k) = b - a\xi_i C_1(k) v_k$ (we use B_s for $B(sleep)$, a and b are defined in equation (2)). Hence, for a given time interval $[t_0, t_e]$, if the initial temperature is T_0, by solving

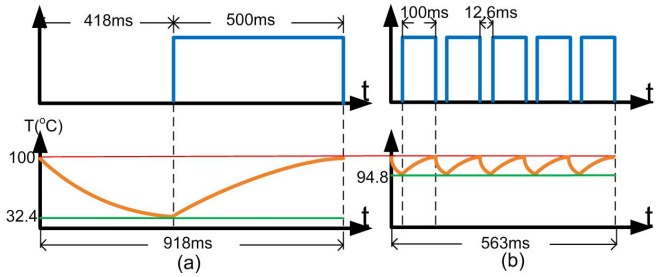


Fig. 1. (a) If a *hot* task starts running from the peak temperature limit, i.e. $T_{max} = 100^oC$, one has to insert 418ms of idle period to cool down the processor to a *safe temperature* which results in a total latency of 918ms. (b) By evenly splitting the task execution into 5 sections (100ms each) and inserting sleep period before each section, only $12.6 \times 5 = 63ms$ of idle period is required to guarantee the same peak temperature limit, which results in 563ms of total latency.

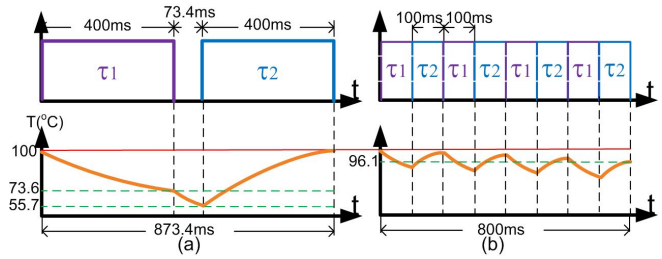


Fig. 2. (a) Given a *cool* task τ_1 and a *hot* task τ_2, one can run the *cool* task to cool down the temperature from T_{max}, then only 73.4ms of idle interval needs to be inserted before the *hot* task. (b) If we divide *cool* task τ_1 and *hot* task τ_2 each into 4 sections and run them alternatively, the sleep period can be totally eliminated while ensuring the same peak temperature constraint.

equation (7), the ending temperature of executing task τ_i can be formulated as below:

$$\begin{aligned} T_e &= \frac{A(i,k)}{B(i,k)} + (T_0 - \frac{A(i,k)}{B(i,k)})e^{-B(i,k)(t_e-t_0)} \\ &= T_{ss}(i,k) + (T_0 - T_{ss}(i,k))e^{-B(i,k)(t_e-t_0)}, \end{aligned} \tag{8}$$

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 of executing a task in one period is minimized. Our research problem can be formulated as follows.

PROBLEM 1. *Given a task set* $\Gamma = \{\tau_1(t_1,\mu_1,\xi_1), \tau_2(t_2,\mu_2,\xi_2), ..., \tau_n(t_n,\mu_n,\xi_n)\}$, *where* t_i, μ_i *and* ξ_i *are the execution time,* activity factor *and the leakage factor of task* τ_i, *respectively, develop a feasible schedule such that the latency of executing one iteration of* Γ *is minimized under the thermal steady state while ensuring a given peak temperature constraint* T_{max}.

4. MOTIVATIONAL EXAMPLES

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

To prevent the temperature from exceeding T_{max}, we can turn the processor into *sleep* mode and let the processor to cool down, as illustrated in Figure 1(a). Based on a processor and power model detailed later in Section 6, the processor has to stay in the *sleep* mode for $418ms$ to make sure the temperature

is dropped to the *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 in 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 (i.e. 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 helps to improve the system throughput. This example demonstrates that it is more effective to insert multiple idle intervals *inside* the task than only at task boundaries to improve the throughput under the same 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 that 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 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 lower down the temperature 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 equally divide the execution of τ_1 and τ_2 each 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 ending temperature is reduced to 96.1^oC . This saved temperature budget (i.e. 3.9^oC) could be utilized to further improve the throughput of the ensuing tasks.

It is worth mentioning that in the above examples as well as the following discussions, the mode switching overhead is only considered in the timing analysis whereas omitted in the thermal analysis for simplicity. The reason is that the t_{sw} is sufficiently small so that the temperature variation during t_{sw} is negligible. Moreover, during the mode switching the processor clock is halted that no workload can be executed. Thus, in fact the chip temperature during t_{sw} is slightly decreasing, if not ignored. Therefore, if one method can guarantee the temperature constraint without considering the temperature variation in t_{sw}, it is bound to be feasible if we incorporate t_{sw} into the thermal analysis.

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 appropriately choose the number of sections that a task needs to be split to achieve the best performance, i.e. throughput? We address these problems in the following sections.

5. OUR APPROACH

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

5.1. Sleep Mode Distribution for Hot Tasks

We begin our discussion by assuming that a 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). When $T_{ss} \leq T_{max}$, the problem becomes trivial because no temperature limit violation can occur. Since Γ is a periodic task set and it has been shown [Chantem et al. 2009; Zhang and Chatha 2010] that the throughput of Γ is maximized when the ending temperature of each period equals T_{max}, we can conveniently make the initial temperature of Γ equal to the ending temperature of each iteration, and set them to T_{max}.

Since all tasks are *hot* tasks, starting at T_{max}, we can only lower down the temperature by inserting idle intervals. The question is how long we should insert the interval. The shorter the total length of all idle

intervals is, the smaller the overall latency is and thus the larger the throughput can be. To quantify the effectiveness of different choices, we introduce a metric called the *idle ratio* (Θ) which is defined as the ratio between the time that a processor stays in the *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 shorter the overall idle interval is needed. The observation is formulated in the following theorem.

THEOREM 5.1. *Given a task τ_i, a processor with only one active and one sleep mode, and the maximal temperature constraint T_{max}, assume that $T_{ss}(i) > T_{max}$ (since there is only one active mode available, for simplicity reason, we omit the parameter that represents the speed level, i.e. k, and use $T_{ss}(i)$, $B(i)$ instead of $T_{ss}(i,k)$, $B(i,k)$). Let $\Theta(m)$ represent the idle ratio of a feasible schedule when τ_i is evenly split into m sections. Assume the transition overhead is negligible. Then,*

— *The idle ratio $\Theta(m)$ is a monotonically decreasing function of m.*
— *The lower bound of the idle ratio Θ_{min} exists as m approaching to infinity such that*

$$\lim_{m->\infty} \Theta(m) = \frac{B(i)}{B_s} \frac{T_{ss}(i) - T_{max}}{T_{max}}. \tag{9}$$

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, we first need to find the *safe* temperature, i.e. $T_{safe}(i)$, of the task given the peak temperature limit T_{max}. For a given hot task, the *safe* temperature is the one that if the starting temperature is T_{safe}, the ending temperature reaches exactly at T_{max}. From temperature dynamic relationship given in equation (8), by letting the ending temperature T_e equal T_{max}, we can solve for T_0 to obtain the corresponding *safe* temperature of task i,

$$T_{safe}(i) = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}, \tag{10}$$

Once the *safe* temperature is available, we can calculate how long the processor needs to stay in the idle mode to reduce the temperature from T_{max} to $T_{safe}(i)$. Again, we can use equation (8). This time, the initial and ending temperatures are given, the sleep period t_s is the length of the interval that we want to solve for,

$$\begin{aligned} t_s &= -\frac{1}{B_s} \cdot \ln(\frac{T_{safe}(i)}{T_{max}}) \\ &= -\frac{1}{B_s} \cdot \ln(\frac{T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}}{T_{max}}). \end{aligned} \tag{11}$$

Therefore, the *idle ratio* of the original task, i.e. when $m = 1$, can be expressed as

$$\Theta(1) = \frac{t_s}{t_i} = -\frac{1}{B_s t_i} \cdot \ln(\frac{T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}}{T_{max}}). \tag{12}$$

If the execution of τ_i is evenly split into m sections, we have t_i/m seconds of active period in each section. The corresponding $safe$ temperature, i.e. $T_{safe}(i,m)$, becomes

$$T_{safe}(i,m) = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)\frac{t_i}{m}}}. \tag{13}$$

Based on this *safe* temperature, we only need to insert $t_s(m)$ seconds of sleep period before each active period that

$$t_s(m) = -\frac{1}{B_s} \cdot \ln(\frac{T_{safe}(i,m)}{T_{max}}). \tag{14}$$

Now we can formulate the idle ratio Θ as a function of m,

$$\begin{aligned}\Theta(m) &= \frac{t_s(m)}{t_i/m} \\ &= -\frac{m}{B_s t_i}\cdot\ln(\frac{T_{safe}(i,m)}{T_{max}}) \\ &= \frac{1}{B_s t_i}\cdot\ln(\frac{T_{max}}{T_{safe}(i,m)})^m. \end{aligned} \tag{15}$$

To prove Theorem 5.1, we only need to show that the first order derivative of $\theta(m)$ is always less than zero, i.e. $\frac{d\Theta(m)}{dm} < 0$. Then, we have

$$\begin{aligned}\frac{d\Theta(m)}{dm} &= \frac{1}{B_s t_i}\cdot(\frac{T_{safe}(i,m)}{T_{max}})^m \\ &\cdot\ m(\frac{T_{max}}{T_{safe}(i,m)})^{m-1}\cdot\frac{(-1)\cdot T_{max}\cdot\frac{dT_{safe}(i,m)}{dm}}{T_{safe}(i,m)^2}. \end{aligned} \tag{16}$$

On the R.H.S of equation (16), the parameter B_s, t_i, m, T_{max}, T_{safe} are all greater than zero, therefore the sign of $\frac{d\Theta(m)}{dm}$ depends only upon the term $\frac{dT_{safe}(i,m)}{dm}$. Observing equation (13), it is not difficult to see that the function $T_{safe}(i,m)$ is monotonically increasing with m. The physical meaning of this equation is that with larger m (smaller active period), the required *safe* temperature can be higher to guarantee the same peak temperature limit. Therefore, we have $\frac{dT_{safe}(i,m)}{dm} > 0$. Hence $\frac{d\Theta(m)}{dm} < 0$ and $\Theta(m)$ monotonically decreases with m is proved.

We next find the lower bound of $\Theta(m)$. As $m \to \infty$, we denote the active time and sleep time in each division as t_i' and t_s' respectively. Also, the corresponding *safe* temperature is $T'_{safe}(i)$. Then, based on equation (11) we have the following relationships,

$$t_s' = -\frac{1}{B_s}\ln(\frac{T'_{safe}(i)-0}{T_{max}-0}), \tag{17}$$

$$t_i' = -\frac{1}{B(i)}\ln(\frac{T_{max}-T_{ss}(i)}{T'_{safe}(i)-T_{ss}(i)}). \tag{18}$$

Because a shorter active time requires a higher *safe* temperature, thus, in the extreme case when $m \to \infty$ we have $T'_{safe}(i) \to T_{max}$. Then, the lower bound of Θ can be calculated as

$$\Theta_{min} = \lim_{T'_{safe}\to T_{max}}(\frac{1}{B_s}\ln(\frac{T'_{safe}(i)}{T_{max}}))/(\frac{1}{B(i)}\ln(\frac{T_{max}-T_{ss}(i)}{T'_{safe}(i)-T_{ss}(i)})). \tag{19}$$

Apparently, when $T'_{safe} \to T_{max}$, the above limit calculation involves indeterminate term ($\frac{0}{0}$ type). Therefore, we apply the L'Hopital's rule [Wikipedia 2013] to find the first derivative of the numerator and denominator of equation (19) respectively. Then we have

$$\begin{aligned}\Theta_{min} &= \lim_{T'_{safe}\to T_{max}}(\frac{dt_s'}{dT'_{safe}})/(\frac{dt_a'}{dT'_{safe}}) \\ &= \frac{B(i)}{B_s}\frac{T_{ss}(i)-T_{max}}{T_{max}}. \end{aligned} \tag{20}$$

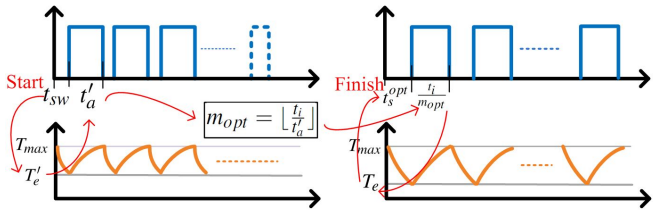


Fig. 3. The procedure to find the appropriate number of m for our sleep time distribution method by considering the non-negligible transition overhead.

□

Theorem 5.1 implies that the smaller we divide a task, the shorter the idle interval is needed. However, the impact of distributing idle intervals between the task execution eventually can be saturated as we increase the number of m.

Based on the above discussion, in order to maximize the throughput of a processor, we should divide the execution of task τ_i into as many sub-intervals as possible. However, since there exists a lower bound of $\Theta(m)$, the impact of dividing the task becomes saturated as m increases. Furthermore, as $m \to \infty$, the context switching overhead cannot be ignored anymore, no matter how small 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 derive a method, i.e. Algorithm 1, to find the optimized sleep distribution pattern. Specifically, we want to find the optimized number of sub-intervals m_{opt} and the sleep time in each sub-interval t_s^{opt}. The procedure is illustrated in Figure 3.

As illustrated in Figure 3, starting from T_{max}, we first calculate the corresponding ending temperature of the sleep period by equation (8),

$$T'_e = T_{amb} + (T_{max} - T_{amb})e^{-B_s t_{sw}}. \tag{21}$$

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

$$t'_i = -\frac{1}{B(i)} \ln\left(\frac{T_{max} - T_{ss}(i)}{T'_e - T_{ss}(i)}\right). \tag{22}$$

Accordingly, we have $m_{opt} = \lfloor \frac{t_i}{t'_i} \rfloor$ (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 determined, i.e. $\frac{t_i}{m_{opt}}$. The ending temperature of the corresponding sleep period can also be obtained as

$$T_e = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)\frac{t_i}{m_{opt}}}}. \tag{23}$$

Finally, the minimized sleep time per sub-interval t_s^{opt} can be solved from equation (11) by replacing $T_{safe}(i)$ with T_e in equation (23)

$$t_s^{opt}(i) = -\frac{1}{B_s} \ln\left(\frac{T_{ss}(i)(e^{-B(i)\frac{t_i}{m_{opt}}} - 1)}{e^{-B(i)\frac{t_i}{m_{opt}}}} - 1\right). \tag{24}$$

The final schedule generated by our method will be t_s^{opt} seconds of sleep period followed by $\frac{t_i}{m_{opt}}$ seconds of task execution. Repeating this pattern by m_{opt} times, the latency of task τ_i can be greatly reduced while the peak temperature limit is guaranteed.

Although Algorithm 1 targets at a 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. For a special case that a task set consisting of hot tasks with homogeneous power, the task set can be considered as a

single task with the execution time $t_{exe} = \sum t_i, (\tau_i \in \Gamma)$. The corresponding sleep distribution pattern can be found conveniently.

Algorithm 1 Sleep mode distribution for single hot task

1: ***SleepDistribution***(TASK τ_i)
2: //Set initial value of $t_s^{opt} = t_{sw}$
3: $t_s^{opt} = t_{sw}$;
4: //Find the ending temperature of sleep mode
5: $T_e'(t_{sw}) = T_{amb} + (T_{max} - T_{amb})e^{-B_s t_{sw}}$;
6: //Find the execution time of the ensuing active period
7: $t_a' = -\frac{1}{B(i)} \ln(\frac{T_{max} - T_{ss}(i)}{T_e'(t_{sw}) - T_{ss}(i)})$;
8: //Find optimum m
9: $m^{opt} = \lfloor \frac{t_i}{t_a'} \rfloor$;
10: //Find the minimal idle interval per division
11: $t_s^{opt} = -\frac{1}{B_s} \cdot \ln(\frac{T_e'}{T_{max}})$;
12: $latency = m \cdot t_s^{opt} + t_i$;
13: return (*latency*);

5.2. Improving Throughput by Task Switching

In this subsection, we extend our discussion to a task set consisting of both *hot* and *cool* tasks. Let us first consider only two tasks, i.e. one *hot* and one *cool* task. 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. However, this method cannot always guarantee the entire temperature curve stays below the temperature threshold. Whether the scheme works or not depends on the power consumptions and durations of the cool task and the hot task, respectively. Therefore, in this subsection, we develop a systematic way to determine whether a task switching scheme can be applied to a hot/cool task pair, and if yes, how to find the optimal number of task switching to achieve the maximum throughput.

Given a task pair, i.e. τ_i and τ_j $(T_{ss}(i) < T_{max} < T_{ss}(j))$, both τ_i and τ_j are equally divided into m divisions and alternatively executed (with τ_i first). Let the initial temperature be T_{max}. The temperature is first lowered down by running one section of the cool task, then it goes up by running one section of the hot task. We call the temperature after running the first section of the hot task as the *critical temperature*, i.e. T_c. With regard to the *critical temperature*, we have the following theorem.

THEOREM 5.2. *Given a task pair, i.e. τ_i and τ_j, with $T_{ss}(i) < T_{max} < T_{ss}(j)$, let both τ_i and τ_j be equally divided into m sections and alternatively executed with the initial temperature equals T_{max} (with τ_i first). Then, the peak temperature constraint (T_{max}) can be guaranteed* iff $T_c \leq T_{max}$.

Proof: The proof of Theorem 5.2 is straightforward. If $T_c > T_{max}$, then apparently we have temperature limit violation. In contrast, if $T_c \leq T_{max}$, we can guarantee that the entire temperature curve stays below T_{max} during the rest of the task execution. This is because if the initial temperature of the second cool task division (T_c) is less than previous initial temperature T_{max}, it results into an even lower starting temperature for the second hot task division. By repeating this pattern, the entire temperature curve continues to decrease from T_{max}. □

In addition, a similar theorem as Theorem 5.1 can be established for a task set consisting both *hot* and *cool* tasks as follows.

THEOREM 5.3. *Given a task pair, i.e. τ_i and τ_j, with $T_{ss}(i) < T_{max} < T_{ss}(j)$, let both τ_i and τ_j be equally divided into m sections and alternatively executed with the initial temperature equals T_{max} (with τ_i first). Then the* critical temperature *w.r.t different m, i.e. $T_c(i, j, m)$, is a monotonically decreasing function of m if*

$$m < \frac{B(i)t_i}{\ln(\frac{K_1}{K_2} \cdot \frac{T_{max}-T_{ss}(i)}{T_{ss}(j)-T_{ss}(i)})}, \tag{25}$$

where $K_1 = B(i)t_i + B(j)t_j$ *and* $K_2 = B(j)t_j$. $T_c(i,j,m)$ *is interpreted as the ending temperature after running the first cool* (τ_i) *and hot* (τ_j) *task sections, if they are divided by m sections.*

Proof: To prove Theorem 5.3, we need to find the range of m that makes the first order derivative of $T_j(i,j,m)$ less than zero. Given an initial temperature T_{max}, based on equation (8), the ending temperature of task τ_i after $\frac{t_i}{m}$ seconds of execution can be expressed as

$$T_i(i,j,m) = T_{ss}(i) + (T_{max} - T_{ss}(i))e^{-B(i)\frac{t_i}{m}}. \tag{26}$$

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

$$T_c(i,j,m) = T_{ss}(j) + (T_i(i,j,m) - T_{ss}(j))e^{-B(j)\frac{t_j}{m}}. \tag{27}$$

After replacing $T_i(i,j,m)$ by equation (26), we have

$$\begin{aligned} T_c(i,j,m) &= T_{ss}(j) + (T_{ss}(i) - T_{ss}(j))e^{\frac{-K_2}{m}} \\ &+ (T_{max} - T_{ss}(i))e^{\frac{-K_1}{m}}. \end{aligned} \tag{28}$$

Find the first order derivative of $T_c(i,j,m)$, we get

$$\begin{aligned} \frac{dT_c(i,j,m)}{dm} &= \frac{K_2}{m^2} \cdot (T_{ss}(i) - T_{ss}(j))e^{\frac{-K_2}{m}} \\ &+ \frac{K_1}{m^2} \cdot (T_{max} - T_{ss}(i))e^{\frac{-K_1}{m}}. \end{aligned} \tag{29}$$

Let $\frac{dT_c(i,j,m)}{dm} < 0$ and solve the inequality,

$$\begin{aligned} \frac{dT_c(i,j,m)}{dm} &< 0, \\ K_1(T_{max} - T_{ss}(i))e^{\frac{-K_1}{m}} &< K_2(T_{ss}(j) - T_{ss}(i))e^{\frac{-K_2}{m}}, \\ \frac{K_1}{K_2} \cdot \frac{T_{max} - T_{ss}(i)}{T_{ss}(j) - T_{ss}(i)} &< e^{\frac{B(i)t_i}{m}}, \\ m &< \frac{B(i)t_i}{\ln(\frac{K_1}{K_2} \cdot \frac{T_{max}-T_{ss}(i)}{T_{ss}(j)-T_{ss}(i)})}. \end{aligned} \tag{30}$$

Hence, Theorem 5.3 proved. □

Theorem 5.3 implies that given a *cool/hot* task pair, increasing m within the bound specified by equation (30), or

$$m^1_{bound} = \lfloor \frac{B(i)t_i}{\ln(\frac{K_1}{K_2} \cdot \frac{T_{max}-T_{ss}(i)}{T_{ss}(j)-T_{ss}(i)})} \rfloor, \tag{31}$$

helps to identify a feasible schedule. When $m > m^1_{bound}$, the *critical temperature* tends to increase. Moreover, increasing m also increases the switching overhead. Therefore, the problem becomes how to judiciously choose the appropriate m ($m \leq m^1_{bound}$) to maximize the throughput with non-negligible switching overhead being considered.

Note that, the non-negligible switching overhead also poses a bound to the choice of m. The total amount of switching time associated with m task switching is $m \cdot t_{sw}$. If the original idle interval required

to cool down a *hot* task from T_{max} to T_{safe} is t_{cool}, we must have $m \cdot t_{sw} < t_{cool}$ in order to further reduce the latency. Therefore, we set another bound of m that

$$m^2_{bound} = \lfloor \frac{t_{cool}}{t_{sw}} \rfloor. \tag{32}$$

Then, we have the upper bound of m defined as

$$m_{max} = MIN\ (m^1_{bound},\ m^2_{bound}). \tag{33}$$

The optimal m can be found by sequentially search from 1 to m_{max} (among positive integers). The searching is stopped as soon as we find an m so that $T_c(i,j,m) \leq T_{max}$. Note that, if the ending temperature at $m = m_{max}$ still cannot meet the temperature constraint, the task switching scheme fails. Thus, we first test if $T_c(i,j,m_{max}) < T_{max}$, and the searching process starts only if the result is true. The hot/cool task pairing algorithm is depicted in Algorithm 2. Our simulation results show that $T_c(i,j,m)$ decreases drastically at the first several steps (small m), in most cases, the optimal m can be obtained within 5 iterations.

Algorithm 2 Pairing hot/cool task

```
1:  TaskPairing(TASK τ_i, TASK τ_j)
2:      //Find the upper bound of m
3:      m_max = MIN (m^1_bound, m^2_bound);
4:      //Find the T_c at m = m_max, check constraint
5:        if (T_c(i,j,m_max)) ≤ T_max
6:      //Find m_opt by sequential search from m = 1
7:          for k = 1; k < m_max; k++
8:            if (T_c(i,j,k)) ≤ T_max
9:      //If constraint satisfied, get m
10:                 m_opt = k;
11:                 Latency = m_opt · t_sw + t_i + t_j;
12:                 return (Latency);
13:           endif
14:         endfor
15:       else return 0
```

We present the proposed throughput maximization algorithm in Algorithm 3, for processors with one active and one sleep mode. 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 decreasing order of their ending temperatures (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. 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. The latency of the hot/cool task pair is calculated by $TaskPairing()$ defined in Algorithm 2. Then, both tasks are marked as scheduled and their ending temperature after m_{opt} switchings needs to be updated as the new initial temperature of the next attempted pairing.

To find the ending temperature, we derived a closed-form formula based on [Quan and Chaturvedi 2010], that if there exists m_{opt} times of task switching between τ_i and τ_j, the ending temperature of the schedule can be formulated as

$$T_{end}(i,j,m) = T_{ini} + \frac{(T_c(i,j,m) - T_{ini}) \cdot (1 - K_3^m)}{1 - K_3}, \tag{34}$$

Algorithm 3 Throughput maximization without DVFS

Input: $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$
Initialization: classify all tasks into cool/hot tasks based on T_{ss};
Find T_{end} for all cool tasks;
Find T_{safe} for all hot tasks;
Sort cool tasks into Q_c in decreasing order of T_{end};
Sort hot tasks into Q_h in increasing order of T_{safe};
$T_{ini} = T_{max}$;
for i=1:length(Q_h)
  for j=1:length(Q_c)
    if (tasks NOT 'scheduled') && (pairing is feasible)
      Latency = *Latency* +***TaskPairing***($Q_c[j]$, $Q_h[i]$);
      Mark $Q_c[j]$ and $Q_h[i]$ as 'scheduled';
      Update initial temperature: $T_{ini} = T_{end}(i, j, m_{opt})$;
    endif
  endfor
endfor
for i=1:length(Q_h)
  if ($Q_h[i]$ is not 'scheduled')
    Latency = *Latency* + ***SleepDistribution***($Q_h[i]$);
    Mark $Q_h[i]$ as 'scheduled';
endfor
for i=1:length(Q_c)
  if ($Q_c[i]$ is not 'scheduled')
    Latency = *Latency* + $t_{Q_c[i]}$;
    Mark $Q_c[i]$ as 'scheduled';
endfor
Return: (*Latency*);

where T_{ini} is the initial temperature when the pairing between task i and j starts, and $K_3 = e^{-B(i)\frac{t_i}{m} - B(j)\frac{t_j}{m}}$. Based on the above equation, the ending temperature after m_{opt} times of task switching can be obtained by replacing m with m_{opt}.

For a given hot task $Q_h[i]$, if the attempted task pairing fails with the pth cool task, i.e. $Q_c[p]$, it is still possible to make a feasible combination with the qth ($p \neq q$) *cool* task, i.e. $Q_c[q]$. 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 attempted task pairing procedure, if there are still tasks left in Q_c or Q_h, the *hot* tasks are executed with the $SleepDistribution()$ introduced in Section 5.1 and the cool tasks are simply attached to the end of the final schedule.

The computational complexity of Algorithm 3 mainly comes from pairing hot/cool tasks. Note that, since m_{max} is usually very small in practice as explained before, Algorithm 2 has a complexity of O(1). Therefore, the complexity of Algorithm 3 is O($Q_h \times Q_c$) where Q_h and Q_c represent the length of the hot and cool task queue, respectively. It is worthy of mentioning that, even though we can find the optimal m for a pair of hot/cool tasks, Algorithm 3 is a greedy algorithm in nature, since we require both the hot and cool tasks be split into the same (i.e. m) and equal sections. Also, one hot task can only be paired with one cool task in Algorithm 3. How to pair a hot task with multiple cool tasks and then split the tasks accordingly is an interesting problem and will be our future work.

5.3. 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. a processor with $N(N > 1)$ different *active* modes, and 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 trade-off needs careful analysis.

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

$$t_i(k-l) = \frac{f_k}{f_{k-l}} t_i(k), \tag{35}$$

where f_k and f_{k-l} are the clock frequencies associated with V_{dd} level k and $k-l$, respectively. From equation (11), the required sleep time of task τ_i under supply voltage levels $k-l$ can be obtained and expressed as

$$t_s(i,k-l) = -\frac{1}{B_s} \ln\left(\frac{T_{ss}(i,k-l)(e^{-B(i,k-l)\cdot t_i(k-l)} - 1)}{e^{-B(i,k-l)\cdot t_i(k-l)}} - 1\right). \tag{36}$$

Note that if the steady state temperature of task τ_i at the $(k-l)$th speed level is equal to or less than T_{max}, then it becomes a cool task and in this case, $t_s(i,k-l) = 0$.

Then, the overall latency of τ_i at speed level $k-l$ is formulated as

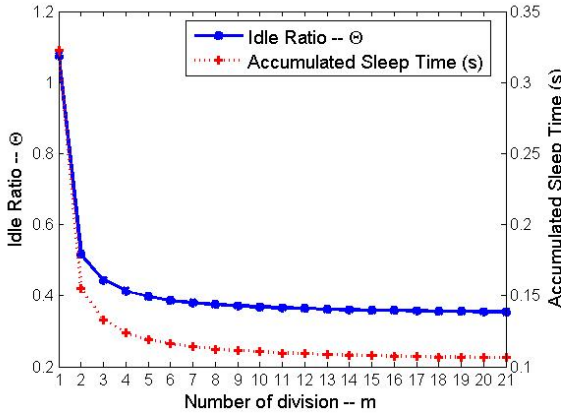
$$t(i,k-l) = t_s(i,k-l) + t_i(k-l). \tag{37}$$

Our approach to optimize the throughput by adjusting the speed level of a given task is to calculate the overall latency of the task under all supply voltage levels. Then, the speed level that leads to the minimized latency, i.e. k_{opt}, will be selected. We can also take advantage of the proposed sleep distribution option to see if we can further improve the throughput. Given a task, we apply Algorithm 4 to find the proper speed level for latency minimization.

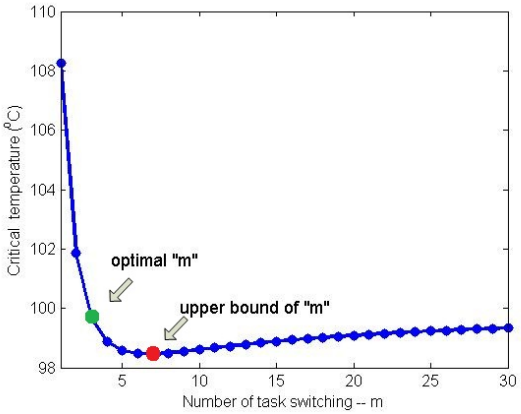
Algorithm 4 Find the optimal speed level for task

Calculate-Speed-Opt(TASK τ)
//Find total latency by sleep distribution at the Max.Spd
  $Latency[SpdMax-1]$=***SleepDistribution***(τ);
//Optimum speed initially set as the Max.Spd
  $SpdOpt = SpdMax-1$;
for $L = SpdMax-2$; $L >= 0$; $L--$
  if $T_{ss} > T_{max}$
    $Latency[L]$ = ***SleepDistribution***(τ);
  else
    $Latency[L] = t(\tau, L)$; Eq. 37
  endif
  if $Latency[L] < Latency[L+1]$
    $SpdOpt = L$;
  endif
endfor
return ($SpOpt$);

In Algorithm 4, given a hot task, we first calculate the overall latency by applying our speed distribution method proposed in Section 5.1 and the speed level is initially set to the maximum speed. Then, we reduce the speed level one by one and check if there is improvement in terms of overall latency. Finally, the speed level is returned after we check all available speed levels. Here, we want to emphasize that, whenever we reduce the speed level, the corresponding steady state temperature of a given task is evaluated because a hot task might become a cool task. If that is the case, no sleep time is needed and instead, we only need to set the latency as the actually execution time of the task at that speed level.



(a) Theorem 5.1 validation: The execution latency of a task is monotonically decreasing as we increase the number that we divide the task, i.e. m.

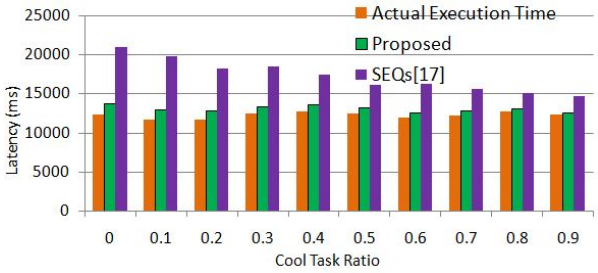


(b) Theorem 5.3 validation: The critical temperature is a monotonically decreasing function of m when $m < m_{bound}$, where $m_{bound} = 7$ in this example.

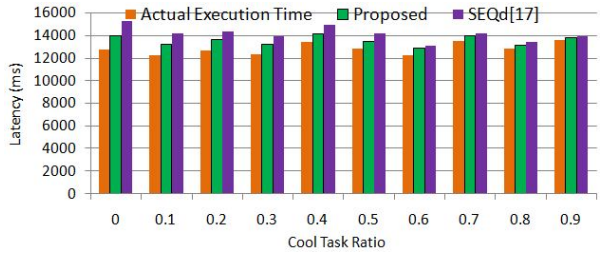
Fig. 4. Theorem Validation

6. EXPERIMENTAL RESULTS

In this section, we validate the theorems and show the performance of the proposed approaches through a set of simulations. Similar to [Quan and Zhang 2009], We built our processor model based on the work by [Liao et al. 2005] for a processor using 65nm technology. Liao et al. developed an analytical formula (equation (4)) that can estimate the leakage current with less than 1% error. We used the same formula to compute the leakage currents for temperature from 25^oC to 110^oC with step size of 10^oC, which were used to determine curve fitting constants $C_0(k)$ and $C_1(k)$ in equation (5). The mode switching overhead is assumed to be 5ms [Yang et al. 2010]. The thermal model is obtained from [hot 2009]. We let the ambient temperature and the maximal temperature limit to be 25^oC and 100^oC [Zhang and Chatha 2010], respectively, unless otherwise specified.



(a) Processor without DVFS



(b) Processor with DVFS

Fig. 5. Simulation Results: Latency Comparisons for synthetic task sets of different scheduling approaches

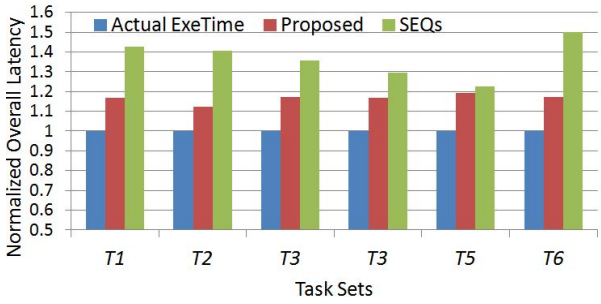
6.1. Theorem Validation

To validate the conclusion made in Theorem 5.1, 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 are plotted in Figure 4(a) while we increase m from 1 to m_{max}. The result conforms to the conclusion in Theorem 5.1 that Θ monotonically decreases with m. When $m = 1$ the schedule is identical to the one proposed in [Zhang and Chatha 2010] that $322ms$ sleep time is required to cool down the processor before τ_1 starts to run (thus $\Theta = 1.07$). As m increases, the total sleep time and thus the idle ratio are decreasing and reaches the minimal at $m = m_{max} = 21$. The final latency is $310ms$ compared with $622ms$ by the approach in [Zhang and Chatha 2010], a 50% reduction.

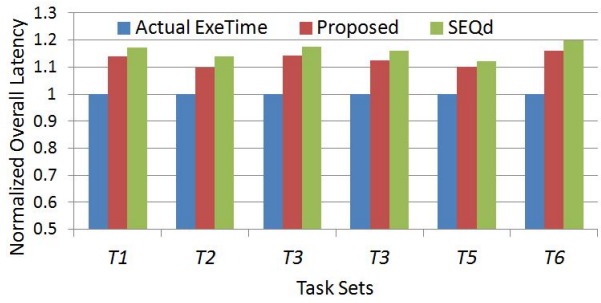
We next validate Theorem 5.3 by running a *cool* task τ_2 ($T_{ss} = 81.7^oC$) followed by a *hot* task τ_3 ($T_{ss} = 115.9^oC$) without introducing any sleep interval in between. The execution time of the cool task and hot task are $700ms$ and $400ms$, respectively. By using the proposed task switching method, the critical temperature, i.e. T_c, is plotted in Figure 4(b). We can see that as m increases from 1 to 30 (an arbitrarily chosen large number), the critical temperature first drops sharply and then increases slowly. The upper bound of T_c's decreasing region is calculated by using equation (25), i.e. 7, in this case. Again, the results conform with Theorem 5.3 that T_c is a monotonically decreasing with m when m is less then the *bound*. From Figure 4(b) we can also see that $T_c(2,3,m)$ drops drastically at the first a few steps and becomes relatively stable when m is further increased. It implies that, in this particular case, m_{opt} can be found within 3 rounds of evaluation (since $T_{max} = 100^oC$ and $T_c(2,3,3) < 100^oC$).

6.2. Latency Minimization for synthetic task sets

In this subsection, we evaluate the performance of the proposed method in terms of latency minimization. We randomly created 10 task sets each consisting of 20 randomly generated tasks with execution time (at the highest speed level) uniformly distributed between $[100, 1000ms]$. For simplicity reason, we assume that the *activity factor* μ is equal to its leakage factor ξ, which is evenly distributed between $[0.4, 1]$. 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.



(a) Processor without DVFS



(b) Processor with DVFS

Fig. 6. Simulation Results: Latency Comparisons for real benchmarks of different scheduling approaches

For processor without DVFS, we compare our approach with a heuristic similar to the one proposed in [Zhang and Chatha 2010], i.e. SEQ_s. In this case, we set the processor to the highest available speed level. We recorded the average total latency achieved by both methods for 10 test cases and plotted in Figure 5(a). We also provide the actual time that the processor spends in task execution as reference. As shown in Figure 5(a), the proposed method consistently outperforms SEQ_s in all task sets. Compared with SEQ_s, on average our method reduces the latency by 23.3% (up to 35.7%) and reduces the sleep time by 83.2%. From Figure 5(a), we can see that, in terms of overall latency, the performance of the proposed approach is relatively stable compared with SEQ_s approach which is sensitive to the amount of cool tasks available in a given task set. Since in SEQ_s, if the number of cool tasks are small, one has to insert large sleep period before each hot task to reduce the temperature. Under the same circumstance, in contrast, the proposed approach can rely on sleep distribution to reduce the latency even without *cool* tasks.

Then, we repeat the entire procedure for a DVFS enabled processor and compare our DVFS scheduling approach with the one similar to SEQ_d [Zhang and Chatha 2010] (a DVFS approach as well). As shown in Figure 5(a), the proposed method still achieves much better performance. Specifically, compared with SEQ_d, on average, our method reduces the latency and idle time by 5.3% and 46.9%, respectively. We notice that, given a processor with DVFS capability, the margins of the performance difference between our approach and SEQ_d are significantly reduced comparing with the ones for a processor with single active state. The reason is that, with the support of voltage scaling, the original hot tasks (at the maximal speed) have a great chance to become cool tasks after speed reduction. As a result, for SEQ_d approach, instead of inserting large chunk of sleep period, only a linear increase of execution time is incurred.

6.3. Latency Minimization for real benchmarks

In this subsection, we evaluate the performance of the proposed scheduling algorithms in terms of latency minimization by using real benchmark programs. Specifically, we choose 12 different benchmark programs from *MediaBench* [MediaBench 1997] and *SPEC2000* benchmark suites [SPEC 2000]. The total execution cycle/time of the benchmark is individually obtained by *Simplescalar* [Simplescalar 2004].

Table I. Representative task set

Task Set	Benchmarks	♯ of HotTasks	♯ of CoolTasks
T1	galgel,bzip2,crafty,gap,gcc,mcf,twolf,crc,dijkstra,ffti,gsm,qsort	5	7
T2	bzip2,crafty,gap,crc,dijkstra,ffti,gsm,qsort	5	3
T3	galgel,bzip2,crafty,gap,dijkstra,ffti,gsm,qsort	4	4
T4	qsort,ffti,dijkstra,bzip2,galgel,gcc,twolf,mcf	3	5
T5	ffti,qsort,bzip2,crafty,gap,gcc,mcf,twolf	2	6
T6	crc,dijkstra,ffti,gsm,qsort	5	0

We used *Wattch* [Brooks et al. 2000] to get the averaged power consumption of each program and then normalize to the highest one to get the *activity factor* μ of each task.

In our experiment, 6 representative task sets, i.e. $T1-T6$, were tested. Specifically, Task set $T1$ includes all 12 tasks we selected. Task set $T6$ contains only hot tasks. Task set $T2-T5$ each consists of 8 tasks with different hot/cool task ratios.

For each task set, the overall latency achieved by using the proposed methods and the approaches in [Zhang and Chatha 2010] were collected for processors with and without DVFS capability. We normalized the overall latency obtained by different methods to the actual execution time of each individual task set and plotted the results in Figure 6(a) and 6(b).

From Figure 6(a) and 6(b), we can see that without surprise, the results obtained from real benchmark programs are similar to those of synthetic benchmarks with and without considering DVFS. The proposed algorithms again outperform the base line approaches, i.e. SEQ_s and SEQ_d. For non-DVFS case, compared with SEQ_s, our method reduces the overall latency by 14.4% on average (up to 21% for $T6$ consisting of all hot tasks). On the other hand, for processor with DVFS capability, our method achieves 3% latency reduction on average.

6.4. 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 the 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 a step size of 10^oC. Under each ambient temperature condition, we generated 10 task sets each including 100 tasks. The average feasibility ratios (number of feasible tasks divided by 100) achieved by the proposed method and SEQ_s are recorded and plotted in Figure 7. The proposed method completes all tasks without infeasible task and thus improves the feasibility ratio by 21% on average compared with SEQ_s.

7. CONCLUSIONS

We study the problem on how to maximize the throughput of a periodic real-time system under a 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 be either in active or sleep mode. By judiciously selecting tasks with different thermal characteristics as well as alternating the processor's active/sleep mode, the sleep period required to cool down the processor is kept at a near optimum level and as the result, the throughput is maximized. We further extend this approach for processors with dynamic voltage/frequency scaling (DVFS) capability. Our experiments on large amount of synthetic as well as real benchmark programs show that the proposed methods not only consistently outperform the existing approaches in terms of throughput maximization, but also significantly improve the feasibility of tasks when a more stringent temperature constraint is imposed.

REFERENCES

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

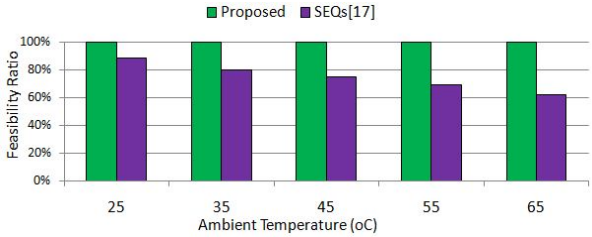


Fig. 7. Simulation Results:Feasibility Comparisons of different scheduling approaches

BANSAL, N., KIMBREL, T., AND PRUHS, K. 2007. Speed scaling to manage energy and temperature. *Journal of the ACM 54,* 1, 1–39.

BAO, M., ANDREI, A., ELES, P., AND PENG, Z. 2009. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *Design Automation Conference*. 490–495.

BAO, M., ANDREI, A., ELES, P., AND PENG, Z. 2010. Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In *DATE*. 21 – 27.

BROOKS, D. AND MARTONOSI, M. 2001. Dynamic thermal management for high-performance microprocessors. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*. 171.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*. 83–94.

CHANTEM, T., HU, X. S., AND DICK, R. 2009. Online work maximization under a peak temperature constraint. In *ISLPED*. 105–110.

CHATURVEDI, V., HUANG, H., AND QUAN, G. Leakage aware scheduling on maximal temperature minimization for periodic hard real-time systems. In *ICESS*. 1802–1809.

CHEN, J.-J., HSU, H.-R., AND KUO, T.-W. 2006. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS*. 408–417.

COSKUN, A., AYALA, J., ATIENZA, D., ROSING, T., AND LEBLEBICI, Y. 2009. Dynamic thermal management in 3d multicore architectures. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.* 1410 –1415.

HANUMAIAH, V., RAO, R., VRUDHULA, S., AND CHATHA, K. S. 2009a. Throughput optimal task allocation under thermal constraints for multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference*. DAC '09. ACM, New York, NY, USA, 776–781.

HANUMAIAH, V., VRUDHULA, S., AND CHATHA, K. 2009b. Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*. 310 –313.

HUANG, H. AND QUAN, G. 2011. Leakage aware energy minimization for real-time systems under the maximum temperature constraint. In *DATE*.

HUANG, H., QUAN, G., AND FAN, J. 2010. Leakage temperature dependency modeling in system level analysis. In *ISQED*. 447–452.

JAYASEELAN, R. AND MITRA, T. 2008. Temperature aware task sequencing and voltage scaling. In *ICCAD*. 618 – 623.

JEJURIKAR, R., PEREIRA, C., AND GUPTA, R. 2005. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. *DAC*, 111 – 116.

LIAO, W., HE, L., AND LEPAK, K. 2005. 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.

LIU, S., QIU, M., GAO, W., TANG, X.-J., AND GUO, B. 2010a. Hybrid of job sequencing and DVFS for peak temperature reduction with nondeterministic applications. In *ICESS*. 1780–1787.

LIU, S., ZHANG, J., WU, Q., AND QIU, Q. 2010b. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*. 390 –398.

LIU, Y. AND YANG, H. 2010. Temperature-aware leakage estimation using piecewise linear power models. *IEICE transactions on electronics 93,* 12, 1679–1691.

LIU, Y., YANG, H., DICK, R. P., WANG, H., AND SHANG, L. 2007. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In *ISQED*. 204–209.

LUNG, C.-L., HO, Y.-L., KWAI, D.-M., AND CHANG, S.-C. 2011. Thermal-aware on-line task allocation for 3d multi-core processor throughput optimization. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*. 1 –6.

MEDIABENCH. 1997. Mediabench, http://euler.slu.edu/ fritts/mediabench/.

POWELL, M. D., GOMAA, M., AND VIJAYKUMAR, T. N. 2004. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. In *ASPLOS*. 260–270.

QUAN, G. AND CHATURVEDI, V. 2010. Feasibility analysis for temperature-constraint hard real-time periodic tasks. *IEEE Transaction on Industrial Informatics 6,* 3, 329–339.

QUAN, G. AND ZHANG, Y. 2009. Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks. *ECRTS*, 207–216.

QUAN, G., ZHANG, Y., WILES, W., AND PEI, P. 2008. Guaranteed scheduling for repetitive hard real-time tasks under the maximal temperature constraint. *ISSS+CODES*.

SANTARINI, M. 2005. Thermal integrity: A must for low-power ic digital design. *EDN*, 37–42.

SIMPLESCALAR. 2004. Simplescalar, http://www.simplescalar.com.

SKADRON, K., STAN, M., HUANG, W., VELUSAMY, S., SANKARANARAYANAN, K., AND TARJAN, D. 2003. Temperature-aware microarchitecture. *ICSA*, 2–13.

SPEC. 2000. Spec2000 benchmarks, http://www.spec.org.

SUN, C., SHANG, L., AND DICK, R. 2007. Three-dimensional multiprocessor system-on-chip thermal optimization. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*. 117 –122.

WANG, S. AND BETTATI, R. 2006. Reactive speed control in temperature-constrained real-time systems. *ECRTS*, 161–170.

WIKIPEDIA. 2013. L'hopital's rule, http://en.wikipedia.org/wiki/l'hopital's-rule.

YANG, C.-Y., CHEN, J.-J., THIELE, L., AND KUO, T.-W. 2010. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *DATE*. 9–14.

YEH, L.-T. AND CHU, R. C. 2002. *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices*. ASME Press, New York, NY.

YUAN, L., LEVENTHAL, S., AND QU, G. 2006. Temperature-aware leakage minimization technique for real-time systems. In *ICCAD*. 761–764.

ZHANG, S. AND CHATHA, K. S. 2007. Approximation algorithm for the temperature-aware scheduling problem. In *ICCAD*. 281–288.

ZHANG, S. AND CHATHA, K. S. 2010. Thermal aware task sequencing on embedded processors. In *DAC*. 585 – 590.

ZHOU, X., XU, Y., DU, Y., ZHANG, Y., AND YANG, J. 2008. Thermal management for 3d processors via task scheduling. In *Parallel Processing, 2008. ICPP '08. 37th International Conference on*. 115 –122.

ZHU, C., GU, Z., SHANG, L., DICK, R., AND JOSEPH, R. 2008. Three-dimensional chip-multiprocessor run-time thermal management. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27,* 8, 1479 –1492.

Received October 2012; revised March 2013; accepted June 2013