Procrastination Determination for Periodic Real-Time Tasks in Leakage-Aware Dynamic Voltage Scaling Systems*

Jian-Jia Chen
Department of Computer Science and Information Engineering
National Taiwan University,Taiwan
r90079@csie.ntu.edu.tw

Tei-Wei Kuo
Department of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia,
National Taiwan University, Taiwan
ktw@csie.ntu.edu.tw

ABSTRACT

Many computing systems have adopted the dynamic voltage scaling (DVS) technique to reduce energy consumption by slowing down operation speed. However, the longer a job executes, the more energy in leakage current the processor consumes for the job. To reduce the power/energy consumption from the leakage current, a processor can enter the dormant mode. Existing research results for leakage-aware DVS scheduling perform procrastination of real-time jobs greedily so that the idle time can be aggregated as long as possible to turn off the processor. This paper proposes algorithms for the procrastination determination of periodic real-time tasks in uniprocessor systems. Instead of greedy procrastination, the procrastination procedures are applied only when the evaluated energy consumption is less than not procrastination. Evaluation results show that our proposed algorithms could derive energy-efficient solutions and outperform existing algorithms.

Keywords: Energy-aware systems, Scheduling, Leakage-aware scheduling, Dynamic voltage scaling, Job procrastination.

1. INTRODUCTION

With the advanced technology of VLSI circuit designs, many modern processors can operate at different supply voltages. Technologies, such as Intel SpeedStep® and AMD PowerNOW!™, provide dynamic voltage scaling (DVS) for laptops to prolong battery lifetime. Different supply voltages lead to different execution speeds on a dynamic voltage scaling processor. The power consumption of processors in dynamic voltage scaling is usually a convex and increasing function of processor speed [18]. The lower the processor speed is, the less the power consumption of the dynamic voltage scaling is. However, executing a task at a lower processor speed stretches its execution time, and, hence, the energy consumption resulting from leakage current increases [10].

For real-time systems, *energy-efficient* task scheduling is to minimize the energy consumption while completing all tasks in time. In the past decade, energy-efficient task scheduling with various deadline constraints received much attention. Recently, researchers have started exploring energy-efficient scheduling with the considerations of leakage current since the power consumption resulting from leakage current is comparable to the dynamic power dissipation [4, 6, 8–10, 12, 16]. To reduce the energy consumption resulting from leakage current, a processor might be turned off (to enter a dormant mode). When the processor is turned to the dormant mode, the power consumption of the processor can be treated as negligible. However, turning on the processor requires time and energy overheads, resulting from the wakeup/shutdown of the processor and data fetch in the register/cache. For example, the Transmeta processor with the 70nm technology has $483\mu J$ energy overhead and less than 2 millisecond timing overhead [9, 10]. For some practical real-time applications, the periods of tasks are generally in the range of 10 milliseconds and 125 milliseconds.

*Support in parts by research grants from ROC National Science Council NSC-96-2752-E-002-008-PAE and Excellent Research Projects of National Taiwan University, 96R0062-AE00-07.

For non-DVS systems with dormant states, Baptiste [3] proposed an algorithm based on dynamic programming to determine when to turn on/off the processor when aperiodic real-time jobs with the same execution time are considered. For multiple idle states, Augustine et al. [1] determined the state that the processor should enter for aperiodic real-time jobs and proposed a competitive algorithm for on-line use.

Leakage-aware scheduling has been recently explored on DVS platforms, such as [4, 8–10, 12]. In particular, Jejurikar et al. [10] and Lee et al. [12] proposed energy-efficient scheduling on a uniprocessor by procrastination scheduling to decide when to turn off the processor. Jejurikar and Gupta [9] then further considered real-time tasks that might complete earlier than its worst-case estimation by extending their previous work [10]. Fixed-priority scheduling was also considered by Jejurikar and Gupta [8] and Chen and Kuo [4]. For uniprocessor scheduling of aperiodic real-time tasks, Irani et al. [6] proposed a 3-approximation algorithm for the minimization of energy consumption with the considerations of leakage current on a uniprocessor DVS system with continuous available speeds. Niu and Quan [16] applied similar procrastination strategies for periodic real-time tasks with leakage considerations under different procrastination interval calculations by expanding all the jobs in the hyper-period of the given tasks. The basic idea behind the above results is to greedily procrastinate the execution of the real-time jobs as long as possible so that the idle interval is long enough to reduce the energy consumption. However, greedy procrastination at this moment might sacrifice the possibility to turn off the processor in the near future, and, hence, might consume more energy.

This paper considers energy-efficient scheduling of periodic real-time tasks in a uniprocessor leakage-aware DVS system, in which the processor might be turned off. As shown in the literature, e.g., [4, 10], there is a *critical speed* in the available processor speeds with the minimum energy consumption for execution. We apply existing DVS scheduling algorithms to determine the execution speeds of tasks in an off-line manner by treating the critical speed as the minimum available speed, e.g., [2, 15]. On-line algorithms are proposed to determine when to turn off/on the processor, and when to execute jobs at speeds lower than the critical speed. When the processor is to be turned off, we have to procrastinate the execution of real-time jobs arriving in the future so that energy consumption can be reduced under the timing constraints. Distinct from greedy procrastination algorithms in the literature [4, 6, 8–10, 12, 16], we develop novel algorithms for procrastination determination of periodic real-time tasks in uniprocessor systems. Procrastination of jobs is applied only when the evaluated energy consumption is less than not procrastination. Moreover, our algorithms can reduce the energy consumption by executing jobs at lower speeds than the critical speed when the processor is decided not to be turned off. To our best knowledge, this is the first approach that might execute jobs at speeds lower than the critical speed with energy reduction. A series of evaluations are conducted, and the results show that our algorithms outperform the algorithm proposed by Jejurikar et al. [10].

The rest of this paper is organized as follows: Section 2 defines the leakage-aware energy-efficient scheduling problem. Section 3 presents

a motivational example. The proposed algorithms are in Section 4. Experimental results for the performance evaluation of the proposed algorithms are presented in Section 5. Section 6 is the conclusion.

2. SYSTEM MODELS

2.1 Task models

This paper explores periodic real-time tasks that are independent in their execution. A periodic task is an infinite sequence of task instances, referred to as *jobs*. Each task τ_i is associated with its initial arrival time (denoted by a_i), its execution cycles (denoted by c_i), and its period (denoted by p_i). The relative deadline of each task τ_i is equal to its period p_i. Namely, the arrival time and deadline of the j-th job of task τ_i are $a_i + (j-1) \cdot p_i$ and $a_i + j \cdot p_i$, respectively. Let $\mathbf{T}$ be the set of input tasks, and n is the number of tasks.

2.2 Power consumption and execution models

The power consumption function $P(s)$ of speed s on a DVS processor can be divided into two parts: $P_d(s)$ and P_{ind}, where $P_d(s)$ is dependent and P_{ind} is independent on the speed [21]. Leakage power consumption mainly contributes to P_{ind}, while the dynamic power consumption resulting from the charging and discharging of gates on a CMOS DVS processor and the short-circuit power consumption contribute to $P_d(s)$. For example, in CMOS DVS processors [18], the power consumption $P_{switch}(s)$ due to gate switching at speed s is

$$P_{switch}(s) = C_{ef} V_{dd}^2 s, \tag{1}$$

where $s = \kappa \frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd}, and κ denote the effective switch capacitance, the threshold voltage, the supply voltage, and a design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, $\kappa > 0$, and $C_{ef} > 0$). The short-circuit power consumption is proportional to the supply voltage. As a result, the speed-dependent power consumption function $P_d(s)$ is a convex and increasing function of the adopted speed. The power consumption function can model many power consumption models in [18, §5.5]. If the leakage power consumption is related to the speeds/voltages, i.e., not a constant, the leakage power is divided into two parts that contribute to $P_d(s)$ and P_{ind} accordingly. In other words, $P_d(s)$ models the voltage-related power consumption while P_{ind} models the voltage-independent power consumption.

The number of CPU cycles executed in a time interval is assumed to be linear to processor speed, and the energy τ_i consumed at the processor speed s for t time units is $t \cdot P(s)$. For the rest of this paper, we only present the results on processors with a continuous spectrum of the available speeds between the upper-bounded speed $s_{\max}$ and the lower-bounded speed $s_{\min}$, while the results can be easily extended to processors with discrete speeds only by applying the results in [7, 11].

A processor here has two modes: *dormant mode* and *active mode*. When the processor is turned to the dormant mode (or is turned off), the power consumption of the processor can be treated as 0 by scaling the static power consumption [6]. To execute jobs, the processor has to be in the active mode. However, switching between the two modes takes time and consumes energy. Since periodic tasks are considered, the procedure to turn the processor to the dormant mode can be assumed to be done instantaneously with negligible energy overhead by treating the overhead as a part of the overhead to turn on the processor. We denote E_{sw} (t_{sw}, respectively) as the energy (time, respectively) of the *switching overhead* from the dormant mode to the active mode.

A processor is said *idle* at time t, if it does not execute any given task at time t. When the processor is idle in the active mode, the processor executes NOP instructions at processor speed $s_{\min}$ for energy minimization. When the processor is idle and the idle interval is longer than *break-even time* $\frac{E_{sw}}{P(s_{\min})}$, turning it to the dormant mode is worthwhile. For notational brevity, let t_θ be the break-even time, i.e., $t_\theta = \frac{E_{sw}}{P(s_{\min})}$. Without loss of generality, we assume that t_{sw} is no more than t_θ.

2.3 Scheduling policy

The earliest-deadline-first (EDF) scheduling algorithm is an optimal uniprocessor scheduling algorithm for independent real-time tasks [13]. A set of tasks is schedulable by EDF if and only if the total utilization of the set of tasks is no more than 100%, where the *utilization* of a task is defined as its execution time divided by its period. Hence, if executing every task at speed $s_{\max}$ is with total utilization greater than 100%, there does not exist any feasible schedule for such an input instance. For the rest discussion, we only consider input instance $\mathbf{T}$ whose utilization is less 1 by executing all tasks at speed $s_{\max}$, i.e., $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i s_{\max}} \leq 1$.

Critical speeds. The concept of the critical speed has been adopted in the literature e.g., [4, 6, 9]. The critical speed s^* is defined as the available speed to execute a cycle with the minimum energy consumption. Although $P(s)$ is a convex and increasing function of speed s, the energy consumption $\frac{P(s)}{s}$ to execute a cycle at speed s is merely a convex function of s. By solving equation $\frac{\mathrm{d}(P(s)/s)}{\mathrm{d}s} = 0$, we can derive the critical speed of the processor. For example, when $P(s) = s^3 + \beta$, the function $P(s)/s$ is minimized when s is $\sqrt[3]{\frac{\beta}{2}}$. Suppose that $\hat{s}$ is the speed s with the minimum $P(s)/s$. Since the critical speed s^* has to be in the range of $[s_{\min}, s_{\max}]$, we know that s^* is $\min\{\max\{\hat{s}, s_{\min}\}, s_{\max}\}$.

Without loss of generality, the critical speed s^* is normalized to 1, and the execution cycles of tasks, switching overheads, available speeds are also normalized according to speed s^*. For example, the execution time to execute task τ_i at the critical speed is c_i after normalization.

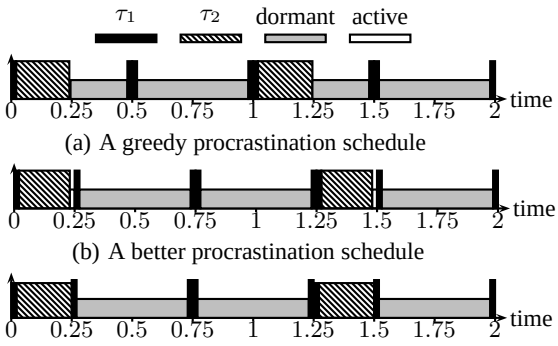
Minimization of the execution energy consumption. After determining the critical speed of the processor, we can apply existing DVS algorithms to assign the execution speeds for tasks, e.g., [2,5,7,11, 15, 20], by treating the critical speed as the minimum available speed. For periodic real-time tasks on processors with continuous available speeds, the algorithm proposed by Aydin et al. [2] can be applied by setting the minimum available speed as s^*. The algorithms in [5, 15] can be applied when the processor has discrete speeds only.

3. A MOTIVATIONAL EXAMPLE

This section presents an example to demonstrate the shortcoming of existing algorithms in [6, 10, 16] on the procrastination of real-time tasks. Suppose that there are two tasks arriving at time 0, in which $p_1 = 0.25$, $c_1 = \epsilon$, $p_2 = 1$, and $c_2 = 0.25 - 2\epsilon$ with a small and positive number ϵ. The power consumption function here is assumed as $P(s) = 2 + s^3$, and speed $s_{\min}$ is 0.5. The critical speed is 1. The energy switching overhead E_{sw} is 0.25, while time switching overhead t_{sw} is less than 0.1. The break-even time is $0.25/(2 + 0.5^3) = 0.118$. We assume that the processor is in the active mode at time 0.

To minimize the energy consumption for task execution, both of these two tasks would be executed at the critical speed s^*. The processor becomes idle at time $0.25 - \epsilon$ after executing the first task instances of task τ_1 and task τ_2. The latest starting time of the second task instance of task τ_1 is $0.5 - \epsilon$. By applying the greedy procrastination algorithms in [6, 10, 16], we can turn the processor off at time $0.25 - \epsilon$, and then turn it on so that the processor is ready for execution at time $0.5 - \epsilon$. Similarly, the processor would be in the dormant mode in time interval $(0.5 + \epsilon, 1 - \epsilon]$. The schedule in time interval $(0, 1]$ is repeated. Figure 1(a) shows the schedule in time interval $(0, 2]$. The energy consumption for task execution in time interval $(0, 2]$ is $6(0.25 - \epsilon)$, and the energy consumption while the processor is idle is 1.

Figure 1(b) provides a better schedule for procrastination. At time $0.25 - \epsilon$, instead of procrastinating the execution of the second task instance of task τ_1, the processor is idle in the active mode for ϵ time unit, and then executes the job arriving at time 0.25. In such a schedule, we sacrifice the possibility to turn the processor to the dormant mode instantly at time $0.25 - \epsilon$, and then have a longer idle interval in the near future, i.e., $(0.25 + \epsilon, 0.75 - \epsilon]$, to turn the processor to the dormant mode. Similarly, at time $1.5 - \epsilon$, the processor is idle in the active



(a) A greedy procrastination schedule

(b) A better procrastination schedule

(c) A better procrastination schedule with speed reduction

Figure 1: A motivational example.

mode for ϵ time unit, and then executes the job arriving at time 1.5 to create longer time interval, i.e., $(1.5+\epsilon, 2-\epsilon]$, to turn the processor to the dormant mode. Then, the above schedule in time interval $(0, 2]$ is repeated. The energy consumption while the processor is idle in the schedule in time interval $(0, 2]$ is $0.75 + 2\epsilon (2.125)$.

When ϵ is very small, the schedule in Figure 1(b) reduces 25% energy consumption while the processor is idle. The ratio of the energy consumption of the schedule in Figure 1(b) to that in Figure 1(a) is $\frac{2(3(0.25-\epsilon))+0.75}{2(3(0.25-\epsilon)+0.5)} \approx 0.9$. The reason why the greedy procrastination algorithms in [6, 10, 16] perform worse in energy efficiency is because turning the processor off might result in a schedule that turns the processor off more frequently, and, hence, might consume more energy. Since the processor is idle in the active mode in time intervals $[0.25-\epsilon, 0.25]$ and $[1.5-\epsilon, 1.5]$ in the schedule in Figure 1(b), we can slow down the execution in time intervals $(0, 0.25]$ and $(1.25, 1.5]$ to further reduce the energy consumption as shown in Figure 1(c).

4. THE PROPOSED ALGORITHMS

This section presents our proposed algorithms. We assume that the speed, i.e., the supply voltage, of each task has been determined by applying existing DVS algorithms to assign the execution speeds for tasks, e.g., [2, 5, 7, 11, 15, 20], in which the critical speed is treated as the minimum available speed. Let s_i be the determined execution speed for task τ_i after applying a DVS algorithm, where $s_i \geq s^*$. For processors with continuous speeds in a specified range, all the tasks in $\mathbf{T}$ are executed at a common speed [2]. Procrastination algorithms are used to turn the processor on/off on the fly.

We will present two different approaches. The first approach is similar to the greedy algorithm in [10] with an additional parameter for the determination of procrastination. The second approach is based on simulated scheduling to evaluate whether turning the processor to the dormant mode at instant is worthwhile. If it is not worthwhile, tasks might exploit the idle time for slowing down for energy saving.

4.1 Parametric Procrastination

When a job arrives, it is inserted into the ready queue. When the processor is in the active mode, if the ready queue is not empty, the scheduler executes the jobs in the ready queue in the earliest-deadline-first order by adjusting the processor speed to satisfy their execution speeds determined in off-line. If a job completes at time t, and the ready queue is empty, we have to decide whether the processor should be turned off or idle at speed $s_{\min}$ in the active mode. At time t, we first evaluate how long we can turn the processor to the dormant mode without making the incomplete task instances miss their deadlines. The procedure can be done in $O(n)$ by applying the approach in [10].

The parametric procrastination here does not use a new method to calculate the interval length for procrastination. The parametric procrastination only decides whether we should turn the processor to the dormant mode at time t. If we decide to turn the processor to the dormant mode, we should use algorithms in [6, 10, 16] to decide when to turn on the processor. Otherwise, the processor is idle in the active mode to serve the job arriving in the near future. For efficiency considerations, we adopt the calculation of *procrastination length* in [10] for the rest of this paper. (The procrastination length is denoted by procrastination interval in [10].) The calculation of procrastination length in [10] is as follows: First, let tasks be sorted by their periods such that $p_i \leq p_j$ if $i < j$. Then, the procrastination length Z_i of task τ_i is $p_i(1-\sum_{j=1}^{i} \frac{c_j}{s_j p_j})$. For example, the procrastination lengths of τ_1 and τ_2 in the example in Section 3 are $0.25-\epsilon$, and $0.75-2\epsilon$, respectively.

Algorithm 1 : P-Procrastination

Input: α;

On arrival of a job of task τ_i:

1: insert the job to the ready queue;
2: **if** the processor is in the active mode **then**
3: schedule the job with the earliest deadline in the ready queue;
4: **end if**

On completion of a job at time t:

1: remove the completed job from the ready queue;
2: **if** there are jobs in the ready queue **then**
3: schedule the job with the earliest deadline in the ready queue;
4: **else**
5: $r_{i,t} \leftarrow a_i + p_i \cdot \left\lceil \frac{t-a_i}{p_i} \right\rceil$ for every task τ_i in $\mathbf{T}$;
6: $W_t \leftarrow \min_{\tau_i \in \mathbf{T}}\{r_{i,t} + Z_i\}$;
7: **if** $\min_{\tau_i \in \mathbf{T}} r_{i,t} - t + \alpha(W_t - \min_{\tau_i \in \mathbf{T}} r_{i,t}) \geq t_\theta$ **then**
8: turn the processor to the dormant mode at time t, and set a timer to start to turn the processor to the active mode at time $W_t - t_{sw}$;
9: **end if**
10: **end if**

On turning the processor to the active mode at time t:

1: schedule the highest-priority job in the ready queue;

Let $r_{i,t}$ be the arrival time of the next job of task τ_i in $\mathbf{T}$ arriving after time instant t, i.e., $r_{i,t} = a_i + p_i \cdot \left\lceil \frac{t-a_i}{p_i} \right\rceil$. Let W_t be $\min_{\tau_i \in \mathbf{T}}\{r_{i,t} + Z_i\}$, which is the time moment at which the processor should be in the active mode for task execution after time t. If we turn the processor to the dormant mode at time t, we have to turn on the processor before time $W_t - t_{sw}$ so that no job will miss its deadline. The greedy procrastination algorithm in [10] turns the processor to the dormant mode greedily when $W_t - t > t_\theta$.

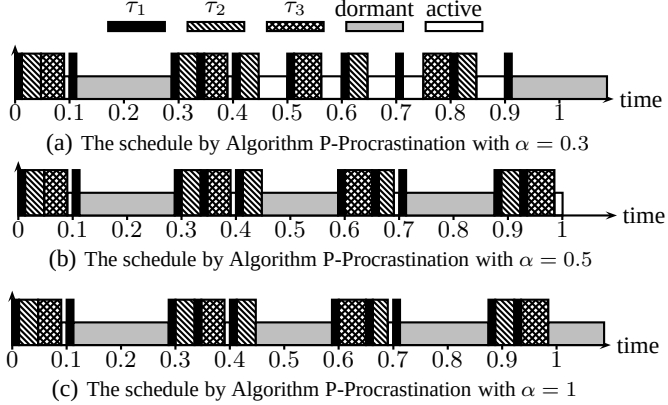
Instead of greedy procrastination, our algorithm, denoted by Algorithm P-Procrastination for parametric procrastination, divides the time interval $(t, W_t]$ into two disjoint parts: the *residual interval* and the *procrastination interval*. The residual interval is the time interval between t and the earliest arrival time of jobs after time t. That is, the residual interval is $(t, \min_{\tau_i \in \mathbf{T}} r_{i,t}]$. The procrastination interval is the time interval from the earliest arrival time of jobs after time t to W_t.

Since the processor must be idle in the residual interval, we should turn the processor to the dormant mode if the length of the residual interval is greater than the break-even time t_θ. However, the use of the procrastination interval for turning the processor to the dormant mode at that instant might divide a longer dormant interval into two dormant intervals, and, hence, consume more energy. If we do not turn the processor to the dormant mode, the time units in the procrastination interval might be used for energy reduction in the near future. The idea behind Algorithm P-Procrastination is to use a parameter to control the portion of the procrastination interval to evaluate whether the processor should go to the dormant mode at time t. Let α be a parameter specified by users with $0 \leq \alpha \leq 1$. The processor enters the dormant mode if

$$\min_{\tau_i \in \mathbf{T}} r_{i,t} - t + \alpha(W_t - \min_{\tau_i \in \mathbf{T}} r_{i,t}) \geq t_\theta. \quad (2)$$

Algorithm 1 shows the pseudo-code of Algorithm P-Procrastination. The time complexity for determining procrastination at time t is $O(n)$. Clearly, setting α as 1 makes Algorithm P-Procrastination as the same as the algorithm in [10] for task procrastination.

For understanding the algorithm, we use the following example for explanation. Suppose that there are three tasks arriving at time 0 in the system in which $p_1 = 0.1, p_2 = 0.2, p_3 = 0.25$, $c_1 = 0.0125$, $c_2 = 0.035$, and $c_3 = 0.05$. The power consumption function in this

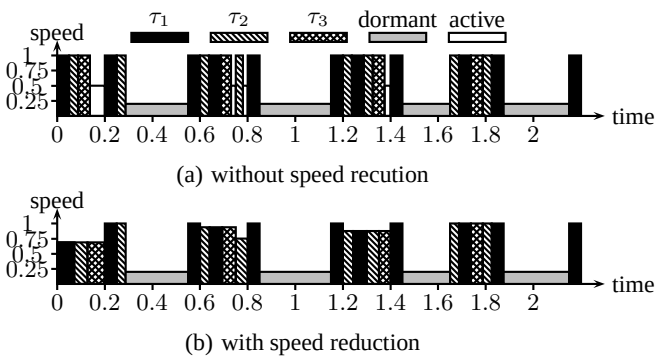


(a) The schedule by Algorithm P-Procrastination with $\alpha = 0.3$

(b) The schedule by Algorithm P-Procrastination with $\alpha = 0.5$

(c) The schedule by Algorithm P-Procrastination with $\alpha = 1$

Figure 2: An example for Algorithm P-Procrastination.



(a) without speed recution

(b) with speed reduction

Figure 3: An example for simulated-scheduling procrastination.

example is assumed as $P(s) = 2 + s^3$, and the minimum available speed $s_{\min}$ is 0.5. The energy of switching overhead E_{sw} is 0.2. The break-even time is $0.2/(2+0.5^3) = 0.0941$. The procrastination length of tasks τ_1, τ_2, and τ_3 are $0.0875, 0.14$, and 0.125, respectively.

Figure 2(a), Figure 2(b), and Figure 2(c) show the resulting schedules for the example when the parameter α is set as $0.3, 0.5$, and 1, respectively. These three schedules are the same before time 0.45. At time 0.45, the length of the residual interval is 0.05 and the length of the procrastination interval is 0.0875. When α is 0.3, $0.05+0.3\cdot 0.0875$ is less than the break-even time, and, hence, the processor remains active. At time 0.9125, since the lengths of the residual interval and the procrastination interval are both 0.0875, the value $0.0875 + 0.3 \cdot 0.0875$ is greater than the break-even time, and, hence, the processor is turned to the dormant mode at time 0.9125. When α is 0.5 or 1, at time 0.45, the processor is determined to be turned to the dormant mode. At time 0.985, since $0.015 + 0.5 \cdot 0.0875$ is less than the break-even time, the schedule in Figure 2(b) makes the processor remain in the active mode. Since $0.015 + 0.0875$ is greater than the break-even time, the processor enters the dormant mode at time 0.9875 when α is 1 as shown in Figure 2(c). The energy consumptions while the processor is idle in the schedules in Figure 2(a), Figure 2(b), and Figure 2(c) for 100 time units are about $72.06, 64.78$, and 80.86, respectively.

4.2 Simulated-Scheduling Procrastination

Another approach on the determination of whether we should turn the processor to the dormant mode is to evaluate whether it is worthwhile by simulated scheduling. Suppose that we perform simulated scheduling to derive virtual EDF schedules at time t^*, which is 0 at initialization. Based on the virtual EDF schedules, we would like to know the best (earliest) moment to turn the processor to the dormant mode when the system is idle. We use the concept of the *effective idle power consumption* to determine when to turn the processor to the dormant mode. The effective idle power consumption in a time interval is defined as the energy consumption when the processor is idle divided by the length of the idle intervals in the specified time interval. That is, if the processor is idle in the active mode for w_1 time units and in the dormant mode for w_2 time units with mode switching for k times in a specified time interval, the effective idle power consumption in the time interval is $\frac{P(s_{\min})w_1+k\cdot E_{sw}}{w_1+w_2}$.

The simulated-scheduling procrastination simulates two schedules to determine when to turn off the processor. The first one is to simulate the schedule by applying the greedy procrastination. Suppose that t' is the earliest time instant after time t^* with $W_{t'} - t' > t_\theta$ by applying the greedy procrastination, and t'' is the earliest time instant after $W_{t'}$ when the system is idle in the virtual schedule, where $W_{t'}$ is defined in Section 4.1 as the time instant that the processor must be in the active mode when the processor is turned off at time t'. If the system is turned to the dormant mode at time instant t'' in the virtual schedule, the effective idle power consumption of the virtual schedule is $\frac{2E_{sw}}{W_{t''}-t''+W_{t'}-t'}$ in time interval $(t', W_{t''}]$; otherwise, the effective idle power consumption of the virtual schedule is $\frac{E_{sw}+P(s_{\min})(R-t'')}{(R-t'')+W_{t'}-t'}$ in time interval $(t', R]$, where R is the earliest time instant at which a job arrives after time t''.

The second virtual schedule is to simulate scheduling to find the earliest time instant $\hat{t}$ after t^* to turn off the processor such that the effective idle power consumption in time interval $(t', W_{\hat{t}}]$ is less than that in the first virtual schedule, where $W_{\hat{t}}$ is defined in Section 4.1. If $\hat{t}$ exists in a specified length for simulated scheduling, the actual) schedule would not turn the processor to the dormant mode in time interval $(t^*, \hat{t}]$, but turn off the processor at time $\hat{t}$ with the update of t^* to $W_{\hat{t}}$; otherwise, the greedy procrastination is applied at time t', and t^* is updated to $W_{t'}$.

The following example demonstrates the simulated-scheduling procrastination. Suppose that there are three tasks in the system in which $p_1 = 0.2, p_2 = 0.25, p_3 = 0.5$, $c_1 = 0.05$, $c_2 = 0.0375$, and $c_3 = 0.05$. The power consumption function in this example is assumed as $P(s) = 2 + s^3$, and the minimum available speed $s_{\min}$ is 0.5. The energy of switching overhead E_{sw} is 0.2. Suppose that t^* is 0. Then, t' is 0.1375 with $W_{t'}$ as 0.35 and t'' is 0.4875 with $W_{t''}$ as 0.65 so that the effective idle power consumption in time interval $(0.1375, 0.65]$ in the first schedule is 1.067. The effective idle power consumption in time interval $(0.1375, 0.55]$ in the second schedule is 1.024 with $\hat{t} = 0.2875$, and, hence, the processor is idle in the active mode in time interval $(0.1375, 0.2]$. Figure 3(a) shows the resulting schedule by applying the simulated-scheduling procrastination with 4.034 total energy consumption in time interval $(0, 2.15]$, while the total energy consumption by applying the greedy procrastination is 4.2 in time interval $(0, 2.15]$ by turning the processor off for 6 times.

Moreover, we also know that the processor is in the active mode in time interval $(t^*, \hat{t})$. We can simply ignore the energy consumption $P_{ind}(\hat{t} - t^*)$ since it is a constant in the time interval. Suppose that $\mathbf{J}_{t^*,\hat{t}}$ is the set of jobs completed in the time interval. We can slow down the execution speeds of jobs in set $\mathbf{J}_{t^*,\hat{t}}$ to minimize the energy consumption contributed from the speed-dependent power consumption. The idea is as follows: (1) revise the deadline of a job in set $\mathbf{J}_{t^*,\hat{t}}$ as no later than $\hat{t}$ or its original deadline. (2) schedule these jobs in set $\mathbf{J}_{t^*,\hat{t}}$ by applying the speed determination algorithm for aperiodic real-time tasks by Yao et al. [20] with the revised deadlines. For example, in Figure 3(a), when $t^* = 0.1375$ and $\hat{t} = 0.2$, all the jobs arrived at time 0 are revised to deadline 0.2 and are executed at speed 0.6875. Figure 3(b) shows the resulting schedule with speed reduction with total energy consumption 3.881 in time interval $(0, 2.15]$. The simulated-scheduling procrastination with speed reduction is denoted as Algorithm SS-Procrastination. Let K be the number of jobs in time interval $(t', W_{t''}]$. The time complexity of Algorithm SS-Procrastination is $O(K \log^2 K)$, dominated by applying the algorithm by Yao et al. [20].

5. PERFORMANCE EVALUATION

This section provides performance evaluations of our algorithms with comparisons to the existing greedy procrastination algorithm [10], denoted by Algorithm Greedy Procrastination, for periodic real-time tasks.

5.1 Experimental Setup

The power consumption model used here is $P(s) = \beta_1 + \beta_2 s^3$. We evaluate our algorithms for Intel XScale, in which there are five speeds: $(0.15, 0.4, 0.6, 0.8, 1)$ GHz with corresponding power consumption $(80, 170, 400, 900, 1600)$ mWatt. The power consumption function can be modeled approximately as $P(s) = 0.08 + 1.52s^3$Watt by treating 1GHz as the speed unit. The critical speed for Intel XScale in such a model is 297 MHz with power consumption 0.12W. To make our presentation consistent, we normalize the speeds in $[0.15, 1]$ GHz by setting 0.297 GHz as 1. Hence, the power consumption $P(s)$ after normalization is $\frac{1}{25}(2 + s^3)$ Watt in our evaluations, where $\frac{1}{25}$ is called the *power normalization factor* denoted by γ. The minimum (maximum, respectively) available speed is about 0.5 (3.37, respectively).

We evaluate the performance of the proposed algorithms by using synthetic real-time tasks, which are also adopted mostly in many leakage-aware energy-efficient studies, such as, [10, 12, 19]. Based on real-life task sets [14] with periods in the scale of milliseconds, for any given task τ_i, period p_i is generated as a random variable in milliseconds. We assume the execution of each task at critical speed s^*. For each task τ_i, the estimated utilization μ_i^* is a random variable in $(0, 1]$. For a specified total utilization U of a set of tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$, the execution time t_i of task τ_i at s^* is set as $\frac{U}{\sum_{j=1}^{n} \mu_j^*} \mu_i^* p_i$, i.e., $c_i = \frac{U}{\sum_{j=1}^{n} \mu_j^*} \mu_i^* p_i$. If U is greater than 1, executing all the tasks at speed U is the optimal solution. Therefore, we only consider task sets with U no more than 1.

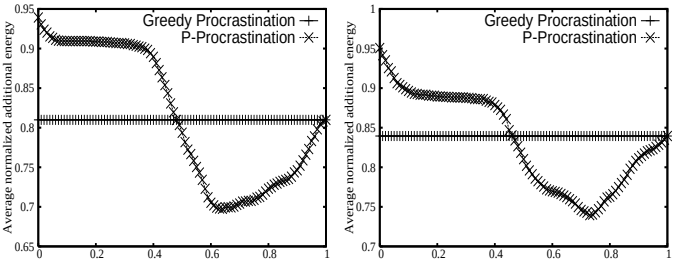
We perform many experimental settings but only representative results are presented. The first and second experiments simulate the effect on the selection of user parameter α in Algorithm P-Procrastination for specified task sets with different settings. The third one focuses on the effect of the number of tasks in the system with $U = 0.5$ and $E_{sw} = 8$ mJ, while the fourth one considers the impact of the utilization U at the critical speed for 20 tasks with E_{sw} as 8 mJ. The last one explores the evaluated algorithms by varying E_{sw}.

The baseline schedule for comparison is to apply the original EDF scheduling by executing jobs at the critical speed without procrastination and by turning off the system when the idle interval is long enough. The *normalized total energy* and *normalized additional energy* are adopted as the performance metrics. The normalized total energy of an algorithm for an input instance is the energy consumption of the derived solution divided by the energy consumption of the baseline schedule. The additional energy in a schedule is the total energy consumption subtracts the energy consumption to execute all the tasks at the critical speed. If all the tasks are executed at the critical speed, the additional energy consumption is the energy consumption while the processor is idle. The normalized additional energy of an algorithm for an input instance is the energy consumption of the derived solution divided by the additional energy of its baseline schedule. Each configuration is run for 256 times independently.

5.2 Experimental Results

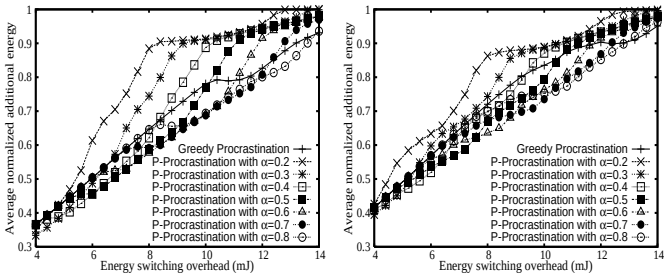
Figure 4 shows the evaluation results for the normalized additional energy of Algorithm P-Procrastination and Algorithm Greedy Procrastination by varying the parameter α from 0 to 1, stepped by 0.01, when the energy of switching overhead E_{sw} is 10mJ and there are 20 tasks. Figure 4(a) is the results for $U = 0.3$, while Figure 4(b) is for $U = 0.5$. As shown in Figure 4, with a proper setting of α, the improvement of Algorithm P-Procrastination compared to the greedy procrastination can be at most 12% in the energy consumption while the system is idle.

Figure 5 shows the evaluation results of Algorithm P-Procrastination for $\alpha = 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8$ by varying the parameter E_{sw} from 4mJ to 14mJ, when the number of tasks is 20. In Figure 5(a) and Figure 5(b), $\alpha = 0.3$ is a good choice when E_{sw} is between 4mJ and 5mJ, $\alpha = 0.4$ is good when E_{sw} is between 5mJ and 6mJ, and so on. When the energy of switching overhead increases, we have to choose a greater value of α to have better results. Since the break-even time is longer for greater E_{sw}, job procrastination to create long enough length of an idle interval can help reduce the energy. On the other hand,



(a) $U = 0.3$ (b) $U = 0.5$

Figure 4: Evaluation results of Algorithm P-Procrastination for different values of user parameter α by varying α.



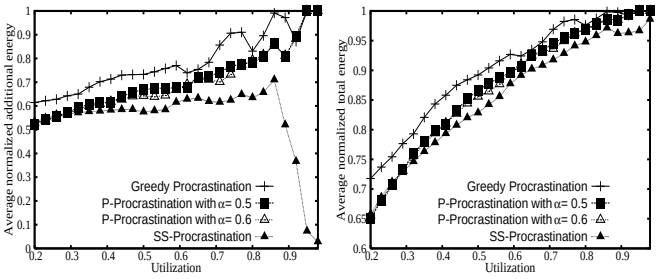
(a) $U = 0.3$ (b) $U = 0.5$

Figure 5: Evaluation results of Algorithm P-Procrastination for different values of user parameter α by varying E_{sw}.

when E_{sw} is smaller, if the next job will arrive soon, the processor should try to remain active to prevent the processor from switching between the active mode and the dormant mode, and, hence, α should be smaller. The best setting of α might be proportional to energy switching overhead E_{sw}. By evaluating more settings, including the variation of tasks, the variation of speed $s_{\min}$, and the energy of switching overhead, we find that setting the user parameter α as $\frac{1}{\gamma} \frac{P(s^*)P_{ind}}{P(s_{\min})P_d(s^*)} E_{sw}$ in Algorithm P-Procrastination could derive effective schedules on the minimization of energy consumption, where γ is the power normalization factor (γ is 0.04 here). Because of space limitation, we could not include the extensive evaluation results here.

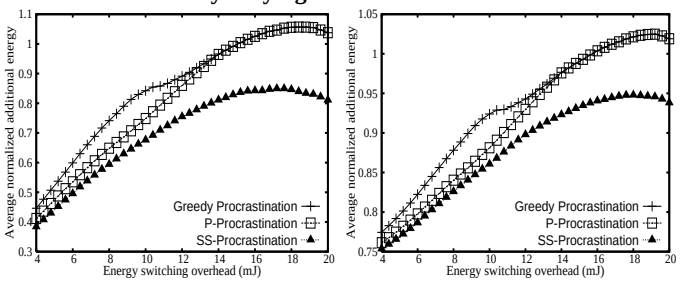
Figure 6 shows the evaluation results of Algorithm P-Procrastination for $\alpha = 0.5, 0.6$ and Algorithm SS-Procrastination with $E_{sw} = 8$mJ and 20 tasks by varying the utilization U at the critical speed. When U is small, Algorithm SS-Procrastination has little chance to execute tasks at speeds lower than the critical speed, and, hence, the performance of Algorithm SS-Procrastination is almost the same as that of Algorithm P-Procrastination. On the other hand, when U is large, Algorithm SS-Procrastination outperforms Algorithm P-Procrastination, especially when $U > 0.85$, because Algorithm SS-Procrastination has more chance to execute tasks at speeds lower than the critical speed to reduce the energy consumption. However, when U is large enough, the improvement of Algorithm SS-Procrastination on the normalized total energy is much smaller than that on the normalized additional energy. This is because the execution energy consumption to execute all the tasks at the critical speed is much greater than the additional energy consumption. The variation of the number of tasks does not significantly affect the performance of Algorithms P-Procrastination and SS-Procrastination, and, hence, the results are omitted here.

Figure 7 shows the evaluation results of Algorithm P-Procrastination for $\alpha = \min\{1, \frac{1}{\gamma} \frac{P(s^*)P_{ind}}{P(s_{\min})P_d(s^*)} E_{sw}\}$ and Algorithm SS-Procrastination by varying energy switching overhead E_{sw}, in which U is a random variable in $[0.2, 0.8]$ for 1024 runs. The reduction of energy consumption of Algorithm P-Procrastination when the system is idle and $E_{sw} <$ 14mJ can be 3% to 11% compared to the greedy procrastination with 2% to 5% reduction in total energy. The reduction of total energy consumption by Algorithm SS-Procrastination is about 5% to 15%. When E_{sw} is great enough, i.e., $E_{sw} >$ 14mJ, Algorithm P-Procrastination



(a) Additional Energy (b) Total Energy

Figure 6: Evaluation results of Algorithm P-Procrastination for $\alpha = 0.5$ and $\alpha = 0.6$ and Algorithm SS-Procrastination when E_{sw} is 8mJ and $n = 20$ by varying U.



(a) Additional Energy (b) Total Energy

Figure 7: Evaluation results of Algorithm P-Procrastination for $\alpha = \min\{1, \frac{1}{\gamma}\frac{P(s^*)P_{ind}}{P(s_{\min})P_d(s^*)}E_{sw}\}$ and Algorithm SS-Procrastination by varying energy switching overhead E_{sw}.

is almost as the same as Algorithm Greedy Procrastination. When $E_{sw} > 16$mJ, Algorithm P-Procrastination and Algorithm Greedy Procrastination might lead to a schedule with more energy consumption than the baseline schedule due to improper procrastination, and, hence, the normalized additional energy of the algorithms might be greater than 1. Algorithm SS-Procrastination has only marginal improvement when E_{sw} is small. However, when E_{sw} is greater, the improvement of Algorithm SS-Procrastination becomes significant, since the schedule has more time intervals to execute tasks at speeds lower than the critical speed instead of being idle in the active mode.

6. CONCLUSION

This paper explores energy-efficient scheduling of periodic real-time tasks in a uniprocessor leakage-aware dynamic voltage scaling system, in which the processor might be turned off. We propose two algorithms to reduce the energy consumption. Algorithm P-Procrastination uses a user-specified parameter α to determine whether the scheduler should turn the processor to the dormant mode when there is no job for execution. Algorithm SS-Procrastination applies on-line simulated scheduling to determine whether we should procrastinate or execute jobs at speeds lower than the critical speed. To our best knowledge, Algorithm SS-Procrastination is the first algorithm that might execute jobs at speeds lower than the critical speed with energy reduction when a processor can enter the dormant mode. The experimental results show that Algorithm SS-Procrastination and Algorithm P-Procrastination could greatly outperform existing greedy procrastination schemes. The reduction of energy consumption of Algorithm P-Procrastination when the system is idle can be 3% to 11% compared to the greedy procrastination proposed in [10], while the total energy is reduced by about 2% to 5%. The reduction of total energy consumption by Algorithm SS-Procrastination is about 5% to 15%.

Although we focus our discussions on dynamic-priority tasks with the earliest-deadline-first scheduling policy, our proposed algorithms can be easily extended to systems with fixed-priority tasks by using different algorithms on the calculation of procrastination lengths of jobs in [4, 8]. Moreover, the algorithms can also be extended easily to processors with multiple modes for power reduction, such as dormant mode, standby mode, etc. When tasks might complete earlier than their worst-case estimations, slack reclamation algorithms in the literature, such as [9, 17], can be applied to the proposed algorithms. For future research, we will explore energy-efficiency for tasks with resource competition.

References

[1] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *FOCS*, pages 530–539. IEEE Computer Society, 2004.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the IEEE EuroMicro Conference on Real-Time Systems*, pages 225–232, 2001.

[3] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367. ACM Press, 2006.

[4] J.-J. Chen and T.-W. Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 153–162, 2006.

[5] J.-J. Chen, T.-W. Kuo, and C.-S. Shih. $1+\epsilon$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor. In *the 2nd ACM Conference on Embedded Software (EMSOFT)*, pages 247–250, 2005.

[6] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 37–46, 2003.

[7] T. Ishihara and H. Yasuura. Voltage scheduling problems for dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 197–202, 1998.

[8] R. Jejurikar and R. K. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 57–66, 2004.

[9] R. Jejurikar and R. K. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *DAC*, pages 111–116, 2005.

[10] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, pages 275–280, 2004.

[11] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the 40th Design Automation Conference*, pages 125–130, 2003.

[12] Y.-H. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–112, 2003.

[13] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[14] C. D. Locke, D. R. Vogel, and T. J. Mesler. Building a predictable avionics platform in ada: A case study. In *IEEE Real-Time Systems Symposium*, pages 181–189, 1991.

[15] P. Mejía-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2):284–306, 2004.

[16] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 140–148, 2004.

[17] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 21–24, 2001.

[18] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2002.

[19] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design*, pages 365–368. IEEE Press, 2000.

[20] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382. IEEE, 1995.

[21] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *IEEE Real-time and Embedded Technology and Applications Symposium*, pages 397–407, 2006.