A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis

Woonseok Kim* Jihong Kim† Sang Lyul Min*

School of Computer Science and Engineering
Seoul National University ENG4190, Seoul, Korea, 151-742
wskim@archi.snu.ac.kr, jihong@davinci.snu.ac.kr, symin@dandelion.snu.ac.kr

Abstract

Dynamic voltage scaling (DVS), which adjusts the clock speed and supply voltage dynamically, is an effective technique in reducing the energy consumption of embedded real-time systems. The energy efficiency of a DVS algorithm largely depends on the performance of the slack estimation method used in it. In this paper, we propose a novel DVS algorithm for periodic hard real-time tasks based on an improved slack estimation algorithm. Unlike the existing techniques, the proposed method takes full advantage of the periodic characteristics of the real-time tasks under priority-driven scheduling such as EDF. Experimental results show that the proposed algorithm reduces the energy consumption by 20∼40% over the existing DVS algorithm. The experiment results also show that our algorithm based on the improved slack estimation method gives comparable energy savings to the DVS algorithm based on the theoretically optimal (but impractical) slack estimation method.

1. Introduction

Since the energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage V_{DD}, lowering the supply voltage V_{DD} is an effective way of reducing energy consumption of embedded mobile systems such as digital cellular phones and personal digital assistants. However, lowering the supply voltage also decreases the maximum achievable clock speed; in the CMOS circuit, the delay T_D is given by $T_D \propto V_{DD}/(V_{DD} - V_T)^{\alpha}$, where V_T is the threshold voltage and α is a velocity saturation index [11]. This energy-delay trade-off makes possible various dynamic voltage scaling (DVS) techniques that adjust the clock speed and the supply voltage dynamically according to the performance requirements of given tasks.

For hard real-time systems where tasks have stringent timing constraints, the energy-delay trade-off makes the DVS problem more challenging. When the supply voltage and clock speed are lowered for reduced energy consumption, the execution times of the tasks may increase, resulting in deadline misses. Since deadline misses in real-time systems can cause catastrophic system failures, dynamic voltage scaling can utilize only slack times (or idle times) when adjusting voltage levels. Therefore, the energy efficiency of a real-time DVS algorithm largely depends on how accurately these slack times are estimated.

Slack time analysis has been extensively investigated in real-time server systems in which aperiodic (or sporadic) tasks are jointly scheduled with periodic tasks [1, 15, 10]. In these systems, the purpose of slack time analysis is to improve the response time of aperiodic tasks or to increase their acceptance ratio. However, since the existing slack analysis methods [1, 15, 10] usually require high time and/or space overheads, they are not applicable to mobile embedded systems where resources are constrained. For this reason, most existing on-line DVS algorithms for embedded systems use simple heuristics in estimating slack times.

While there have been various research efforts on *off-line* DVS scheduling algorithms that try to identify the slack times under worst-case execution scenarios [16, 3, 7, 14, 5, 2, 9], little has been done on efficiently estimating on-line slack times due to dynamic workload variations, often called workload-variation slack times (VSTs) [6]. In existing algorithms such as [13, 14] for periodic real-time systems, the VSTs are estimated too conservatively. For example, in the **`lpps/EDF`** algorithm [14], the VSTs are identified only when a single task is activated and the arrival time of the next task is later than the task's worst case execution time (WCET). This is overly pessimistic in estimating VSTs. In many task sets, more VSTs can be computed without significantly increasing overheads.

In this paper, we focus on improving the on-line slack estimation part of a DVS algorithm and show that a better slack estimation method can significantly improve the energy effi-

*This work was supported in part by the Ministry of Education under the BK21 program, and by the Ministry of Science and Technology under the National Research Laboratory program.

†This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation.

ciency. In particular, we propose a novel on-line DVS algorithm for periodic real-time tasks that are scheduled under the *Earliest-Deadline-First* (EDF) algorithm. Unlike the existing algorithms, the proposed algorithm fully takes advantage of the task periodicity in estimating available slack times. The novel aspect of the proposed algorithm is that it can reclaim future slack times from lower-priority tasks as well as those from already completed higher-priority tasks. Experimental results show that the proposed algorithm can reduce the energy consumption by 20∼40% over the existing DVS algorithm.

The rest of the paper is organized as follows. Before we describe the proposed algorithm in detail, we explain the target system model and present a motivational example in Section 2. In Section 3, we give the basic idea of the proposed slack estimation method. The details of the proposed voltage scheduling algorithm including the improved slack estimation algorithm are given in Section 4. Section 5 discusses experimental results and, finally, Section 6 concludes with a summary and discussion of future work.

2. Motivation

2.1. System model

We consider a preemptive hard real-time system in which periodic real-time tasks are scheduled under the EDF scheduling policy. The target variable voltage processor can scale its supply voltage and clock speed continuously within its operational ranges, $[v_{min}, v_{max}]$ and $[f_{min}, f_{max}]$. A set $\mathcal{T}$ of n periodic tasks is denoted as $\mathcal{T} = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}$ where tasks are assumed to be mutually independent. Each task τ_i has its own period p_i and worst case execution time (WCET) w_i. The relative deadline d_i of τ_i is assumed to be equal to its period p_i. Each task activates (or releases) its instance periodically, and the (j+1)-th instance of τ_i is denoted by $\tau_{i,j}$. We also denote a task instance by a single subscript such as $\tau_{\boldsymbol{\alpha}}$ if its meaning is clear by the context. Each task instance $\tau_{\boldsymbol{\alpha}}$ has its own arrival time $a_{\boldsymbol{\alpha}}$ and absolute deadline $d_{\boldsymbol{\alpha}}$.

In the remainder of this paper, we assume that the processor utilization $U_{\mathcal{T}}$ of $\mathcal{T}$ is 1 when $\mathcal{T}$ is executed under the worst-case execution scenario. In the case where $U_{\mathcal{T}}$ is not 1, we can lower the maximum clock frequency to $f'_{max} = U_{\mathcal{T}} \cdot f_{max}$ without violating any timing constraint. With f'_{max}, the processor utilization becomes 1, leaving no idle intervals if every task instance takes the WCET for its execution.

2.2. Motivational example

Consider a periodic task set $\mathcal{T}$ shown in Table 1. In addition to periods and WCETs, we assume that the average-case execution time (ACET) of each task is 0.5 in Table 1[1]. In order to illustrate that a better on-line slack analysis is possible for more energy savings, suppose that $\mathcal{T}$ is scheduled

[1]We assume that tasks' execution times are based on the maximum clock frequency.

Table 1. An example real-time task set $\mathcal{T}$.

	period (p_i)	WCET (w_i)	ACET (a_i)
τ_0	2	1	0.5
τ_1	3	1	0.5
τ_2	6	1	0.5

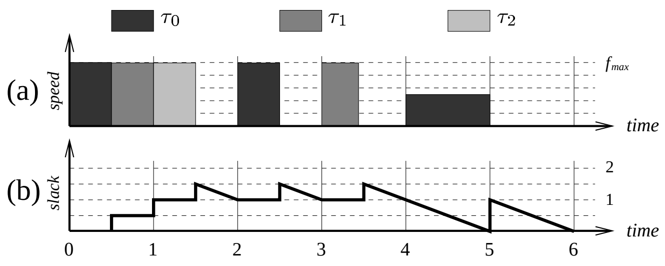
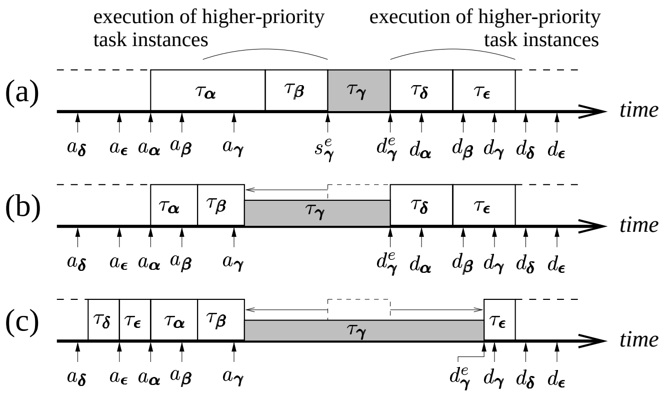


Figure 1. Voltage scheduling examples; (a) the voltage schedule from the `lpps/EDF` algorithm, and (b) the cumulative slack times at each scheduling point under the `lpps/EDF` algorithm.

under **`lpps/EDF`** [14]. Under the **`lpps/EDF`** scheduling algorithm, the workload-variation slack times exist if 1) there is currently no other ready task instance and 2) the earliest arrival time of the next task instance is later than the worst-case completion time of the current task instance. Otherwise, the **`lpps/EDF`** algorithm assumes that there exists no slack time and schedules the current task instance with f_{max}.

The above simple heuristics leads to an overly conservative approach in estimating slack times. Figure 1(a) shows the speed schedule when the **`lpps/EDF`** algorithm is used for the example task set $\mathcal{T}$ assuming that the actual execution time of each task is equal to its ACET. When the first scheduled task instance $\tau_{0,0}$ completes its execution at $t = 0.5$, there is a slack time of 0.5 time units, which can be used to lower the execution speed for the next scheduled task instance $\tau_{1,0}$. However, since $\tau_{2,0}$ was already activated, $\tau_{1,0}$ is scheduled with the maximum speed. When $\tau_{1,0}$ is completed and $\tau_{2,0}$ is scheduled at $t = 1.0$, there exists no other activated task. But, since the arrival time of $\tau_{0,1}$ is not later than the completion time of $\tau_{2,0}$ when $\tau_{2,0}$ takes its WCET, $\tau_{2,0}$ also cannot be scheduled with a lowered voltage even though there exists a slack time of 1. In fact, only one task instance ($\tau_{0,2}$) within a hyperperiod of 6 can be scheduled with a lowered speed.

The inability of using lower speeds for the example task set $\mathcal{T}$ with the **`lpps/EDF`** comes from an inefficient slack time estimation method used in the algorithm. As shown in Figure 1(b), there *do exist* slack times at most scheduling points. Our main goal in this paper is to devise an efficient and accurate slack time estimation method that can identify most of



execution of higher-priority task instances
execution of higher-priority task instances
(a)
(b)
(c)
time

Figure 2. Sources of slack times for τ_γ; (a) no slack times available, (b) slack times from higher-priority tasks, and (c) slack times both from higher/lower-priority tasks.

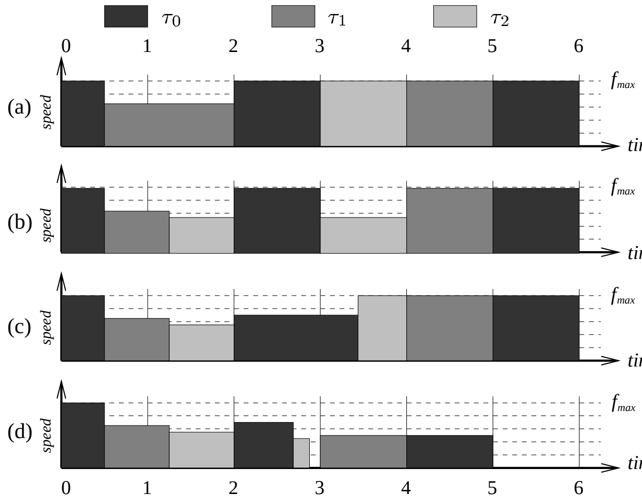


Figure 3. Voltage scheduling examples.

available slack times without incurring too much overhead.

3. Basic idea

Consider a set $\mathcal{T}$ of periodic tasks whose processor utilization $U_{\mathcal{T}}$ is 1. Since $U_{\mathcal{T}}$ is equal to 1, there is no slack time available under the worst case execution scenario in which every task instance takes the WCET for its execution. In this case, the time interval between the arrival time a_γ and the deadline d_γ of a task τ_γ consists of three sub-intervals: (1) the sub-interval $[a_\gamma, s^e_\gamma]$, (2) the sub-interval $[s^e_\gamma, d^e_\gamma]$, and (3) the sub-interval $[d^e_\gamma, d_\gamma]$ where s^e_γ and d^e_γ are the expected start time and finish time for τ_γ under the worst case scenario. For example, in Figure 2, in $[a_\gamma, s^e_\gamma]$, tasks whose priority is higher than τ_γ (such as those denoted by τ_α and τ_β in Figure 2) are executed and in $[d^e_\gamma, d_\gamma]$, tasks whose priority is lower than τ_γ (such as those denoted by τ_δ and τ_ϵ in Figure 2) are executed.

When τ_α and τ_β are completed earlier than the expected start time s^e_γ of τ_γ, τ_γ can use their unused execution times, effectively moving its start time earlier than s^e_γ, as shown in Figure 2(b). It is also possible that τ_γ can delay the expected finish time d^e_γ of τ_γ (under the worst case scenario). For example, if the release time of τ_δ and τ_ϵ are earlier than a_γ, they may have (partially) completed its execution before a_γ. In this case, τ_γ may extend its deadline as shown in Figure 2(c). We call this new extended deadline the *effective deadline* of τ_γ. The effective deadline of a task instance τ_γ is defined to be the latest time that guarantees the feasible execution of all the remaining task instances.

In summary, the available slack times for a task come from two sources: 1) slack times from higher-priority tasks and 2) slack times from lower-priority tasks. In this paper, we propose a slack estimation method that accounts for both types of slack times efficiently.

In order to demonstrate that a better slack estimation can improve the energy efficiency significantly, consider the example task set of Figure 1 again. When $\tau_{0,0}$ is completed at $t = 0.5$, $\tau_{1,0}$ starts its execution at $t = 0.5$. Since $\tau_{1,0}$ was scheduled to begin its execution at $t = 1.0$, even if the execution $\tau_{1,0}$ is prolonged by 0.5 time units, the remaining schedule is still feasible. Thus, as shown in Figure 3(a), $\tau_{1,0}$ may stretch its execution with the lowered clock speed ($\frac{1}{1+0.5} f_{max}$). When $\tau_{1,0}$ completes its execution at $t = 1.25$, $\tau_{2,0}$ can stretch its execution with the clock speed of $\frac{1}{1+0.75} f_{max}$, as shown in Figure 3(b).

When $\tau_{0,1}$ is activated at $t = 2$, $\tau_{2,0}$ is still running and $\tau_{0,1}$ will preempt $\tau_{2,0}$'s execution because $\tau_{0,1}$ has a higher-priority than $\tau_{2,0}$. At this point, since $\tau_{2,0}$ has been already partially executed, the remaining execution of $\tau_{2,0}$ will not take its WCET. Therefore, $\tau_{0,1}$ may extend its deadline accordingly. In this case, only 0.57 time units is left for $\tau_{2,0}$ assuming it requires its WCET, thus $\tau_{0,1}$ may stretch its execution up to $t = 3.43$ with the clock speed of $\frac{1}{1+0.43} f_{max}$ as shown in Figure 3(c).

The remaining task instances, $\tau_{2,0}$, $\tau_{1,1}$, and $\tau_{0,2}$, can be scheduled in a similar manner. The final schedule is shown in Figure 3(d). Assuming that the power consumption is proportional to the square of clock speed, the schedule in Figure 3(d) consumes 28.6% less energy than the schedule determined by the **lpps/EDF** algorithm.

Slack times from higher-priority tasks that completed earlier than expected can easily be estimated by identifying how much their unused times are left at the scheduling point. However, estimating slack times from lower-priority tasks requires non-trivial computational overhead. In order to identify the latter type of slack times exactly, it must be checked that how much time can be delayed for each of the remaining task instances within the hyperperiod. The overhead of such an exact slack estimation rapidly increases with the number of task instances in the hyperperiod of a given task set. In the next section, we describe in detail an approximate but still accurate slack estimation algorithm for the second type of slack times.

4. Low-power scheduling using slack estimation heuristic

4.1. Slack estimation algorithm

Before describing the proposed algorithm for slack estimation, we define the following two notations that are used in keeping track of available slack times of each task instance.

- U_i^{rem} : the unused execution time by τ_i
- W_i^{rem} : the remaining WCET of τ_i

We assume that a real-time scheduler has two queues: *waitQueue* and *readyQueue*. The *waitQueue* and the *readyQueue* contain the completed tasks and the currently activated tasks, respectively. All the tasks are initially queued in *waitQueue*, in which the tasks are sorted by their next arrival time. When a task is activated, the task is moved from *waitQueue* to *readyQueue*. At each task activation, both U_i^{rem} and W_i^{rem} are set to w_i, i.e., $U_i^{rem} = W_i^{rem} = w_i$. Among the tasks in *readyQueue*, the *active task* $\tau_{\boldsymbol{\alpha}}$ with the earliest deadline is scheduled to run under the EDF scheduling policy.

As $\tau_{\boldsymbol{\alpha}}$ executes, its $W_{\boldsymbol{\alpha}}^{rem}$ decreases and consumes its available execution time. $\tau_{\boldsymbol{\alpha}}$ may complete its execution or be preempted by a higher-priority task instance. When $\tau_{\boldsymbol{\alpha}}$ is preempted by a newly activated higher-priority task instance, $\tau_{\boldsymbol{\alpha}}$ is re-queued into *readyQueue* while waiting for the resumption. When $\tau_{\boldsymbol{\alpha}}$ completes its execution, its remaining WCET $W_{\boldsymbol{\alpha}}^{rem}$ is reset to 0, and $\tau_{\boldsymbol{\alpha}}$ is inserted into *waitQueue*. Note that, however, we do not reset the unused time $U_{\boldsymbol{\alpha}}^{rem}$ of $\tau_{\boldsymbol{\alpha}}$; $U_{\boldsymbol{\alpha}}^{rem}$ is used to estimate the slack time available for other task instances.

At the current scheduling point t_{cur}, the available execution time for $\tau_{\boldsymbol{\alpha}}$ consists of the following three times:

- $S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$: the sum of the unused times from higher-priority task instances already completed before t_{cur},
- $U_{\boldsymbol{\alpha}}^{rem}$: the currently remaining time for $\tau_{\boldsymbol{\alpha}}$, and
- $S_L(\tau_{\boldsymbol{\alpha}}, t_{cur})$: the sum of the slack times from the lower-priority task instances.

Let $\mathcal{T}_{\mathcal{H}}(\tau_{\boldsymbol{\alpha}}, t_{cur})$ be the set of higher-priority task instances already completed before t_{cur}. Then $S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$ is computed as follows:

$$S_H(\tau_{\boldsymbol{\alpha}}, t_{cur}) = \sum_{\tau_i \in \mathcal{T}_{\mathcal{H}}(\tau_{\boldsymbol{\alpha}}, t_{cur})} U_i^{rem}. \qquad (1)$$

Note that $U_{\boldsymbol{\alpha}}^{rem}$ may differ from $W_{\boldsymbol{\alpha}}^{rem}$. For example, if $\tau_{\boldsymbol{\alpha}}$ is resumed from the previous preemption, part of work for $\tau_{\boldsymbol{\alpha}}$ might have been processed in the previous execution while consuming the unused times of higher-priority task instances. In this case, $U_{\boldsymbol{\alpha}}^{rem}$ may be greater than $W_{\boldsymbol{\alpha}}^{rem}$.

While $S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$ and $U_{\boldsymbol{\alpha}}^{rem}$ can be estimated easily, the exact estimation of $S_L(\tau_{\boldsymbol{\alpha}}, t_{cur})$ requires non-trivial time and/or space overhead. For example, if we were to use the optimal slack-stealing algorithm in [10], we should maintain the information on the amount of work that must be completed before the deadline of each task instance in the hyperperiod. With this information, the available slack time at a scheduling point can be exactly estimated. This optimal approach has $O(N)$ of time and space complexity in the worst case, where N is the number of task instances in the hyperperiod.

In this paper, we estimate $S_L(\tau_{\boldsymbol{\alpha}}, t_{cur})$ approximately as follows. Since $U_{\boldsymbol{\alpha}}^{rem}$ and $S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$ are the time resources which can be used for $\tau_{\boldsymbol{\alpha}}$'s execution, $\tau_{\boldsymbol{\alpha}}$ may stretch its execution with a lowered clock speed by exploiting these times. That is, $\tau_{\boldsymbol{\alpha}}$ can be scheduled with the clock speed of $\frac{W_{\boldsymbol{\alpha}}^{rem}}{U_{\boldsymbol{\alpha}}^{rem} + S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})} f_{max}$. Suppose that $\tau_{\boldsymbol{\alpha}}$ is scheduled with this clock speed at t_{cur}. If $\tau_{\boldsymbol{\alpha}}$ requires the WCET for its execution, $\tau_{\boldsymbol{\alpha}}$ will be completed at $t_a = t_{cur} + S_H(\tau_{\boldsymbol{\alpha}}, t_{cur}) + U_{\boldsymbol{\alpha}}^{rem}$. In the case where a higher-priority task is activated before t_a, $\tau_{\boldsymbol{\alpha}}$ will be preempted and cannot complete its execution before t_a. If such a task instance exists, it is computationally expensive to estimate when $\tau_{\boldsymbol{\alpha}}$ resumes and how much slack times will be available when $\tau_{\boldsymbol{\alpha}}$ is resumed. Thus, in this case we just assume that the available execution time for $\tau_{\boldsymbol{\alpha}}$ is ($U_{\boldsymbol{\alpha}}^{rem} + S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$). If such a higher-priority task does not exist, then we check whether more slack times are available for $\tau_{\boldsymbol{\alpha}}$ from lower-priority task instances. At t_a, if there is no activated task, it is easy to see that $\tau_{\boldsymbol{\alpha}}$ could have extended its execution to the earliest arrival time (t'_a) of a task in *waitQueue* as in [14]. Moreover, at t_a (or t'_a), if the task instances in *readyQueue* have slack times, $\tau_{\boldsymbol{\alpha}}$ could have further extended its execution using these slack times. Let $\tau_{\boldsymbol{\beta}}$ be the highest-priority task instance among the ones which are activated but not completed before t_a. If $\mathcal{T}_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur}) \neq \emptyset$,where $\mathcal{T}_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur})$ is the set of task instance that have lower-priority than $\tau_{\boldsymbol{\alpha}}$ and have already completed before t_{cur}, $\tau_{\boldsymbol{\beta}}$ could have slack times, and they can be estimated as follows:

$$\mathcal{T}'_{\mathcal{L}}(\tau_{\boldsymbol{\beta}}, t_a) = \{\tau_i \mid \tau_i \in \mathcal{T}_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur}) \text{ and } d_i \leq d_{\boldsymbol{\beta}}\},$$

$$S_L(\tau_{\boldsymbol{\alpha}}, t_{cur}) = (U_{\boldsymbol{\beta}}^{rem} - W_{\boldsymbol{\beta}}^{rem}) + \sum_{\tau_i \in \mathcal{T}'_{\mathcal{L}}(\tau_{\boldsymbol{\beta}}, t_a)} U_i^{rem} \qquad (2)$$

The above estimation of $S_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur})$ is valid only when there is no task instance that can be activated during [t_a, $t_a + S_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur})$] (or [$t'_a$, $t'_a + S_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur})$]) and has an earlier deadline than $\tau_{\boldsymbol{\beta}}$ (as shown in Figure 4(a)). If such a task instance $\tau_{\boldsymbol{\gamma}}$ exists (as shown in Figure 4(b)), the execution time extension of $\tau_{\boldsymbol{\alpha}}$ is restricted to the arrival time of $\tau_{\boldsymbol{\gamma}}$, i.e., $S_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur}) = a_{\boldsymbol{\gamma}} - t_a$.

Figure 5 summarizes the slack estimation procedure described above. In this algorithm, since at most n tasks can be activated at any time, Equations 1 and 2 (lines 5 and 15 in Figure 5) can be computed in $O(n)$. In computing Equation 2,

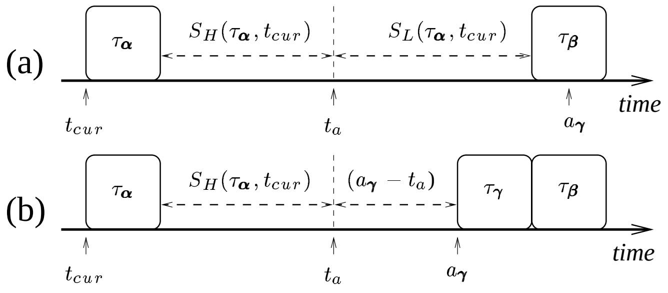

(a)
$\tau_{\boldsymbol{\alpha}}$
$S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$
$S_L(\tau_{\boldsymbol{\alpha}}, t_{cur})$
$\tau_{\boldsymbol{\beta}}$
time
t_{cur}
t_a
$a_{\boldsymbol{\gamma}}$
(b)
$\tau_{\boldsymbol{\alpha}}$
$S_H(\tau_{\boldsymbol{\alpha}}, t_{cur})$
$(a_{\boldsymbol{\gamma}} - t_a)$
τ_{γ}
$\tau_{\boldsymbol{\beta}}$
time
t_{cur}
t_a
$a_{\boldsymbol{\gamma}}$

Figure 4. Slack estimation examples; when the arrival time of higher-priority task is (a) later than t_a + $S_L(\tau_{\boldsymbol{\alpha}}, t_{cur})$, and (b) later than t_a but earlier than t_a + $S_L(\tau_{\boldsymbol{\alpha}}, t_{cur})$.

```
Algorithm 1  Estimate the available execution time for $\tau_{\boldsymbol{\alpha}}$

1.  Input: the active task $\tau_{\boldsymbol{\alpha}}$, waitQ, readyQ, current time $t_{cur}$;
2.  Output: the available execution time $S(\tau_{\boldsymbol{\alpha}})$ for $\tau_{\boldsymbol{\alpha}}$;
3.  $\mathcal{T}_{\mathcal{H}}(\tau_{\boldsymbol{\alpha}}, t_{cur}) =$ the set of already completed higher-priority task instances;
4.  $\mathcal{T}_{\mathcal{L}}(\tau_{\boldsymbol{\alpha}}, t_{cur}) =$ the set of already completed lower-priority task instances;
5.  $S_H = \sum_{\tau_i \in \mathcal{T}_{\mathcal{H}}(\tau_{\boldsymbol{\alpha}}, t_{cur})} U_i^{rem} + U_{\boldsymbol{\alpha}}^{rem}$;
6.  $S_L = 0$;
7.  $t_{\boldsymbol{\alpha}}^h =$ the earliest arrival time of a task instance whose priority is higher than $\tau_{\boldsymbol{\alpha}}$;
8.  if $t_{cur}+S_H < t_{\boldsymbol{\alpha}}^h$ then
9.      $t_a$ = the worst case completion time of $\tau_{\boldsymbol{\alpha}}$ with the clock speed of $\frac{W_{\boldsymbol{\alpha}}^{rem}}{S_H}$
10.     if there will be no activated task instance (i.e., readyQ = $\emptyset$ ) at $t_a$ then
11.         let $t'_a$ = min($a_i | \tau_i \in$ waitQ));
12.     $t_a$ = max($t_a$,$t'_a$);
13.     $\tau_{\boldsymbol{\beta}}$ = the task instance expected to be scheduled at $t_a$;
14.     $\mathcal{T}'_{\mathcal{L}}(\tau_{\boldsymbol{\beta}}, t_{cur})$ = the set of completed task instances whose priorities are
                higher than or equal to that of $\tau_{\boldsymbol{\beta}}$ but less than that of $\tau_{\boldsymbol{\alpha}}$;
15.     $S_L = (U_{\boldsymbol{\beta}}^{rem} - W_{\boldsymbol{\beta}}^{rem}) + \sum_{\tau_i \in \mathcal{T}'_{\mathcal{L}}(\tau_{\boldsymbol{\beta}}, t_{cur})} U_i^{rem}$;
16.     $t_{\boldsymbol{\beta}}^h =$ the earliest arrival time of task instance whose priority is
            higher than $\tau_{\boldsymbol{\beta}}$;
17.     $S_L =$ min $(S_L, t_{\boldsymbol{\beta}}^h - t_a)$;
18. end if
19. $S(\tau_{\boldsymbol{\alpha}}) =$ min $(S_H+U_{\boldsymbol{\alpha}}^{rem}+S_L\ ,\ d_{\boldsymbol{\alpha}} - t_{cur})$;
20. Output $S(\tau_{\boldsymbol{\alpha}})$;
```

Figure 5. Slack estimation algorithm.

we need to know which task will have the highest priority at the given time. This step can be performed by simply scanning *readyQueue* and *waitQueue*, which can be done in $O(n)$ time. Hence, the proposed algorithm has the time complexity of $O(n)$ where $n << N$.

4.2. Voltage scheduling algorithm

The DVS algorithm based on the proposed slack estimation method is summarized in Figure 6. The algorithm is executed at every scheduling point such as the activation, resumption, and completion of task instances.

At each scheduling point, we first update the unused times of task instances in order to reflect the consumed times during the execution of previously scheduled task instance or an idle interval. To keep the consistency in reclaiming unused times of task instances, the unused times of task instances are consumed in decreasing order of the task instances' priority. That is, when $\tau_{\boldsymbol{\alpha}}$ is completed, the unused times of already completed higher-priority task instances are consumed first, and then $\tau_{\boldsymbol{\alpha}}$'s unused time is consumed. Finally, the unused times of (partially) completed lower-priority task instances are consumed. If the system has been idle, the unused times of completed task instances in *waitQueue* are also consumed in decreasing order of task instances' priority. Since this step (line 3 in Figure 6) is required only for task instances in *waitQueue* and the active task instances in *readyQueue*, whose sum is n at most, it can be done in $O(n)$.

At the scheduling point t_{cur}, if the previous task instance τ_p completes its execution, its W_p^{rem} is reset to 0 and τ_p is requeued into *waitQueue*. If τ_p is not completed but preempted by a new active task τ_{cur}, τ_p remains in *readyQueue* updating its W_p^{rem} only (line 8 in Figure 6).

When there is no active task, the processor enters a power-down mode until a new task instance is activated. Otherwise, we compute the available execution time $S(\tau_{\boldsymbol{\alpha}})$ for $\tau_{\boldsymbol{\alpha}}$ using the algorithm in Figure 5, and then adjust the clock speed to

$$f_{clk}(t_{cur}) = \frac{W_{\boldsymbol{\alpha}}^{rem}}{S(\tau_{\boldsymbol{\alpha}})} f_{max} \qquad (3)$$

The supply voltage is also adjusted accordingly.

Managing the slack information (in line 3 in Figure 6) requires $O(n)$ operations. Thus, the entire steps of the proposed DVS algorithm can be performed with the worst-case time complexity of $O(n)$. Furthermore, the proposed DVS algorithm does not require any static information which should be prepared in the off-line phase; U_i^{rem} is the only additional online information required for each task τ_i. Thus, the space overhead of the proposed algorithm is also marginal.

5. Experimental results

To evaluate the energy efficiency of the proposed voltage scheduling algorithm, we performed several experiments with three DVS algorithms using an energy simulator: 1) the **lpps/EDF** algorithm [14], 2) the DVS algorithm based on the proposed slack-estimation method, and 3) the DVS algorithm with the optimal slack-estimation method [10]. We denote three algorithms by **lpps/EDF**, **lp/SEH**, and **lp/OPT**, respectively. The energy simulator is based on the ARM8 microprocessor core. The clock speed is scaled in the range of [8, 100] MHz with a step size of 1 MHz and the supply voltage

Algorithm 2 Schedule $\tau_{\boldsymbol{\alpha}}$ with an appropriate clock speed and corresponding voltage

```
1.  Input: $\tau_{\boldsymbol{\alpha}}$ = the currently scheduled task instance,
           $\tau_p$ = the previously scheduled task instance,
           $t_{prv}$ = the previous scheduling point,
           $t_{cur}$ = the current scheduling point, and
           $f_{clk}$ = the current clock speed;
2.  Output: voltage and clock speed setting;
3.  Update unused times of tasks
        by the amount of the consumed time $I = |t_{cur} - t_{prv}|$;
        : i.e., for each task $\tau_i$ in decreasing order of its priority,
        decrease $U_i^{rem}$ until the entire consumed time of $I$ is reflected
4.  if $\tau_p$ is completed at $t_{cur}$ then
5.      Reset $W_p$ to 0 and move $\tau_p$ from readyQ to waitQ;
6.  end if
7.  if $\tau_p$ is preempted by $\tau_{\boldsymbol{\alpha}}$ then
8.      Decrease $W_p$ by the execution time of $\tau_p$;
        (i.e., $W_p = W_p - I\frac{f_{clk}}{f_{max}}$)
9.  end if
10. if there is no activated task instance (i.e., readyQ = $\emptyset$) then
11.     Powerdown the processor until the next task instance arrives;
12. else
13.     Estimate the available times $S(\tau_{\boldsymbol{\alpha}})$ for $\tau_{\boldsymbol{\alpha}}$ using Algorithm 1;
14.     Set the clock speed by $f_{clk} = (W_{\boldsymbol{\alpha}}/S(T_{\boldsymbol{\alpha}})) * f_{max}$ &
15.         adjust the corresponding voltage;
16. end if
```

Figure 6. Low-power scheduling algorithm based on the slack estimation algorithm.

is scaled in the range of [1.1, 3.3] V. We assume that the system enters into a power-down mode when the system is idle. (The power consumption of a power-down mode is assumed to be 0.) In the experiments, we assume that the voltage scaling overhead is negligible both in the time delay and power consumption.

Figure 7 shows the experimental results for three real-world application task sets and synthesized task sets. The three real-world application task sets are the task sets from the Computerized Numerical Control (CNC) machine controller application [4], Avionics application [8], and Video phone application [12]. The characteristics of these applications are summarized in Table 2. In each experiment, the execution time of each task instance was randomly drawn from a Gaussian distribution[2] in the range of [BCET, WCET] where BCET is the best case execution time. For the three application task sets, whose results are reported in Figure 7(a)∼(c), we performed the experiments by varying the BCET from 10% to 100% of WCET for each application. In each figure, the x-axis represents the ratio of BCET to WCET while the y-axis represents the normalized energy consumption ratio to the energy consumption of the same application running on a DVS-

[2]With the mean $m = \frac{BCET+WCET}{2}$ and the standard deviation $\sigma = \frac{WCET-BCET}{6}$.

Table 2. Task sets for experiments.

Applications	♯ tasks	WCETs (ms)	Periods (ms)	Utilization
CNC	8	0.035 ∼ 0.72	2.4∼9.6	0.489
Avionics	17	1 ∼ 9	25∼1,000	0.848
Videophone	4	1.4 ∼ 50.4	40∼66.7	0.986

unaware system with a power-down mode only.

Since the average execution times of task instances decrease as the BCET gets smaller, the slack times of task instances increase as the BCET decreases. Thus, as shown in Figure 7, the energy efficiency of each DVS algorithm increases as the ratio of BCET to WCET decreases. However, the energy efficiency of the proposed **lp/SEH** algorithm increases much faster than **lpps/EDF** because the proposed **lp/SEH** algorithm is more efficient in exploiting the slack times of tasks than the **lpps/EDF** algorithm. The experimental results show that the **lp/SEH** achieves 20∼40 % more energy savings compared to the **lpps/EDF**. Figure 7 also shows that, in most cases, the proposed **lp/SEH** algorithm is comparable to the **lp/OPT** in its energy efficiency, although the **lp/OPT** algorithm is computationally much more expensive than the **lp/SEH** algorithm.

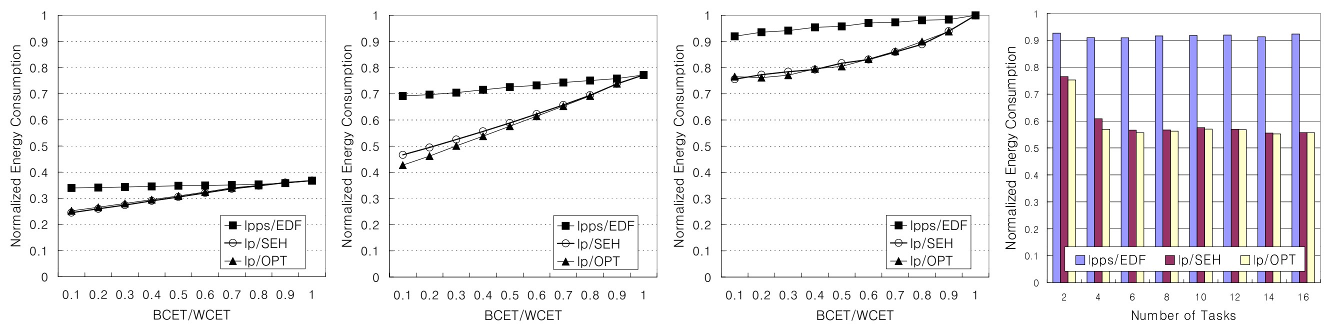
We also performed extensive experiments using synthesized application sets by varying the number of tasks in a task set whose results are given in Figure 7(d). For each number of tasks, we randomly generated 100 task sets[3] whose utilization is 1. The results show that as the number of tasks increases, the energy efficiency of **lp/SEH** and **lp/OPT** increases while that of **lpps/EDF** is not changed. This can be explained by the fact that with an increased number of tasks, **lp/SEH** and **lp/OPT** have more task instances from which the two algorithms take slack times while in **lpps/EDF** the slack time estimation is limited to the time between the completion of a task instance and the arrival of the next task instance, which is largely independent of the number of tasks in the system.

Results in Figure 7 also show that, while **lp/OPT** estimates all the slack times available at the scheduling point, its energy efficiency is not much better than that of **lp/SEH**. This is because **lp/OPT** is optimal only in the slack estimation step. In **lp/OPT**, the currently scheduled task consumes all the available slack times. However, this greedy slack consumption by the current task may result in less balanced voltage schedule. For example, it could produce a better voltage schedule if some of the available slack time were left for the following task. The results shown in Figure 7 strongly suggest that for optimal DVS scheduling, an intelligent slack distribution as well as efficient slack estimation is important.

6. Conclusion

We have presented a novel voltage scheduling algorithm based on an efficient slack estimation heuristic. The proposed

[3]The period and WCET of each task were randomly generated using the uniform distribution within the ranges of [10, 100] ms and [1, period) ms.



(a) CNC (b) Avionics (c) Video Phone (d) Synthesized Applications

Figure 7. Experimental results: (a) $\sim$ (c) real-world applications, and (d) synthesized applications.

algorithm is motivated by the observation that the current DVS algorithms estimate the slack times too conservatively. The main contribution of the proposed algorithm is that the slack times can be estimated more efficiently with a small additional overhead, achieving much higher energy efficiency over the existing DVS algorithm. Experimental results show that our algorithm reduces the energy consumption up to 40% over the existing algorithm.

The proposed **lp/SEH** algorithm described in this paper can be extended in several directions. For example, as suggested in the previous section, a more intelligent slack distribution method can be used to further improve the energy efficiency of the **lp/SEH** algorithm. We are currently investigating a profile-based slack distribution method for the **lp/SEH** algorithm. We also plan to apply the proposed slack estimated method to other scheduling policies such as fixed-priority scheduling policies.

References

[1] H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[2] F. Gruian. Hard Real-Time Scheduling Using Stochastic Data and DVS Processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 46–51, August 2001.

[3] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 178–187, December 1998.

[4] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual Assesment of a Real-Time System Design: a Case Study on a CNC controller. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 300–310, December 1996.

[5] C. M. Krishna and Y.-H. Lee. Voltage-Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems. In *Proceedings of the Sixth IEEE Real-Time Technology and Applications Symposium*, pages 156–165, June 2000.

[6] S. Lee and T. Sakurai. Run-time Voltage Hopping for Low-power Real-Time Systems. In *Proceedings of the 37th Design Automation Conference*, pages 806–809, June 2000.

[7] Y.-H. Lee and C. M. Krishna. Voltage-Clock Scaling for Low Energy Consumption in Real-time Embedded Systems. In *Proceedings of the Real-Time Computing Systems and Applications*, pages 272–279, December 1999.

[8] C. Locke, D. Vogel, and T. Mesler. Building a Predictable Avionics Platform in Ada: a Case Study. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 181–189, December 1991.

[9] G. Quan and X. S. Hu. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proceedings of the Design Automation Conference*, pages 828–833, June 2001.

[10] I. Ripoll, A. Crespo, and A. G. Fornes. An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Dynamic-Priority Preemptive Systems. *IEEE Transactions on Software Engineering*, 23(6):388–400, 1997.

[11] T. Sakurai and A. Newton. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Dealy and Other Formulars. *IEEE Journal of Solid State Circuits*, 25(2):584–594, 1990.

[12] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design and Test of Computers*, 18(2):20–30, March 2001.

[13] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the Design Automation Conference*, pages 134–139, June 1999.

[14] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proceedings of the International Conference on Computer-Aided Design*, pages 365–368, November 2000.

[15] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 10(2):179–210, March 1996.

[16] F. Yao, A. Demers, and A. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proceedings of the IEEE Foundations of Computer Science*, pages 374–382, 1995.