Preemption-Aware Dynamic Voltage Scaling in Hard Real-Time Systems

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) is a well-known low-power design technique for embedded real-time systems. Because of its effectiveness on energy reduction, several variable voltage processors have been developed and many DVS algorithms targeting these processors have been proposed. However, most existing DVS algorithms focus on reducing the energy consumption of CPU only, ignoring their negative impacts on task scheduling and system wide energy consumption. In this paper, we address one of such side effects, an increase in task preemptions due to DVS. We present two preemption control techniques which can reduce the number of task preemptions of DVS algorithms. Experimental results show that the delayed-preemption technique is effective in reducing the number of preemptions incurred by DVS algorithms while achieving a high energy efficiency.

Categories and Subject Descriptors: D.4.9 [Operating Systems]: Systems Programs and Utilities

General Terms: Algorithms.

Keywords: Dynamic voltage scaling, low-power systems, real-time systems.

1. INTRODUCTION

Dynamic voltage scaling (DVS), which adjusts the supply voltage and clock frequency dynamically, is an effective low-power design technique for embedded real-time systems. Since the energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage, lowering the supply voltage is one of the most effective ways of reducing the energy consumption.

With a recent growth in the portable and mobile embedded device market, where a low-power consumption is an important design requirement, several commercial variable-voltage microprocessors were developed. Targeting these microprocessors, many DVS algorithms have been proposed or developed, especially for hard real-time systems [1, 2]. Since lowering the supply voltage also decreases the maximum achievable clock speed [3], various DVS algorithms for hard real-time systems have the goal of reducing supply voltage dynamically to the lowest possible level while satisfying the tasks' timing constraints.

Each DVS algorithm is known to be quite effective in reducing the energy/power consumption of a target system [2]. However, the existing DVS algorithms mainly focus on reducing energy consumption of CPU only, ignoring their negative impacts on system-wide energy consumption. For example, when the tasks' execution times are increased due to a lowered clock speed, the patterns of device usage and memory traffic may be changed, potentially increasing the energy consumption in system buses, I/O devices and memory chips. In particular, when a lower-priority task's execution time is extended with a lowered clock speed, it may be preempted more often by higher-priority tasks. According to the experiments reported in [4], the number of task preemptions can grow up to 500% under dynamic voltage scaling over non-DVS executions.

The increase in the number of task preemptions can negatively impact on the system energy consumption in several ways. First, the preemption overhead may increase the energy consumption in memory subsystems. In multi-tasking real-time systems, when a task is preempted by a higher priority task, the memory blocks used by the preempted task are displaced from the cache by the memory blocks used by the preempting higher priority task. Later, when the preempted task resumes its execution, a considerable amount of time is consumed to reload the previously displaced memory blocks into the cache. When preemptions are frequent due to the lengthened task execution time, cache-related preemption costs can take a significant portion of processor time and energy consumption in memory subsystems [5]. In addition, since the voltage scaling is performed usually at each context switching point, such frequent preemptions may degrade not only the system energy efficiency but also the system utilization when the voltage scaling overhead is not negligible as shown in [6].

Second, the lengthened task lifetime may increase the energy consumption in system devices. Since the execution of a preempted task should be delayed while a preempting task is running, its lifetime - the time interval between its activation and completion - is lengthened. If we assume that the system devices are active (i.e., powered up) dur-

*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. The ICT at Seoul National University provides research facilities for this study.

†This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation, and by University IT Research Center Project.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

ing the lifetime of the task (that use the system devices), the increased lifetime of the task may also increase the energy consumption in the system devices. Furthermore, as the number of simultaneously activated tasks increases, the number of active system devices is likely to increase, consuming more energy. In addition, since the code and data sections of the activated tasks should be kept in memory, the amount of active memory may increase, thus increasing the leakage power in memory subsystems.

Since the overhead of task preemptions has been an important issue in conventional real-time systems as well [7, 8, 5], there has been several research efforts to reduce the negative side effects of task preemptions. Y. Wand and M. Saksena proposed the preemption-threshold mechanism for Rate Monotonic (RM) scheduling in [7, 8]. In the preemption-threshold mechanism, each task has its own preemption-threshold. The currently running task is preempted only by a task whose priority is higher than the preemption-threshold of the current task. Although not impossible, it is very difficult to devise a DVS algorithm for this mechanism because the slack estimation is not straightforward. In [5], S. Lee *et al.* described limited preemption scheduling which minimizes the cache-related preemption delay of tasks so as to improve the system throughput. In this limited preemption scheduling, each task has non-preemptable code sections, which are expected to cause significant cache-related preemption delays when the task is preempted while executing these code sections. By prohibiting preemptions for these code sections, the system can reduce the cache-related preemption delay and can improve the system throughput. However, when the execution time of a non-preemptable code section is lengthened due to a lowered clock speed, the schedulability of given tasks is changing, making it very difficult to apply this technique to DVS algorithms in hard real-time systems.

In this paper, we address the problem of the increased task preemptions due to DVS algorithms. In order to control the number of preemptions, we propose two preemption control methods, *accelerated-completion* based technique and *delayed-preemption* based technique, which can reduce the negative impacts on energy consumption incurred by DVS algorithms. The accelerated-completion based technique tries to avoid preemptions by adjusting the voltage/clock speed *higher* than the lowest possible values computed using a given DVS algorithm, based on the task execution profile and the periodicity of tasks. On the other hand, the delayed-preemption based technique tries to *postpone* preemption points by delaying the activation of a higher-priority task as late as possible while guaranteeing the feasible schedule of tasks. By the delayed activation of higher-priority task, the scheduled task may avoid the preemption by completing its execution before the higher-priority task is activated. The experimental results show that the delayed-preemption technique can decrease a significant number of preemptions under dynamic voltage scheduling, avoiding potential negative side effects of DVS algorithms on system energy consumption.

The rest of this paper is organized as follows. In Section 2, we explain the motivation of this work. The proposed preemption control methods are described in Section 3. We present experimental results in Section 4 and conclude with a summary in Section 5.

2. MOTIVATION

2.1 System Model

We consider a preemptive hard real-time system in which periodic real-time tasks are scheduled under RM scheduling policy (that is, the shorter the task period is, the higher the task priority is.) [1]. 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 $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ where tasks are assumed to be mutually independent. In the task set $\mathcal{T}$, tasks are sorted based on their period length in a nondecreasing order, i.e., the priority of τ_i is higher than that of τ_{i+1}. Each task τ_i has its own period p_i and worst case execution time (WCET) w_i [2]. The relative deadline d_i of τ_i is assumed to be equal to its period p_i. Each task releases its instance periodically, and the j-th instance of τ_i is denoted by $\tau_{i,j}$. A task instance is denoted by a single subscript such as τ_α when no confusion arises. Each task instance τ_α has its own arrival time r_α and absolute deadline d_α. We denote $\alpha < \beta$ if $i < k$ where $\alpha \equiv i, j$ and $\beta \equiv k, l$.

2.2 Motivational Example

Since a DVS algorithm generally lowers the task execution speed, the execution time of a task will be increased under a DVS-enabled environment. Although the lowered execution speed is desirable for reducing the processor energy consumption, it may introduce negative side effects on the energy consumption of other system components. As we mentioned in the previous section, as the task execution time increases, the frequency of preemptions between tasks may increase. More frequent task preemptions, in turn, may incur considerable overhead due to increased memory accesses. Furthermore, the lengthened task lifetime may increase the energy consumption of the system devices which are used by the associated tasks.

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 0.7, 0.7, and 1.4 as shown in Table 1, respectively. Figure 1(a) shows the execution schedule of three tasks when DVS is not used. In this example, we assume that the actual execution time of each task is equal to its ACET. As shown in Figure 1(a), there are nine jobs in the hyper-period[3] , and the scheduling decision is made 10 times. That is, there is only one preemption in this case. However, when a DVS algorithm is used, the task schedule can be changed as Figure 1(c). Figure 1(c) shows the execution schedule of tasks when `lpWDA` algorithm [2] is used to schedule the processor voltage and clock speed. Unlike in Figure 1(a), the execution times of tasks are lengthened with lowered clock speeds and voltages. Especially, in the case of $\tau_{3,1}$, $\tau_{3,2}$, and $\tau_{2,3}$, due to their lengthened execution times, they cannot complete their executions before the arrival times of higher-priority tasks, thus being preempted by the corresponding higher-priority tasks. In Figure 1(c), there are four preemptions. Compared with Figure 1(a), there are three more preemptions.

[1]The proposed algorithm can be generalized to different preemptive priority-based scheduling policies such as EDF with minor modifications. In this paper, for the description purpose, we focus on the RM policy.

[2]We assume that tasks' execution times are based on f_{max}.

[3]Since the hyper-period is computed as LCM of tasks' periods, the task schedule will be similar in each hyper-period.

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

	period (p_i)	WCET (w_i)	ACET (a_i)
task 1 (τ_1)	3	1	0.7
task 2 (τ_2)	4	1	0.7
task 3 (τ_3)	6	2	1.4

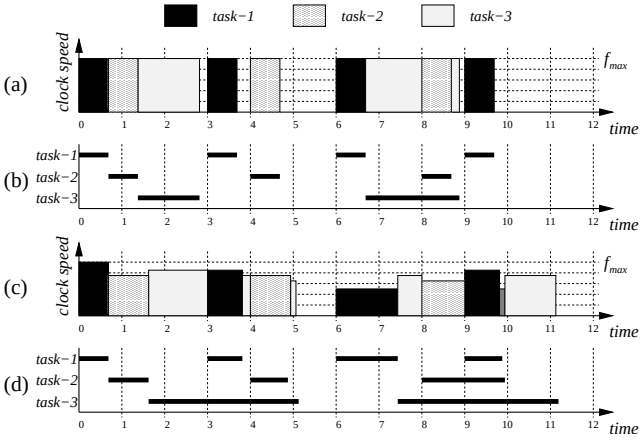


Figure 1: Voltage scheduling examples.

Assuming that the power consumption is proportional to the square of voltage (and clock speed is proportional to voltage), the schedule in Figure 1(c) consumes about 33% less processor energy than the schedule in Figure 1(a). However, when we consider the system-wide energy consumption including the system devices (which, we assume, are powered off when no associated tasks are running), the overall energy efficiency can be quite different, because the task lifetimes can be changed due to the DVS algorithm. Figures 1(b) and 1(d) illustrate the lifetime of three tasks under two different schedules of Figures 1(a) and 1(c), respectively. As shown in the figures, due to the lengthened task executions and the frequent preemptions, the sum of task lifetimes in Figure 1(d) is about 73% longer than that in Figure 1(b). Assuming that the energy consumption of each device is proportional to the lifetime of a task that uses the device and each task requires the same type of devices with the same power consumption during its lifetime, the schedule in Figure 1(c) consumes about 73% more energy from the system devices than the schedule in Figure 1(a). Furthermore, if we consider the increased memory accesses and memory leakage energy due to more frequent preemptions, the negative side effects on the system energy consumption become more significant.

One simple method to reduce the number of preemptions is to assign a higher voltage to the scheduled task which is expected to be preempted by higher priority tasks. However, such a technique is too conservative and may over-degrade the energy efficiency of a given DVS algorithm. in the following section, we present preemption control techniques which can reduce negative side effects of a given DVS algorithm by reducing the task preemptions in a schedule with a negligible degradation in the energy efficiency.

3. PREEMPTION CONTROL METHODS

The task preemption occurs when a lower-priority task could not complete its execution before the activation of a higher-priority task. Thus, we can approach the preemption control problem from two directions. The first approach is to shorten the completion time of a lower-priority task before the arrival time of a higher-priority task, by accelerating its execution. The second one is to delay the activation point of a higher-priority task so that a scheduled lower-priority task can complete its execution without the preemption. In this section, we propose two preemption control methods based on these two approaches for minimizing the number of context switchings.

Before describing the proposed techniques, we define the following notations.

- $slack(\tau_{i,j}, t)$: the amount of slack time that $\tau_{i,j}$ has at time t.
- $w_{i,j}^{rem}(t)$: the remaining worst case execution time of $\tau_{i,j}$ at time t.
- $f_{dvs}(t)$: the clock speed computed by a given DVS algorithm at scheduling time t.

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 arrived tasks, respectively. All the tasks are initially queued in *waitQueue*. When a task arrives, the task is moved from *waitQueue* to *readyQueue*, and the remaining WCET of τ_α is set to w_α, i.e., $w_\alpha^{rem}(t) = w_\alpha$. A task is said to *arrive* when the task releases its instance, while a task is said to be *activated* when the task starts its execution. As τ_α executes, $w_\alpha^{rem}(t)$ decreases and consumes its available execution time. When τ_α completes execution, its $w_\alpha^{rem}(t)$ is reset to 0.

3.1 Accelerated-Completion based Preemption Control Technique

A simple method to avoid the preemption is to accelerate the task execution so that the scheduled task can complete its execution before the next arrival time of a higher-priority task. The basic idea of this technique is similar to that of Stretching-to-NTA method used in [9]. The main difference is that the clock speed is adjusted based on the *worst case* execution time of the scheduled task and the next arrival time of a task (regardless of its priority) in [9], while the accelerated-completion technique adjusts the clock speed based on the execution time distribution of the scheduled task and the next arrival time of a higher-priority task.

When a task is scheduled at time t, the clock speed for the scheduled task can be adjusted within the range of $[f_{dvs}(t), f_{max}]$. As the adjusted $f(t)$ is close to $f_{dvs}(t)$ (or f_{max}), the probability $P_{preempt}$ of the scheduled task being preempted gets increased (or decreased). During online scheduling, $P_{preempt}$ largely depends on the task's execution time distribution, task's priority and the scheduling time t.

In our method, we determine the clock speed based on $P_{preempt}$ as follows. Let the cumulative density function of the execution time distribution [4] of τ_α be $cdf(\alpha, x)$, i.e., the probability that the execution time of τ_α is less than or equal to x is computed by $cdf(\alpha, x)$, and the earliest arrival time of the higher-priority tasks be $nhta_\alpha(t)$, i.e., $nhta_\alpha(t) =$

[4] In order to compute a cumulative density function, we may use a well known distribution function or construct it during online execution (e.g., as done in [10]). In this paper, we use the Gaussian distribution function.

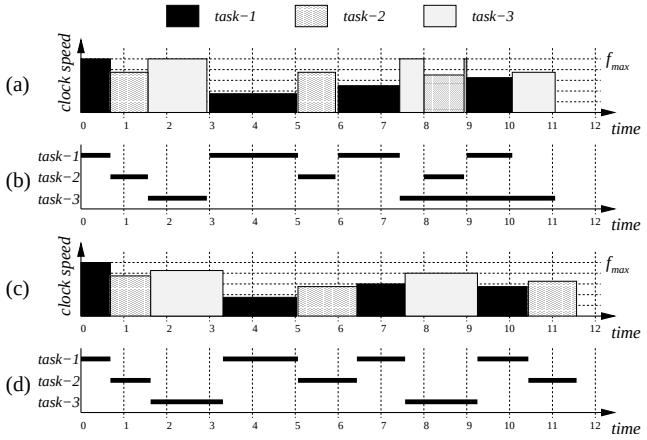


Figure 2: Voltage scheduling examples with the preemption control techniques.

$MIN\{nta_i(t)|nta_i(t) = \lceil \frac{t}{p_i} \rceil \cdot p_i \ \ , \ 0 < i < \alpha\}$. If $\theta(\alpha, p)$ is the inverse function of $cdf(\alpha, x)$, we adjust the clock speed to $f_{new}(t) = \frac{\theta(\alpha,p)-(w_\alpha - w_\alpha^{rem})}{nhta_\alpha(t)-t} \cdot f_{max}$. With the clock speed set to f_{new}, τ_α will complete its execution by $nhta_\alpha(t)$ with the probability of p. That is, if we want the scheduled task to avoid the preemption with the probability of p, the clock speed can be adjusted as

$$f(t) = \mathrm{MAX}(f_{dvs}(t), \mathrm{MIN}(f_{max}, f_{new}(t))). \qquad (1)$$

In this equation, as p is close to 1.0, the clock speed is adjusted to a higher speed in a pessimistic way, and the probability of preemption decreases while the processor energy consumption increases. That is, we can trade the number of preemptions in a hyper-period with the processor energy consumption. For example, in the situation that the portion of preemption-related energy consumption is relatively larger than that of the processor energy consumption, we may choose a higher value as p.

Consider the example shown in Figure 1 again. When we set $\theta(\alpha, p)$ value as ACET of τ_α, the task schedule can be changed as Figure 2(a) using the accelerated-completion technique. In order to avoid the preemption, the clock speeds for $\tau_{3,1}$ and $\tau_{3,2}$ are set to higher values than in Figure 1(c), i.e., 1.0. As a result, the number of task preemptions is reduced from four to two as shown in Figure 2(a), and the sum of task lifetimes is reduced by about 10% as shown in Figure 2(b), compared to that of Figure 1(d). Also, the processor energy consumption is reduced by 3.3%.

When the clock speed is set by using Eq. 1, the determined clock speed is usually higher than $f_{dvs}(t)$. Thus, it is clear that this method may reduce the number of context switchings. However, since most tasks are scheduled with higher clock speeds and voltages than in the original DVS schedule, the energy consumption in the processor core may increase. On the contrary, when a task is completed before the preemption, its unused time, which is uncertain before the task completion, can be utilized much earlier by the following higher-priority tasks than in the original DVS schedule. In this case, the increased energy consumption (from early completions of tasks) may be compensated by the more accurate slack estimation.

```
Algorithm :  Compute non-preemptable section
1.  τα : currently running active task
2.  tcur : current time
3.  When τβ is arrived at tcur
4.      IF ( pβ < pα ) THEN
5.          IF τβ's priority is highest among tasks in readyQ, THEN
6.              compute slack(τβ, tcur)
7.              IF ( slack(τβ, tcur) > 0 ) THEN
8.                  set [tcur, slack(τβ, t)] as non-preemptable section
9.              ELSE allow τβ preempt τα
10.                 (i.e., put τα into readyQ and schedule τβ )
11.             ENDIF
12.         ELSE put τβ into readyQ
13.         ENDIF
14.     ELSE put τβ into readyQ
15.     ENDIF
```

Figure 3: Delayed-preemption based preemption control algorithm.

3.2 Delayed-Preemption based Preemption Control Technique

The basic idea of the delayed-preemption method is to postpone the schedule of higher-priority tasks as late as possible so that the lower-priority task has more chance to be completed before the preemption. In order to compute the lowest possible clock speed which guarantees the feasible schedule of tasks, every DVS algorithm estimates slack times using its own manner. If a DVS algorithm determines the lowest possible clock speed as $f_{dvs}(t)$ for the scheduled task τ_α, the task can be said to have the slack time $slack(\tau_\alpha, t)$ given by

$$slack(\tau_\alpha, t) = \frac{w_\alpha^{rem}(t)(f_{max} - f_{dvs}(t))}{f_{dvs}(t)}. \qquad (2)$$

When tasks have slack times, even if their execution is delayed by the amount of slack, the schedulability of tasks is not affected. That is, when a higher-priority task τ_β arrives at t during the execution of a lower-priority task τ_α, the execution of τ_β can be safely delayed by $slack(\tau_\beta, t)$. If τ_α completes by $t + slack(\tau_\beta, t)$, the overall task schedule is still feasible[5].

In this method, when a task τ_β arrives, τ_β is scheduled with $f_{dvs}(t)$ if the system is idle. If not idle and τ_β has the highest-priority among all the arrived tasks and the running task τ_α, we compute $f_{dvs}(t)$ and $slack(\tau_\beta, t)$. Then, we set the time interval $[t, t + slack(\tau_\beta, t)]$ as a *non-preemptable section*, and make τ_α to keep running with the current clock speed which was computed at the previous scheduling point. Of course, if another higher-priority task τ_γ (whose priority is higher than τ_β) arrives during these non-preemptable section, the non-preemptable section is recomputed and can be changed. However, if the arrived task's priority is higher than τ_α but is not the highest among the tasks in *readyQueue*, the non-preemptable section is not affected. If the non-preemptable section is changed in this case, the tasks in *readyQueue* may violate their deadline in the future. When a task could not complete its execution until the end of the non-preemptable section, we can extend the non-preemptable section based on the slack time of the highest-priority task in *readyQueue*. The algorithm of the delayed-preemption method is summarized in Figure 3.

In the accelerated-completion based preemption control

[5]We assume that, even though the higher-priority tasks are delayed, if they do not miss their deadline, the system is still feasible.

technique, the resulting clock speed at each scheduling point is usually higher than the lowest possible clock speed. However, the delayed-preemption method changes preemption points only without modifying the speed computed by the DVS algorithm, thus the energy dissipation in processor core is not affected significantly. Furthermore, as in the accelerated-completion based preemption control technique, tasks complete their execution earlier than in normal cases, thus the unused times of tasks can be utilized earlier by the following tasks for energy saving.

Figures 2(c) and 2(d) show the task schedule using the delayed-preemption technique. When $\tau_{1,2}$ arrives at $t = 3$, $\tau_{3,1}$ is still running. In normal cases, $\tau_{1,2}$ will preempt $\tau_{3,1}$ at $t = 3$. However, since $\tau_{1,2}$ has 0.16 time units of the slack time, the activation of $\tau_{1,2}$ can be delayed to $t = 3.16$. At $t = 3.16$, since $\tau_{1,2}$ has 0.14 time units of additional slack time due to the reduced remaining execution time of $\tau_{3,1}$, the execution of the non-preemptable section can be extended to $t = 3.3$ and $\tau_{3,1}$ completes its execution at $t = 3.27$ without the preemption. The remaining tasks can be scheduled in a similar way, and the resulting schedule is shown in Figure 2(c). In this case, there is no preemption in the schedule as shown in the figure, and the sum of task lifetimes is reduced by about 20% as shown in Figure 2(d) compared to that of Figure 1(d). Also, the schedule in Figure 2(c) consumes 8% less energy than the schedule in Figure 1(c).

4. EXPERIMENTAL RESULTS

To evaluate the efficiency of the proposed preemption-control techniques, several experiments were performed for three cases: (1) when a DVS algorithm is used only (`lppsRM` [9], `ccRM` [11], and `lpWDA` [2], respectively), (2) when the accelerated-completion based preemption control technique was applied to `lpWDA` (denoted by `lpWDA-AC`), and (3) when the delayed-preemption technique was applied to `lpWDA` (denoted by `lpWDA-DP`). In this experiments, the probability value p of `lpWDA-AC` was chosen as 0.5. Experiments were performed using SimDVS, an integrated DVS simulation environment [12, 13]. The energy simulator in SimDVS 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 is scaled in the range of [1.1, 3.3] V. It is assumed that the system enters into a power-down mode when the system is idle. (The power consumption in the power-down mode is assumed to be zero.) In the experiments, the voltage scaling overhead is assumed negligible both in the time delay and power consumption.

We performed extensive experiments using synthesized application sets by varying the number of tasks in a task set and by varying the processor utilizations of sets. For a given number of tasks, 100 random task sets[6] were generated, whose utilization is 0.1∼0.9. In each experiment, the execution time of each task instance was randomly drawn from a Gaussian distribution[7] in the range of [0, WCET]. The results are given in Figure 4.

First, we estimated the energy efficiency of each algorithm for 8-task sets, and the results are given in Figure 4(a). In this figure, the x-axis represents the worst case processor utilization, and the y-axis represents the normalized energy consumption ratio to the energy consumption of non-DVS case. As shown, an aggressive DVS algorithm `lpWDA` saves the processor energy significantly. However, when the preemption control techniques are applied, the energy efficiencies are somewhat different. In the case of `lpWDA-AC`, its energy efficiency is about 5% worse than that of `lpWDA`, while the energy efficiency of `lpWDA-DP` is about 3% better than that of `lpWDA`. These difference can be explained as follows. In `lpWDA-AC`, in order to avoid the preemption, each task is scheduled with higher clock speeds and voltages than in the schedule of `lpWDA` while `lpWDA-DP` does not raise up the clock speeds and voltages. As a result, `lpWDA-AC` consumes more energy than `lpWDA`. However, in `lpWDA-DP`, since the lower-priority tasks are usually complete its execution before the preemptions, the unused time of the lower-priority tasks can be utilized by higher-priority tasks, so that the higher-priority tasks can be scheduled with lower clock speeds than in `lpWDA`.

Then, we estimated the changes in task scheduling due to DVS algorithms. Figure 4(b) shows the normalized number of context switchings of each algorithm to that of non-DVS case. However, in `lpWDA-DP`, we can see the number of context switchings was reduced to the same level of the non-DVS case. The effect of `lpWDA-AC` was rather disappointing.

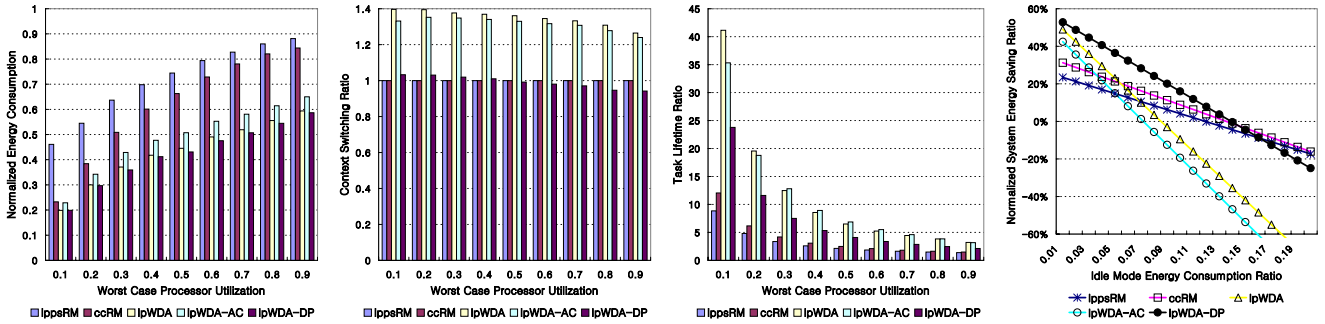
In order to see the impact of the low-speed execution and preemption on the task lifetime, we estimated the changed task lifetimes, and the results are given in Figure 4(c). The y-axis of Figure 4(c) represents the normalized task lifetime ratio to that of non-DVS case. The figure shows, as the processor utilization of the given task set decreases, the lifetime of tasks increases. Furthermore, as the energy efficiency of an algorithm is better, the lifetime of tasks also increases. However, in the case of `lpWDA-DP`, its energy efficiency is similar to or better than that of `lpWDA`, but the task lifetime does not increase; It actually decreased by about 35%. That means, using delayed-preemption technique, we can reduce the leakage power consumption of other devices in a system by reducing task lifetimes.

Finally, in order to understand the impact of task lifetime increases on the system energy consumption, we estimated the system energy consumption with the following simplifying assumptions: (1) Each device has three power states, active (high-power mode), idle (low-power mode), and standby (power-off mode). A device enters into the idle mode when the associated tasks are activated, and consumes the active mode power only when it is accessed. In other words, when a task execution time is lengthened due to a lowered clock speed, associated devices consume more energy in their idle mode but the energy consumption in their active mode is not changed. (2) When a task is idle (i.e., not activated), its associated devices enter into the standby mode and do not consume energy. (3) Each task does not share its devices with other tasks.

Usually, the power consumption of a device in its idle mode is much lower than in its active mode, and widely varies device by device (e.g., in the idle mode, WLAN card [14] and CT802 (Audio Chip) [15] consume 3% and 15% of active power, respectively). We define R_{idle} as the ratio of the idle mode power consumption to the active mode power consumption of a device (i.e., $Power_{idle} = R_{idle} \cdot Power_{active}$), and experiments were performed by varying R_{idle}. Figure 4(d) shows the evaluation results of algorithms for 8-task sets whose worst case processor utilization is 0.5. In this evaluation, we assume the CPU energy

[6] 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.

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



(a) Processor energy consumption for 8-task sets

(b) Context switchings for 8-task sets

(c) Lifetime for 8-task sets

(d) System energy consumption for 8-task sets

Figure 4: Experimental results.

consumption takes 50% portion of the system energy consumption in the non-DVS case[8]. In this figure, the x-axis represents R_{idle}, and the y-axis represents the normalized system energy saving ratio to that of the non-DVS case. As shown in Figure 4(d), as R_{idle} increases, the system energy saving benefit of each algorithm decreases. In particular, in the case of `lpWDA` and `lpWDA-AC` which increase task lifetimes significantly, the energy saving ratio decreases much faster than others as R_{idle} increases. `lpWDA` and `lpWDA-AC` consume more energy than the non-DVS case even when the idle mode energy consumption is relatively small (i.e., $R_{idle} < 0.085$ and $R_{idle} < 0.072$, respectively). However, in the case of `lpWDA-DP` which suppresses the task lifetime increase by avoiding unnecessary preemptions between tasks, its energy efficiency is higher than that of the non-DVS case for even larger R_{idle} values (up to 0.15).

5. CONCLUSIONS

We have discussed the possible side effects of a DVS algorithm on the system energy consumption, and presented the techniques that reduce the number of task preemptions while keeping the energy efficiencies of the existing DVS algorithms. For this, we proposed two preemption control methods, i.e., the accelerated-completion based technique and the delayed-preemption based technique. Our experimental results show that the delayed-preemption based technique can reduce the preemption-related system overhead, thus reducing the negative impacts of the DVS algorithms on the system level power consumption.

The work described in this paper can be extended in several directions. In this paper, we focused on `lpWDA`, but the proposed preemption control technique can be applied to other existing DVS algorithms (such as `ccEDF` and `laEDF` in [11]) as well. A similar study can be also performed using actual applications on a hardware DVS platform. Such a study will be useful to better understand the preemption effects on the energy consumption in the real system.

6. REFERENCES

[1] W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In *Proceedings of Design, Automation and Test in Europe*, pages 788–794, March 2002.

[2] W. Kim, J. Kim, and S. L. Min. Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 396–401, August 2003.

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

[4] W. Kim, J. Kim, and S. L. Min. Quantitative Analysis of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proceedings of the SoC Design Conference*, Novermber 2003.

[5] S. Lee, S. L. Min, C. S. Kim, C. G. Lee, and M. Lee. Cache-Conscious Limited Preemptive Scheduling. *Real-Time Systems*, 17(2/3):257–282, November 1999.

[6] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, pages 106–115, May 2003.

[7] Y. Wand and M. Saksena. Scheduling Fixed-Priority Tasks with Preemption Threshold. In *Proceedings of the Real-Time Computing Systems and Applications*, pages 328–335, December 1999.

[8] M. Saksena and Y. Wand. Scalable Real-Time System Design Using Preemption Thresholds. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 25–36, Novermber 2000.

[9] 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.

[10] J. Lorch and A. J. Smith. Improving Dynamic Voltage Scaling Algorithm with PACE. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2001 / PERFORMANCE 2001)*, pages 50–61, June 2001.

[11] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, pages 89–102, October 2001.

[12] W. Kim, D. Shin, J. Jeon, J. Kim, and S. L. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, pages 219–228, September 2002.

[13] D. Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min. SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms. In *Proceedings of Workshop on Power-Aware Computer Systems*, February 2002.

[14] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Evern-Driven Power Management. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 840–857, July 2001.

[15] DSP Group Corporation. CT8022 TrueSpeech CoProcessor. *http://www.dspg.com*, January 2004.

[8] The evaluation results in Figure 4(d) can be varied according to system model. Also, the overall energy consumption can be further minimized by employing an intelligent (or existing) power-management scheme for devices. However, in this paper, such a power-management scheme is not considered.