Research Article

Applying Dynamic Priority Scheduling Scheme to Static Systems of Pinwheel Task Model in Power-Aware Scheduling

Ye-In Seol¹ and Young-Kuk Kim²

¹ Green Energy Institute, Sangmyung University, Seoul 110-743, Republic of Korea

² Department of Computer Science & Engineering, Chungnam Nat'l University, Daejeon 305-764, Republic of Korea

Correspondence should be addressed to Young-Kuk Kim; ykim@cnu.ac.kr

Received 20 February 2014; Accepted 14 June 2014; Published 8 July 2014

Academic Editor: Jong-Hyuk Park

Copyright © 2014 Y.-I. Seol and Y.-K. Kim. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Power-aware scheduling reduces CPU energy consumption in hard real-time systems through dynamic voltage scaling (DVS). In this paper, we deal with pinwheel task model which is known as static and predictable task model and could be applied to various embedded or ubiquitous systems. In pinwheel task model, each task's priority is static and its execution sequence could be predetermined. There have been many static approaches to power-aware scheduling in pinwheel task model. But, in this paper, we will show that the dynamic priority scheduling results in power-aware scheduling could be applied to pinwheel task model. This method is more effective than adopting the previous static priority scheduling methods in saving energy consumption and, for the system being still static, it is more tractable and applicable to small sized embedded or ubiquitous computing. Also, we introduce a novel power-aware scheduling algorithm which exploits all slacks under preemptive earliest-deadline first scheduling which is optimal in uniprocessor system. The dynamic priority method presented in this paper could be applied directly to static systems of pinwheel task model. The simulation results show that the proposed algorithm with the algorithmic complexity of $O(n)$ reduces the energy consumption by 10–80% over the existing algorithms.

1. Introduction

Energy consumption issues are becoming more important for mobile or battery-operated systems. These mobile or portable embedded systems could be stand-alone or wireless networked. In the case of wireless networked system, much research effort was done on managing energy consumption of overall network [1, 2] or radio components [3]. On the other hand, energy consumption issues in the stand-alone embedded system or individual processing component in networked system have another side of problem because the applications in these systems have real-time requirements, so we should guarantee the applications from missing deadlines while reducing energy consumption.

Since the energy consumption of CMOS circuits, adopted in various microprocessors which are generally used in mobile or portable systems, has a quadratic dependency on the operating voltage ($E \propto V^2$) [4], it is a very useful method for reducing energy consumption to lower the operating voltage of circuits. But, lowering the operating voltage also

decreases its clock speed, so, the execution times of tasks are prolonged. This makes the problem more complex for embedded hard real-time systems, where timing constraints of tasks should be met. There has been significant research effort on dynamic voltage scaling (DVS) for real-time systems to reduce energy consumption while satisfying the timing constraints [5–12].

DVS algorithm depends on scheduling policy, task model, and processor architecture. Scheduling policy depends on the priorities of tasks which could be static or dynamic. Generally, static priority system is more manageable in embedded or ubiquitous systems, but more complicated or limited in the aspect of power-saving [13]. One of the static systems is the system of pinwheel task model and its system behavior is really static and predetermined in the sense of online task execution. So, there have been many static approaches to power-aware scheduling in pinwheel task model. But we will show that pinwheel task model could adopt the power-aware scheduling results of dynamic priority systems, so, it is more simple and effective in power-saving.

Among the dynamic priority scheduling policies, we consider earliest-deadline-first (EDF) scheduling policy [14] to be applied to pinwheel task model. EDF is known as optimal in uniprocessor system and has been applied to various task models, for example, periodic, sporadic, or aperiodic [15] task models. We present an algorithm which adopts and improves the result of CC-EDF [9] which is a power-aware version of EDF. The proposed power-aware scheduling method based on dynamic priority could be applied directly to static systems of pinwheel task model and can acquire more effective power-saving than the previous static schemes. The simulation results show that the proposed algorithm reduces energy consumption by 10–80% over the existing algorithms.

The rest of this paper is organized as follows. In Section 2, we present the system model and notations adopted in this paper and introduce the previous work which motivates the work done in this paper. In Section 3, we introduce an approach based on dynamic priority to the pinwheel task scheduling and present a power-aware scheduling algorithm which is applied directly to pinwheel task model. In Section 4, simulation results will be provided and Section 5 will conclude and discuss the future directions of this paper.

2. Motivation

In this section, we present the system model and introduce the result of related work.

2.1. System Model. Pinwheel task model was introduced in [16] to schedule distance-constrained real-time tasks. This kind of tasks should be executed in distance-constrained manner; that is, the distance of two consecutive jobs of a task, which is defined as the difference of their finishing times, is constrained.

In a distance-constrained task set $T_c = \{T_1, T_2, \dots, T_n\}$, every task T_i consists of infinite sequence of jobs $J_{i,1}, J_{i,2}, \dots$. Let $f_{i,j}$ denote the finish time of job $J_{i,j}$, for $1 \leq i \leq n$ and $j \geq 1$. Then, the distance between two consecutive jobs of a task T_i is limited by c_i ; that is, $f_{i,j+1} - f_{i,j} \leq c_i$. Also, there is a precedence constraint between consecutive jobs; that is, $J_{i,j+1}$ can be started only after job $J_{i,j}$ has been finished, and we assume that each job request is ready to be executed as soon as its precedent job is finished. Moreover, jobs are preemptable.

Based on the work done in [16, 17], pinwheel task model was applied to schedule successfully distance-constrained task set. By transforming distance-constrained task sets using pinwheel model, each task is converted to normal periodic task and periods of all tasks are harmonic [16, 17]. Then, rate-monotonic (RM) [14] scheduling policy of static priority systems could be applied.

As stated in [18], video systems need to ensure the interarrival time between frame relays is constrained rather than being just periodic. Also, some applications could not accept pinwheel transformation, because transformed period is shorter than original one. In this case, we may make a job of a task idle in order to prevent it from being executed too frequently [18].

Now, we can introduce periodic or sporadic task system because pinwheel task model could be considered as a special form of periodic task system. Each task of the system we considered is mutually independent. The target processor is DVS enabled uniprocessor and its supply voltage and frequency are varied continuously between $[v_{\min}, v_{\max}]$ and $[f_{\min}, f_{\max}]$, respectively. Let $T = \{T_1, T_2, \dots, T_n\}$ be a set of periodic or sporadic tasks. Each task is represented as $T_i = (P_i, C_i, D_i)$, where

- (i) P_i is period for periodic task, minimum interarrival time for sporadic task, or distance constraint of pinwheel task;
- (ii) C_i is the worst-case computation time for a task T_i at the maximum frequency;
- (iii) D_i is relative deadline of a task T_i .

If an instance or job of a task T_i is released at R_i , then its absolute deadline (d_i) is $R_i + D_i$. We will consider only tasks with $D_i = P_i$, so, a task T_i could be represented as (P_i, C_i) . Also, the following notations will be used.

- (i) TU_i : the worst-case utilization of a task T_i at the maximum frequency; that is, $TU_i = C_i/P_i$.
- (ii) TU : total utilization of all tasks in the system; that is, $TU = \sum_i TU_i$.
- (iii) CC_i : a task's actual computation time which should be less than C_i .
- (iv) CU_i : actual utilization of a task; that is, $CU_i = CC_i/P_i$.
- (v) CU : actual total utilization of the system; that is, $CU = \sum_i CU_i$.
- (vi) RC_i : a task's remaining computation time.
- (vii) α : current frequency ratio, that is, $f_{\text{cur}}/f_{\text{max}}$.
- (viii) TU^α : total utilization of all tasks when processor speed is α .

2.2. Related Work. The previous work on power-aware scheduling for pinwheel task model is based on offline algorithm [19], which could not utilize slacks of early completed tasks during run-time, or based on online algorithms which require high order of algorithmic complexity [18, 20] or deal with static priority tasks [21]. Our algorithm is based on dynamic priority scheduling scheme in spite of the fact that pinwheel task system is static one.

EDF, an optimal dynamic priority scheduling policy, has been extensively investigated in the area of real-time and power-aware scheduling [7–11, 14]. While devising a new power-aware scheduling algorithm, we especially considered the result presented by Pillai and Shin [9]. They introduced a cycle-conserving method to real-time DVS. This method reduces the operating frequency on each task completion and increases on each task release. When a task completes its current invocation after using CC_i computation time, they treat the task as if its worst-case execution time was CC_i . So, processor speed could be set as the actual total utilization CU which is always less than or equal to the worst-case total utilization TU .

FIGURE 1: CC-EDF schedule and its available slack.

Mei et al. [8] integrated the above cycle-conserving method and the result of Qadi et al. [10] for sporadic task set. But these methods do not fully utilize the slacks generated. Let us see Figure 1. If a task is complete at t_c , then the system has operated at higher frequency than required during the time interval $[R_i, t_c]$. This observation provides a clue to slow down the processor speed more when a task is complete. We will show later that the number of slacks which could be used for lowering processor frequency is related to temporal idleness of the completed task.

3. Dynamic Priority Scheme for Pinwheel Task Model

3.1. Behavior of Pinwheel Task Model. In this section, we investigate the property of pinwheel task model and its behavior of scheduling result. As stated in Section 2, pinwheel task model was introduced to schedule distance-constrained task set.

Let $T_c = \{T_1, T_2, \dots, T_n\}$ be a distance-constrained task set and let $\{c_1, c_2, \dots, c_n\}$ be the distance constraint of each task T_i , respectively. Without loss of generality, we can assume $c_1 \leq c_2 \leq \dots \leq c_n$. Then, using algorithm Sr [16], the original distance constraints are transformed to harmonic ones; that is, each c_i is 2^n multiples of one number Sr. This transformation reduces the distance-constrained scheduling problem into normal periodic task execution, where each periodic task's period is the same as the transformed distance constraint of corresponding distance-constrained task.

Suppose a distance-constrained task system consists of five tasks with distance constraints $\{9.2, 10.6, 10.8, 21.2, 22.6\}$ and their worst-case computation times are $\{1.5, 2.9, 1.7, 2.1, 1.7\}$, respectively. After pinwheel transformation [16], the five tasks will have distance constraints $\{5.3, 10.6, 10.6, 21.2, 21.2\}$. If we compose a periodic task set where each task's period is distance constraint of original task, optimal static priority scheduling policy like rate-monotonic scheduler [14] could be used to satisfy the distance constraints. Figure 2 shows the execution sequences generated by rate-monotonic scheduling for the above example when each task executes at its worst-case computation time. As you can see at this figure, the execution schedule for each task has no jitter; that is, each task's relative starting and ending times are fixed. So, the pinwheel task scheduling is more predictable compared to normal periodic task system with fixed priority.

It is known that if total utilization of a task system after pinwheel transformation is lower than or equal to 1, then

FIGURE 2: A scheduling result of pinwheel task model.

the task system is schedulable [16]. So, the following theorem holds immediately.

Theorem 1. *If TU of a pinwheel task system is α , one can schedule it with processor speed α .*

Proof. If we execute the same task at processor speed α instead of 1, each task's computation time is C_i/α . So,

$$TU^\alpha = \sum \frac{(C_i/\alpha)}{P_i} = \frac{1}{\alpha} \sum \frac{C_i}{P_i} = \frac{1}{\alpha} \cdot \alpha = 1. \quad (1)$$

□

By Theorem 1, we can always maintain total utilization of all tasks as 100% for the pinwheel task system in power-aware scheduling. This property is really the same as that of EDF scheduling. Now, we say that pinwheel scheduling result generated by static priority of rate-monotonic scheduling policy is the same as that generated by dynamic priority of EDF.

Theorem 2. *There is no difference between the scheduling results of rate-monotonic scheduling and EDF in case of pinwheel task model.*

Proof. Only at the beginning (or the end) of smallest period can start a pinwheel task and also can exist the deadline of each task in pinwheel task model because periods of all tasks are harmonic.

The priority of rate-monotonic scheduling is determined by the length of task's period and is static. That of EDF scheduling is determined by shorter deadline and is dynamic. Now, consider when a task preempts another task. Both rate-monotonic scheduling and EDF preempt lower priority task when a higher priority task arrives and execute the new arrived higher priority task. But the preemption of pinwheel task system can exist only at the beginning (or the end) of the smallest period because all tasks can start only there. The deadline of the higher priority task of rate-monotonic scheduling is always the same as or shorter than that of the preempted lower priority task because periods are harmonic. And if two tasks have the same deadline, let the task of smaller index have higher priority. So, if we index each task by its increasing period at EDF, the preempting higher priority task in rate-monotonic scheduling has also higher priority in EDF than the preempted lower priority task in the case of the same deadline.

For the cases of choosing the highest priority ready task when multiple tasks start at the same time and when

the previous task ends or an existing ready task resumes, the selected highest priority task of rate-monotonic scheduling has the shortest deadline and the smallest index because all periods are harmonic. So, all tasks have the same priority orders at every scheduling point between RM and EDF in case of pinwheel task model. $\square$

Theorem 2 implies that EDF scheduling system which indexes each task by ascending order of its period and chooses the task with the smallest index when two or more tasks have the same deadline is exactly the same as RM scheduling system for pinwheel task model. So, we can safely apply the result of EDF scheduling policy to pinwheel task system if we impose above two conditions. For the rest of this paper, we assume that EDF scheduling system satisfies them, but this assumption does not sacrifice generality because many or most of EDF systems adopt them.

3.2. Power-Aware Scheduling Algorithm. In this section, we introduce a new power-aware scheduling algorithm which adopts dynamic priority scheduling policy. But we already stated in Section 3.1 that the proposed algorithm could be applied to pinwheel task model without modification. Also, it is more effective than the previous work based on EDF as follows.

3.2.1. An Online Algorithm. As we stated in Section 2, the previous algorithms such as CC-EDF [9], DVSST [10], and CC-DVSST [8] do not fully utilize the slacks generated by early completed tasks. Before presenting more discussion, let us introduce a new definition.

Definition 3. Temporal idleness $TI_i(t)$ of a task T_i at time t is defined as follows:

0 until its completion and after its deadline,

$$\frac{RC_i}{d_i - t_c} \text{ if it was completed at time } t_c \text{ and } t = t_c, \quad (2)$$

$$TI_i(t_c) \pm \gamma \text{ if } t > t_c.$$

The real value of γ depends on the status of system and how to calculate it will be presented later. The following lemma computes the available slack at Figure 1 of CC-EDF and CC-DVSST algorithms.

Lemma 4. At CC-EDF or CC-DVSST scheduling, the amount of computation time exceeding its actual pace until its completion time ($= t_c$) is the same as $[TI_i(t_c) - (TU_i - CU_i)] \times (d_i - t_c)$.

Proof. Consider

$$\begin{aligned} & [TI_i(t_c) - (TU_i - CU_i)] \times (d_i - t_c) \\ &= \left[\frac{RC_i}{(d_i - t_c)} - \frac{C_i}{D_i} + \frac{CC_i}{D_i} \right] \\ & \times (d_i - t_c) \end{aligned}$$

$$\begin{aligned} &= [RC_i D_i - C_i R_i - C_i D_i + C_i t_c \\ & \quad + CC_i R_i + CC_i D_i - CC_i t_c] \\ & \times (D_i)^{-1} \\ & \quad \because R_i + D_i = d_i \\ &= [C_i(t_c - R_i) - CC_i(t_c - R_i)] \\ & \times (D_i)^{-1} \\ & \quad \because C_i = RC_i + CC_i \\ &= \left(\frac{C_i}{D_i} - \frac{CC_i}{D_i} \right) \times (t_c - R_i) \\ &= (TU_i - CU_i) \times (t_c - R_i), \end{aligned} \quad (3)$$

where “ $(TU_i - CU_i)$ ” is the height of the shaded square of Figure 1 and “ $(t_c - R_i)$ ” is the length of its base line, so, “ $(TU_i - CU_i) \times (t_c - R_i)$ ” equals exactly the area of the square and is the same as the amount of computation time exceeding its actual pace ($= CU_i \times (t_c - R_i)$). $\square$

Using Lemma 4, if a task T_i was complete at time t , then we can slow down processing speed by amount of TI_i when executing lower priority tasks than T_i until its deadline, because CC-EDF or CC-DVSST slows down processing speed by the amount of $(TU_i - CU_i)$. The proposed algorithm tries to use slacks of already completed higher priority tasks which are necessarily generated by assuming the worst-case execution scenarios and applies the result of CC-EDF when the running task's priority is higher than that of already completed tasks. Also, if we could not utilize those slacks by some reasons, that is, when executing higher priority tasks or when slacks are too large to fully utilize, then we evenly distribute those unused slacks until the corresponding deadlines.

One more consideration occurs when there is idle period. Let us consider a periodic task set $A = \{T_i = (P_i, C_i) \mid T_1 = (3, 1), T_2 = (3, 1), T_3 = (6, 2)\}$. If $T_{1,1}$ and $T_{2,1}$ are complete at $t = 1$ and $t = 2$, respectively, and $T_{3,1}$ is complete early at $t = 2.5$, then $TI_3(2.5) = 1.5/3.5 = 3/7$. If we lower the processing speed as much as $TI_3(2.5)$, then actual processing capacity during $t = [3, 6]$ is $(1 - 3/7) \times 3 = 12/7$ which is less than the sum of WCETs of $T_{1,2}$ and $T_{2,2}$.

Deadline missing occurs because there is idle period $t = [2.5, 3)$. During the idle period, the total processing capacity which should be processed under the actual execution scenarios is larger than the sum of slacks used. So, we should reduce the future slacks to compensate this mismatch. Figure 3 shows it.

The number of slacks which should be reduced is $TU - TI$ and is the same as $(CU - (TI - (TU - CU)))$. In Figure 3, the area “a” is the same as “b” + “c” and if TI exceeds TU , we could save some slacks because only “ $TU \times (\text{length of idle period})$ ” should be processed by CPU.


```

When task  $T_i$  arrived,  $T_i$  completed and at deadline
if (ex_flag) increase_temporal_idleness()
else if (cpu was idle) decrease_temporal_idleness()
insert  $T_i$  into TC (at completed) or delete (at deadline)
recompute  $TU = TU \pm C_i/P_i$  (when arrived or at deadline)
last_cpu_speed = compute_cpu_speed( $T_{cur}$ )
if there is no ready task to execute then set cpu as idle
else set cpu speed as last_cpu_speed
compute_cpu_speed( $T_j$ ) //  $T_j$  is the highest priority ready task or null
cpu_speed = CU
for all tasks  $T_i$  at TC
  if  $d_i \leq d_j$  or  $T_j$  is null
    cpu_speed -= ( $TI_i - (TU_i - CU_i)$ )
    if cpu_speed < 0 then ex_flag = 1; break;
  else
    ex_flag = 1; break;
return (cpu_speed) or (0 if cpu_speed < 0)
decrease_temporal_idleness()
idle_work = idle_period * last_cpu_speed
for all tasks  $T_i$  at TC
  if ( $TI_i - (TU_i - CU_i)$ )  $\geq$  idle_work
    decrease  $TI_i$  as much as idle_work; break;
  else
    idle_work -= ( $TI_i - (TU_i - CU_i)$ );  $TI_i = TU_i - CU_i$ ;
increase_temporal_idleness()
for each unused slack of tasks
  distribute unused slack evenly until deadline

```

ALGORITHM 1: A power-aware scheduling algorithm.

FIGURE 3: A slack reduction example.

Now, we will show briefly that the presented algorithm is correct. Theoretic full proof using temporal workload analysis could be found at our previous paper [22].

Theorem 5. *The algorithm presented, in Algorithm 1, schedules every periodic or sporadic task set using EDF if and only if $\sum_i C_i/P_i \leq 1$.*

Proof. “Only If” part: if $U > 1$, then EDF will not find a feasible schedule; therefore, our algorithm will not also find a feasible schedule because our algorithm is the same as EDF or DVSSST when tasks execute always at their worst-case execution scenarios; that is, $TI_i(t) = 0$ and $TU_i = CU_i$ for $\forall t$ and i .

“If” part: we will show that total amounts of executed computations using our algorithm are always larger than or equal to those of another ideal system which schedules always tasks without violating their timing constraints. Also, we will show that our algorithm executes jobs of higher priority tasks more than or the same as the ideal system, so, our scheduling algorithm also satisfies the real-time constraints.

Let us consider ideal CPU of ideal system which can anticipate each task’s actual computation time when it is released and can execute active tasks concurrently with the speed of CU. This ideal system of Figure 4 always satisfies all timing constraints and is really the most effective in power consumption.

Our real system starts to execute with TU speed and slows down as much as $(TU - TI)$ when a task is completed. And we already said that $[TI_i - (TU_i - CU_i)] \times (d_i - t_c) = B$ is exactly the same as $(TU_i - CU_i) \times (t_c - R_i) = A$. Therefore, we can apply the substitution operation which replaces some areas of A by the same areas of B to the scheduling result of our real system. Figure 5 shows it.

During the time interval $[t_c, d_i]$ which has no idle period, we can apply the above operation to all areas of A. So, total computing capacity of our algorithm is larger than or equal to that of the ideal system of Figure 4. And the computing capacity of the former is always exhausted by tasks which have shorter or the same deadlines of tasks of the latter.

During idle period, there exist some areas of ideal system which cannot find counterparts of real system. Those areas

FIGURE 4: Ideal CPU and ideal execution.

FIGURE 5: Slack exchange process.

could be filled up or substituted with the areas of future slacks of already completed tasks as shown in Figure 3 and the above exchange operation could be also applied to them. This substitution consisted of two steps as shown in Figure 6. Area "D" (future slack) is substituted by area "C" (idle slot) and is exchanged by area "A" (a portion of computation time exceeding its actual pace).

Now, total computing capacity of our real system is always larger than or equal to that of the substitution and exchange result. And the computing capacity of the former is always exhausted by tasks which have shorter or the same deadlines of tasks of the latter as we already stated. $\square$

3.2.2. An Illustrative Example. Let us consider an illustrative example, a pinwheel task system with distance constraints $\{9.2, 10.6, 10.8, 21.2, 22.6\}$ and the worst-computation times $\{1.5, 2.9, 1.7, 2.1, 1.7\}$, respectively, which is already introduced in Section 3. After pinwheel transformation, distance constraints of the tasks are $\{5.3, 10.6, 10.6, 21.2, 21.2\}$, respectively. Now, suppose that actual computation times of the tasks are always $\{0.75, 1.45, 0.85, 1.05, 0.68\}$, respectively. Total utilization of the tasks after pinwheel transformation is approximately 0.9. Figure 7 shows their execution sequences.

At $t = 0$, T_1 starts to execute with processor speed of $\alpha \approx 0.9$ and, at $t \approx 0.84$, it is completed. Then, CU_1 is 0.14 and TI_1 is 0.17, approximately. T_2 starts now to execute with processor speed of $\alpha \approx 0.728$ ($= TU - TI_1 = CU - (TI_1 - (TU_1 - CU_1))$) and, at $t \approx 2.83$, it is completed. Now then, CU_2 is 0.14 and TI_2 is 0.19, approximately. The processor speed for T_3 is $\alpha \approx 0.54$ ($= TU - TI_1 - TI_2 = CU - (TI_1 - (TU_1 - CU_1)) - (TI_2 - (TU_2 - CU_2))$). T_3 will be completed at $t \approx 4.4$. T_4 will be executed with processor speed of $\alpha \approx 0.4$ from $t \approx 4.4$ to 5.3 and will be preempted by T_1 there. Now, T_1 starts with processor speed of $\alpha \approx 0.57$ ($= TU - TI_2 - TI_3 = CU - (TI_2 - (TU_2 - CU_2)) - (TI_3 - (TU_3 - CU_3))$).

FIGURE 6: Slack exchange process for idle period.

FIGURE 7: An illustrative example.

Above process will be continued until $t = 10.6$. There, T_1 , T_2 , and T_3 will be released at the same time and T_1 will preempt T_5 . T_4 was already completed at $t \approx 8.4$ with CU_4 (≈ 0.05) and TI_4 (≈ 0.082). But T_1 could not use TI_4 because deadline of T_1 is shorter than that of T_4 . So, T_1 will be executed with processor speed of $\alpha \approx 0.85$ which is the same as CU and it will be completed at $t \approx 11.5$. During the execution of T_1 , T_4 's slack was not used, so, we can increase its temporal idleness TI_4 into 0.085, approximately. The unused slack is $(TI_4 - (TU_4 - TI_4)) \times \Delta t \approx 0.032 \times (11.5 - 10.6) \approx 0.03$ and is distributed evenly until deadline, so, increased ratio is $0.03 / (21.2 - 11.5) \approx 0.003$. Now, T_2 will be executed with processor speed of $\alpha \approx 0.64$ ($= TU - TI_1 - TI_4 = CU - (TI_1 - (TU_1 - CU_1)) - (TI_4 - (TU_4 - CU_4))$) because deadline of T_2 is the same as that of T_4 . T_2 will be completed at $t \approx 13.7$ and its temporal idleness is 0.195, approximately.

T_3 will be completed at $t \approx 15.65$. Now, T_3 resumes its execution and will be completed at $t \approx 15.69$. There is no ready task until $t = 15.9$ when T_1 is released. So, at $t = 15.9$, temporal idleness should be reduced. The last calculated CPU speed is 0.11 ($= TU - \sum T_i$), and the number of slacks which should be reduced is 0.023 ($= \alpha \times \Delta t = 0.11 \times (15.9 - 15.69)$). This slack is obtained from T_2 until deadline, so,

FIGURE 8: Simulation result for periodic, sporadic, and pinwheel tasks.

the recalculated TI_2' is 0.19 ($= TI_2 - 0.023/(21.2 - 15.9)$), approximately. T_1 will start at $t = 15.9$ with processor speed of $\alpha \approx 0.28$ and will be completed at $t \approx 18.55$. Note that the calculated processor speed is exactly "0" at $t \approx 18.55$.

4. Experimental Results

We evaluated our proposed algorithm using RTSIM [23] which is a real-time simulator. RTSIM can simulate the behaviors of dynamic voltage scaling algorithms as well as traditional real-time scheduling algorithms. In this simulation, it is assumed that a constant amount of energy is required for each cycle of operation at a given voltage. This quantum is scaled by the square of the operating voltage, consistent with energy dissipation in CMOS circuits ($E \propto V^2$) [4, 7]. Only the energy consumed by CPU was computed and any other sources of energy consumption were ignored. Also, we do not consider preemption overheads, task switch overheads, and operating frequency change overheads. It is also assumed that the CPU consumes no energy during idle period and its operating frequency range is continuous at $[f_{\min} = 0, f_{\max} = 1]$.

We compared our proposed algorithm with CC-EDF for periodic task model, with DVSST and CC-DVSST for sporadic task model, and with CC-EDF and CC-RM [9] for pinwheel task model. CC-EDF assumes periodic task model, and CC-DVSST is a direct result of CC-EDF and DVSST. DVSST and CC-DVSST assume sporadic task model, so, we compared them at sporadic task system. CC-RM is a cycle-conserving rate-monotonic scheduling algorithm, so, it adopts static priority task model. CC-RM could be applied to pinwheel task model because of its static property. In spite of the fact that CC-EDF is a dynamic priority scheduling policy, it could be applied to pinwheel task model by Theorem 2. So, we compared our algorithm with CC-RM and CC-EDF at pinwheel task model.

To evaluate the effect of number of tasks in the system, we generated 10 or 20 tasks for each comparison. Their periods or minimum interarrival times are chosen randomly in the interval [1-1000] ms. We divided each task of system into three groups to reflect more real environments. One group of tasks has short period in the interval [1-10] ms, another group of tasks has medium period in the interval [10-100] ms, and the last group of tasks has long period in the interval [100-1000] ms. The simulation was also performed by varying the load ratio of tasks, that is, the ratio of the actual computation time to the worst-case computation time. For the periodic and sporadic task simulations, the worst-case total utilization of system is always 1; that is, $TU = 1$. For pinwheel task model, the generated task sets were transformed by the single-number reduction technique [17], so, their periods are harmonic and their total utilization after transformation should be less than 1. If the total utilization of a task set after transformation exceeds 1, then it was discarded.

Figure 8 shows the simulation result for periodic, sporadic, and pinwheel task systems. For periodic task system, our algorithm always outperforms CC-EDF. For sporadic task system, our proposed algorithm outperforms both DVSST

and CC-DVSST. For DVSST, the ratio of energy saving is up to 70% and for CC-DVSST up to 10%. The effect of number of tasks on the system could be also neglected, but the number of CPU frequency changes of our algorithm was larger than that of DVSST and almost the same as that of CC-DVSST. So, the ratio of energy saving to DVSST could be decreased. But our algorithm has huge performance gain to DVSST, so, in spite of frequency change overheads, it is expected that our algorithm still outperforms DVSST at real environments.

For pinwheel task model, our algorithm always outperforms both CC-RM and CC-EDF. CC-EDF always outperforms CC-RM except the case of 20 tasks and 90% load ratio. For the number of CPU frequency changes, that of CC-RM is slightly lesser than those of CC-EDF and our algorithm in the range of small load ratio and almost the same in the range of large load ratio. As you can see, CC-EDF and our algorithm outperform CC-RM by up to 80% in the range of small load ratio, so, we can still insist that dynamic priority scheduling algorithms of CC-EDF and our proposed algorithm have huge performance gain in the power saving in the range of small and medium load ratios.

5. Conclusion

In this paper, we investigated the property of pinwheel task model and showed that this task model can adopt the scheduling results of dynamic priority systems. So, this pinwheel task system which is integrating static and dynamic scheduling results is simple enough to be applied to embedded or ubiquitous computing and is more effective in power saving than the systems of only static method. Also, we presented a power-aware scheduling algorithm for general periodic and sporadic task systems and applied it to pinwheel task model without modification. The proposed algorithm adopts the results of cycle conserving method (CC-EDF) and sporadic task scheduling (DVSST) and improves them. The simulation results show that the proposed algorithm outperforms existing algorithms by up to 10-80% with respect to CPU energy saving.

In the future, we would like to improve the proposed algorithm. This could be done if we assign all slacks generated by the early completed higher priority tasks into the task of the highest priority among the uncompleted ready tasks instead of evenly distributing them until the ends of deadlines. This method may lower processor frequency much more than the proposed algorithm. Then, there may be more chances to save power consumption.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported in part by the Industrial Strategic Technology Development Program (10041740, development of a software that provides customized real-time optimal

control monitoring services by integrating equipment in buildings with web service), funded by the Ministry of Trade, Industry, and Energy (MOTIE, Korea), and also supported in part by Research Fund of Chungnam National University, Korea.

References

- [1] G. H. S. Carvalho, I. Woungang, A. Anpalagan, and S. K. Dhurandher, "Energy-efficient radio resource management scheme for heterogeneous wireless networks: a queueing theory perspective," *Journal of Convergence*, vol. 3, no. 4, pp. 15–22, 2012.
- [2] B. Singh and D. K. Lobiyal, "A novel energy-aware cluster head selection based on particle swarm optimization for wireless sensor networks," *Human-Centric Computing and Information Sciences*, vol. 2, no. 1, pp. 1–18, 2012.
- [3] C. Liu, O. Granados, R. Duarte, and J. Andrian, "Energy efficient architecture using hardware acceleration for software defined radio components," *Journal of Information Processing Systems*, vol. 8, no. 1, pp. 133–144, 2012.
- [4] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," in *Proceedings of the 28th Hawaii International Conference on System Sciences*, vol. 1, pp. 288–297, Wailea, Hawaii, USA, January 1995.
- [5] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, 2004.
- [6] 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 the Conference on Design, Automation and Test in Europe (DATE '02)*, pp. 788–794, 2002.
- [7] C.-H. Lee and K. G. Shin, "On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS '04)*, pp. 319–327, December 2004.
- [8] J. Mei, K. Li, J. Hu, S. Yin, and E. H.-M. Sha, "Energy-aware preemptive scheduling algorithm for sporadic tasks on DVS platform," *Microprocessors & Microsystems*, vol. 37, no. 1, pp. 99–112, 2013.
- [9] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating System Review*, vol. 35, no. 5, pp. 89–102, 2001.
- [10] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS '03)*, pp. 52–62, December 2003.
- [11] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '04)*, pp. 653–658, January 2004.
- [12] F. Yao, A. Demers, and S. Shenker, "Scheduling model for reduced CPU energy," in *Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science*, pp. 374–382, October 1995.
- [13] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proceeding of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 219–228, 2002.
- [14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, 1973.
- [15] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, 1996.
- [16] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 814–826, 1996.
- [17] C.-W. Hsueh and K.-J. Lin, "An optimal pinwheel scheduler using the single-number reduction technique," in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 196–205, December 1996.
- [18] H. Lin and C. Hsueh, "Applying pinwheel scheduling and compiler profiling for power-aware real-time scheduling," *Real-Time Systems*, vol. 34, no. 1, pp. 37–51, 2006.
- [19] D. R. Chen and Y. S. Chen, "An efficient DVS algorithm for pinwheel task schedules," *Journal of Information Processing Systems*, vol. 7, no. 4, pp. 613–626, 2011.
- [20] S.-H. Yu, H.-H. Lin, and C.-W. Hsueh, "An application using pinwheel scheduling model," in *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS '04)*, pp. 683–689, July 2004.
- [21] D. R. Chen, Y. S. Chen, and M. F. Lai, "A practical slack-time analysis method for DVS real-time scheduling," in *Proceedings of the 18th International Conference on Real-Time and Network Systems*, pp. 139–148, 2010.
- [22] Y. I. Seol, J. U. Kim, and Y. K. Kim, "Temporal workload analysis and its application to power-aware scheduling," *International Journal of Embedded Systems and Applications*, vol. 3, no. 3, pp. 1–17, 2013.
- [23] "RTSIM: real-time system simulator," <http://rtsim.sssup.it>.