PAPER

# Enhanced Cycle-Conserving Dynamic Voltage Scaling for Low-Power Real-Time Operating Systems

**Min-Seok LEE**[†], ***Nonmember and*** **Cheol-Hoon LEE**[††a)], ***Member***

**SUMMARY** For battery based real-time embedded systems, high performance to meet their real-time constraints and energy efficiency to extend battery life are both essential. Real-Time Dynamic Voltage Scaling (RT-DVS) has been a key technique to satisfy both requirements. This paper presents EccEDF (Enhanced ccEDF), an efficient algorithm based on ccEDF. ccEDF is one of the most simple but efficient RT-DVS algorithms. Its simple structure enables it to be easily and intuitively coupled with a real-time operating system without incurring any significant cost. ccEDF, however, overlooks an important factor in calculating the available slacks for reducing the operating frequency. It calculates the saved utilization simply by dividing the slack by the period without considering the time needed to run the task. If the elapsed time is considered, the maximum utilization saved by the slack on completion of the task can be found. The proposed EccEDF can precisely calculate the maximum unused utilization with consideration of the elapsed time while keeping the structural simplicity of ccEDF. Further, we analytically establish the feasibility of EccEDF using the fluid scheduling model. Our simulation results show that the proposed algorithm outperforms ccEDF in all simulations. A simulation shows that EccEDF consumes 27% less energy than ccEDF.

***key words:*** *RT-DVS, low-power operating systems, real-time operating systems, real-time dynamic voltage scaling, ccEDF, cycle-conserving DVS, laEDF, EccEDF*

## 1. Introduction

As the global industry trend of information and communication technologies has been concentrated on mobile and handheld devices, which run on limited battery power, energy efficiency is becoming a key issue. Especially battery-based real-time systems require not only energy efficiency but also high performance to meet real-time constraints. To satisfy both requirements, studies have considered Real-Time Dynamic Voltage Scaling (RT-DVS). RT-DVS is a technique intensively explored by many researchers for real-time systems; it achieves energy-savings by scaling the voltage and frequency while maintaining real-time deadline guarantees [1], [2], [5], [7], [11]–[14]. Most recent microprocessors are implemented using CMOS technology because CMOS devices create less waste heat than other approaches, such as TTL or NMOS. The maximum frequency of a CMOS based processor depends on the supply voltage, so when a frequency is reduced, the processor can operate at a lower supply voltage. Since energy consumption E is proportional to the processor frequency and square of the supply voltage ($E \propto fV^2$), RT-DVS can potentially offer very large net energy savings, of cubic order, through frequency and voltage scaling [4].

Pillai and Shin [1] devised two novel algorithms: one is Cycle Conserving Earliest Deadline First (ccEDF), the other Look-Ahead Earliest Deadline First (laEDF). These algorithms reclaim the dynamic slack to scale the processor frequency. ccEDF is one of the most simple but efficient RT-DVS algorithms. It lowers the required operating frequency at the current scheduling point by recalculating the utilization using the actual computing time consumed by the task. Since the recalculation is performed only at each scheduling points, it can be easily implemented. laEDF is also an aggressive and efficient algorithm which tries to minimize current frequency by deferring work beyond next immediate deadline, but at the same time making sure that the future deadlines are met. Pillai and Shin [1] observed that, if the machine has a large number of voltage settings, ccEDF outperforms laEDF. With more settings in laEDF, the low frequency setting is closely matched, requiring high-voltage, high-frequency processing later in order to complete all of the deferred work, hurting performance. This observation enabled us to focus on enhancing ccEDF instead of laEDF in this paper.

The simple structure of ccEDF allows the algorithm to be easily and intuitively coupled with a real-time operating system without incurring any significant cost. It also approaches the high efficiency of static DVS in the DVS-EDF scheduling overheads [7]. When considering non-negligible voltage/frequency transition overheads, non ideal time accounting, and an integrated OS scheduling environment, DSR (Deadline Satisfaction Ratio) of RT-DVS can deviate from the theoretical performance. Nonetheless, ccEDF is very close to EDF, and exhibits low DSR degradation [8]. While ccEDF has these advantages based on its good structural simplicity, further improvements are needed. When a task is completed, ccEDF compares the actual utilization used in this invocation to the worst-case specification, and the unused utilization that was allotted to the task is used for reducing the operating frequency. It lowers the operating frequency by subtracting the unused utilization from the total utilization. The unused utilization is calculated by dividing the unused execution time by the period without considering the time needed to run the task. Here, we can know that ccEDF makes an excessively conservative assumption in calculating the unused utilization for ensuring deadline

Manuscript received May 22, 2013.
Manuscript revised October 15, 2013.

[†]The author is with ADD, Yuseong P.O.Box 35–1, Daejeon, 305–600, Korea.
[††]The author is with the Department of Computer Engineering, Chungnam National University, Daejeon, Korea.
a) E-mail: clee@cnu.ac.kr (Corresponding author.)

DOI: 10.1587/transinf.E97.D.480

guarantees when the operating frequency is reduced. If the elapsed time is considered, we can find the maximum unused utilization on completion of the task. In this paper, we present a more efficient algorithm called EccEDF (Enhanced ccEDF) that can precisely calculate the maximum unused utilization with consideration of the elapsed time while keeping the structural simplicity of ccEDF. We also prove that if the input task set is feasible, the algorithm EccEDF guarantees the deadlines of all the tasks, and show that EccEDF always outperforms ccEDF through a theoretical comparison and simulations.

The paper is organized as follows. In the next section, we present the system model considered in this paper and introduce the basic idea. In Sect. 3, we present details of our EccEDF algorithm, prove feasibility of the algorithm, and illustrate how it works. The theoretical comparison and simulation results are presented in Sect. 4. We conclude the paper in Sect. 5.

## 2. Motivation

### 2.1 System Model

We consider a preemptive hard real-time system with uniprocessor on which real-time tasks are scheduled under the EDF scheduling policy [5], [6]. The supply voltage and clock frequency of the processor can be continuously scaled within its operational ranges, $[v_{min}, v_{max}]$ and $[f_{min}, f_{max}]$. Let $\alpha()$ be a frequency scaling function such that the frequency is scaled down to $\alpha(t) \cdot f_{max}$ at time $t (0 \leqslant \alpha(t) \leqslant 1, \forall t)$. Also, let $\alpha_i$ be the frequency scaling factor of task $T_i$, as in [1], [5]. Each task $T_i$ has an associated period $P_i$, the worst-case execution time (WCET) $C_i$. The task is released periodically once every $P_i$ time units and its relative deadline $D_i$ is same as $P_i$. Tasks are assumed to be independent. Energy is consumed only by the processor. The processor does not consume energy during idle time since we consider a perfect machine in which a perfect software-controlled halt feature is provided. The context switching overheads and transition time for an operating voltage (DC/DC conversion time) and frequency (PLL locking time) are assumed to be negligible, so those can be included into the WCET of each task.

### 2.2 The Basic Idea

We use the fluid scheduling model [3], [10] to precisely calculate the available slack which is generated by early completion of a task. If we can linearly control an operating frequency of a processor and normalize it between 0 and 1, the utilization needed to process a task $T_i$ is same as the normalized operating frequency $f_i$. And if the actual execution time $cc_i$ which is consumed by $T_i$ equals to its worst-case execution time $C_i$, the minimum operating frequency required to guarantee the deadline of $T_i$ is $C_i/P_i$ [2]. It can be explained by the fluid scheduling model in which a task is executed at a constant rate at all times. Figure 1 shows the

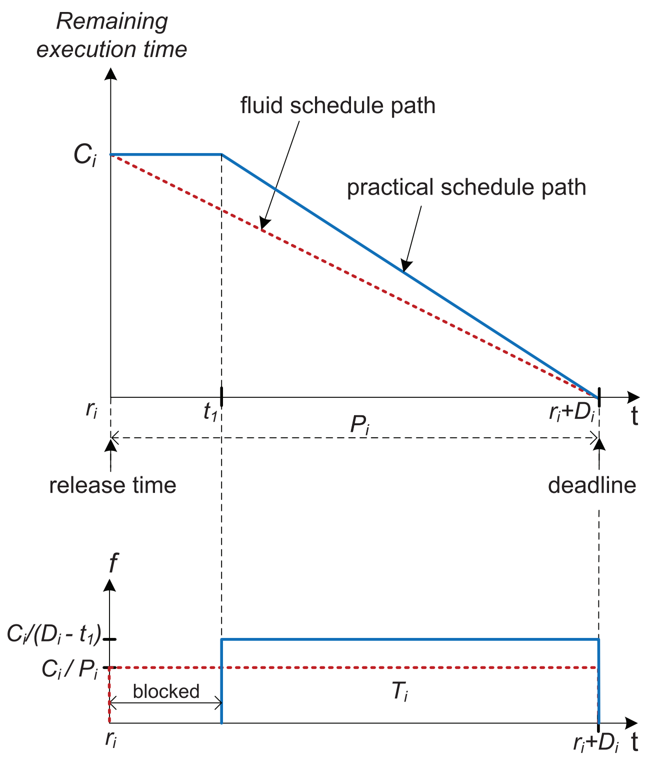


**Fig. 1** Fluid scheduling model.

fluid scheduling model and depicts the difference between the fluid schedule and a practical schedule. The figure represents time on horizontal axis and task's remaining execution time on vertical axis. The fluid schedule path is the dotted line which has the slope of $-C_i/P_i$. If $T_i$ is executed along its fluid schedule path, the task's remaining execution time at the release time is WCET($C_i$) and the remaining execution time at the deadline is 0.

What a real-time task is schedulable means the task's remaining execution time is 0 when the task arrives at the deadline. To that end, the operating frequency of the processor should be set to $C_i/P_i$. If the operating frequency is set to a lower value than $C_i/P_i$, $T_i$ will miss the deadline. On the other hand, If the operating frequency is set to a higher value than $C_i/P_i$, it causes the processor idle. In the RT-DVS system, though $T_i$ is not early completed, what an idle time is generated means that the processor has been overrun with a higher frequency than $C_i/P_i$ consuming more power. So generation of idle time is not desirable for low-power systems. Since it is important to meet the deadline with the minimum frequency in low-power systems, setting the frequency to $C_i/P_i$, that is, following the fluid schedule path is the best way to reduce power consumption if no task is completed earlier than its worst-case execution time.

In practical scheduling, the task can be blocked by other tasks that have a higher priority since a processor can execute only one task simultaneously. As shown in Fig. 1, if $T_i$ is blocked until $t_1$ by a task, remaining execution time is not reduced until $t_1$. After $T_i$ is dispatched at $t_1$, scheduler has to increase the operating frequency to compensate the blocked time. The minimum utilization of $T_i$ at $t_1$ to guarantee deadline is the slop of the recalculated fluid schedule path (the practical schedule path in Fig. 1), which is as follows:

$$\frac{\textit{remaining execution time}}{\textit{remaining time}} = \frac{C_i}{D_i - t_1} \tag{1}$$

## 3. Enhanced Cycle-Conserving RT-DVS

### 3.1 Calculation of the Utilization Saved by Slack Time

Most tasks in practical real-time systems are completed earlier than its worst-case execution time. In that case, while non-DVS systems waste the unused cycle on idling, RT-DVS systems use the extra processor cycle to lower the operating frequency. The key of RT-DVS is to exhaustively find and utilize the available slack.

Before explaining our algorithm, we will explore how the slack is used in the ccEDF algorithm. In ccEDF, utilization, $U_i$ on each task release and completion is $cc_i/P_i$, where $cc_i$ on task completion is the actual cycles used in this invocation, while $cc_i$ on task release is $C_i$. The total utilization of $T_i$ at a scheduling point is

$$U_{ccEDF} = \sum_{i=1}^{N} \frac{cc_i}{P_i},$$

which can be replaced as follows:

$$U_{ccEDF} = \sum_{i=1}^{N} \left\{ \frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i} \right\} \tag{2}$$

From Eq. (2), we can know that the utilization saved by the slack on current invocation is $(C_i - cc_i)/P_i$, where $(C_i - cc_i)$ is the slack generated by early completion of $T_i$. We can also know that ccEDF reduces energy expenditure by subtracting the saved utilization from the total utilization on task completion. It calculates the saved utilization simply by dividing the slack by the period without considering the time needed to run the task. If the elapsed time is considered, we can find the maximum utilization saved by the slack on completion of the task by calculating the minimum utilization needed to process the slack by its deadline using following Eq. (3) which is proved in Theorem 1.

$$U_{s,i} = \frac{C_i - cc_i}{P_i - E_i} \tag{3}$$

**Theorem 1:** When a task $T_i$ is early completed and $P_i > E_i$, the minimum utilization needed to process the slack by the deadline of $T_i$ is $U_{s,i} = (C_i - cc_i)/(P_i - E_i)$, where $E_i$ = Elapsed time of the task $T_i$.

**Proof:** When a task $T_i$ is released at $r_i$ but executed at $t_1$, the minimum utilization needed to process $C_i$ by its deadline is $C_i/(D_i - t_1)$ (see Fig. 2). But if $T_i$ is completed

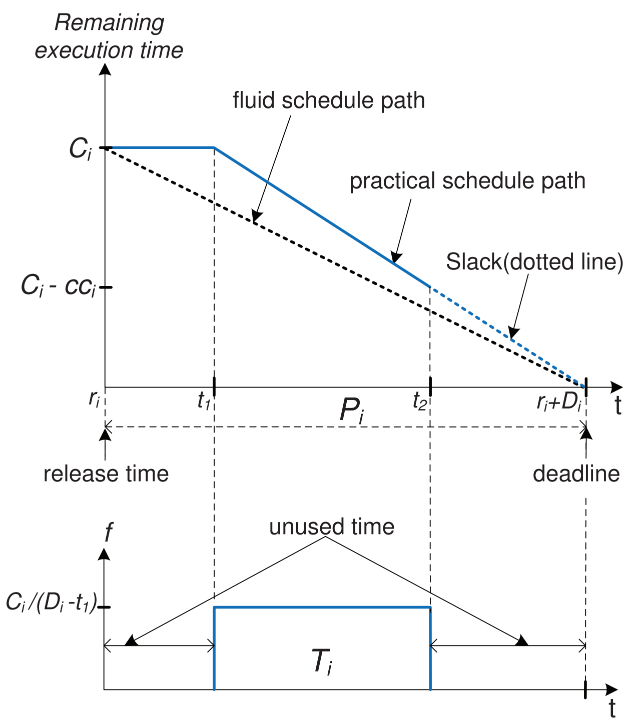


**Fig. 2** Fluid scheduling model with slack.

early at $t_2$, the available slack is $(C_i - cc_i)$, the elapsed time to run $T_i$ is $E_i = t_2 - t_1$, and the remaining execution time is $D_i - E_i$. Since the minimum utilization needed to process the slack by the deadline of $T_i$ is the slop of the fluid schedule path, it can be calculated by Eq. (1) such that $U_{s,i} = (\textit{remaining execution time})/(\textit{remaining time}) = (C_i - cc_i)/(P_i - E_i)$, where $P_i = D_i$ in case of a periodic task (see Fig. 3). □

If $\alpha$, a normalized processor frequency is larger than $U_{s,i}$ in processing the slack in Fig. 3, the slack is completed before the deadline. If $\alpha$ is less than $U_{s,i}$, the slack is not completed by the deadline. Consequently $U_{s,i}$ is the minimum utilization needed to process the slack. It also means the maximum utilization saved by the slack.

### 3.2 Enhanced Cycle-Conserving EDF

ccEDF can be tightly coupled with the operating system's task management services, since the algorithm may need to reduce frequency on each task completion, and increase frequency on each task release [1]. Such a simple structure is a strong point of the algorithm. The proposed EccEDF has also the same structure as ccEDF. Figure 4 shows the EccEDF algorithm. At task release, $U_i$ is computed using the specified WCET in the same way as ccEDF. At task completion, $U_i$ is recomputed by subtracting $U_{s,i}$ from $C_i/P_i$, the utilization calculated at task release, while $U_i$ is set to $cc_i/P_i$ in ccEDF. With the reduced utilization, we can find a lower operating frequency while guaranteeing the deadlines of all tasks. It is proved in the following Definition 1 and Theorem 2.

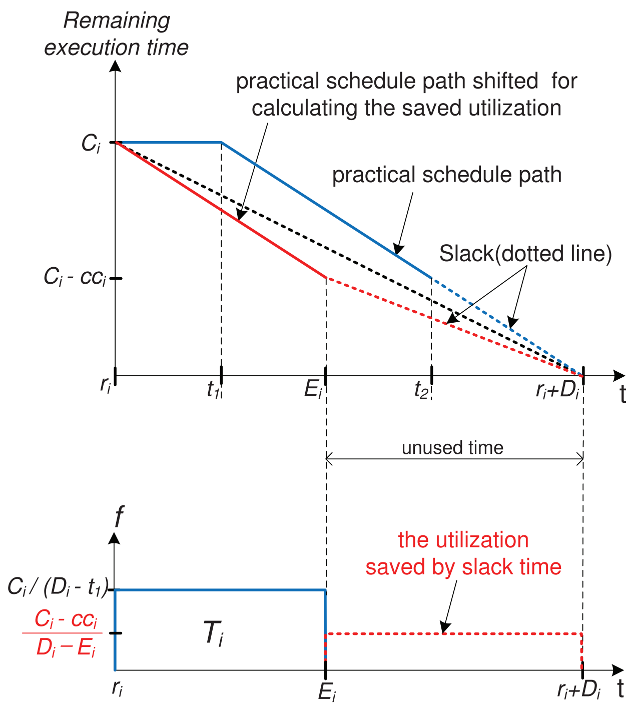


**Fig. 3** Calculation of the utilization saved by the available slack.

```
select_frequency() :
    use lowest freq. $f_i \in \{f_1, \cdots, f_m | f_1 < \cdots < f_m\}$
    such that $U_1 + \cdots + U_n \leq f_i / f_m$

upon task_release($T_i$) :
    set $U_i$ to $C_i/P_i$;
    select_frequency();

upon task_completion($T_i$) :
    set $U_i$ to $U_i - U_{s,i}$;
    select_frequency();
```

**Fig. 4** EccEDF DVS for EDF Schedulers.

**Definition 1:** At each EDF scheduling point, all tasks in a task set **T** are classified as followings.

- If $C_i = cc_i$, $T_i$ is classified into class 1.
- If $C_i > cc_i$, $T_i$ is classified into class 2.

Class 1 is a set of tasks that is not yet completed or completed without slack using its WCET and class 2 is a set of tasks that are early completed.

At each scheduling point, the tasks in class 2 are not treated as dynamic tasks but treated as static tasks because the tasks have been already completed and the scheduler can know the actual execution time. The tasks in class 1, however, are treated as dynamic tasks of which the actual execution time is WCET because the tasks are not completed and the scheduler cannot know $cc_i$.

**Theorem 2:** Each feasible set **T** of real-time tasks is feasibly schedulable by EccEDF.

**Proof:** Assume that if a task set **T** is feasible, then the task set is not feasibly schedulable by EccEDF. If it is not feasible, then there should be exist at least one deadline miss. Deadline miss implies that a total utilization $U_{EccEDF}$ calculated by EccEDF at a scheduling point is larger than $\sum_{i=1}^{N} C_i/P_i$. A feasible task set **T** can be classified at each EDF scheduling point as the following by definition 1.

$$\{T_1, \cdots, T_f\} \in class\ 1,$$
$$\{T_{f+1}, \cdots, T_N\} \in class\ 2$$

Since class 1 is a set of tasks that is not yet completed or completed without slack, the total utilization $U_{class1}$ of class 1 is calculated such that $\sum_{i=1}^{f} C_i/P_i$ by EccEDF. The total utilization $U_{class2}$ of class 2 is calculated as the following by EccEDF.

$$U_{class\_2} = \sum_{i=f+1}^{N} \left\{ \frac{C_i}{P_i} - U_{s,i} \right\} = \sum_{i=f+1}^{N} \left\{ \frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i - E_i} \right\}$$

By summing $U_{class1}$ and $U_{class2}$, we get $U_{EccEDF}$ calculated by EccEDF as follows:

$$U_{EccEDF} = U_{class1} + U_{class2} = \sum_{i=1}^{f} \frac{C_i}{P_i} + \sum_{i=f+1}^{N} \left\{ \frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i - E_i} \right\} \quad (4)$$

Eq. (4) can be rearranged as following.

$$U_{EccEDF} = \sum_{i=1}^{N} \frac{C_i}{P_i} - \sum_{i=f+1}^{N} \left\{ \frac{C_i - cc_i}{P_i - E_i} \right\} \quad (5)$$

Since

$$\sum_{i=1}^{f} \left\{ \frac{C_i - cc_i}{P_i - E_i} \right\} = 0,$$

$$U_{EccEDF} = \sum_{i=1}^{N} \left\{ \frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i - E_i} \right\} \quad (6)$$

, where

$$\sum_{i=1}^{N} \left\{ \frac{C_i - cc_i}{P_i - E_i} \right\} \geq 0.$$

Hence

$$U_{EccEDF} \leq \sum_{i=1}^{N} \frac{C_i}{P_i},$$

a contradiction. Therefore, the algorithm follows. □

From Eq. (6), we can know that EccEDF can be simply

**Table 1** Example Task set.

| Task | WCET($C_i$) | Period($P_i$) |
|---|---|---|
| 1 | 3ms | 8ms |
| 2 | 3ms | 10ms |
| 3 | 4ms | 14ms |

**Table 2** Actual execution time($cc_i$) of the example task set.

| Task | $cc_i$ on invocation 1 | $cc_i$ on invocation 2 |
|---|---|---|
| 1 | 0.7ms | 2ms |
| 2 | 1ms | 1ms |
| 3 | 2ms | 1ms |

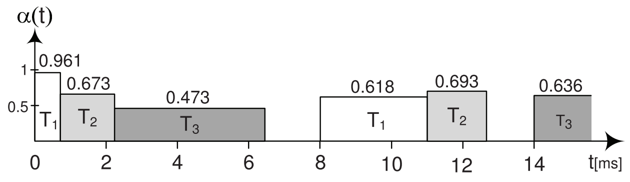


**Fig. 5** Example of ccEDF.

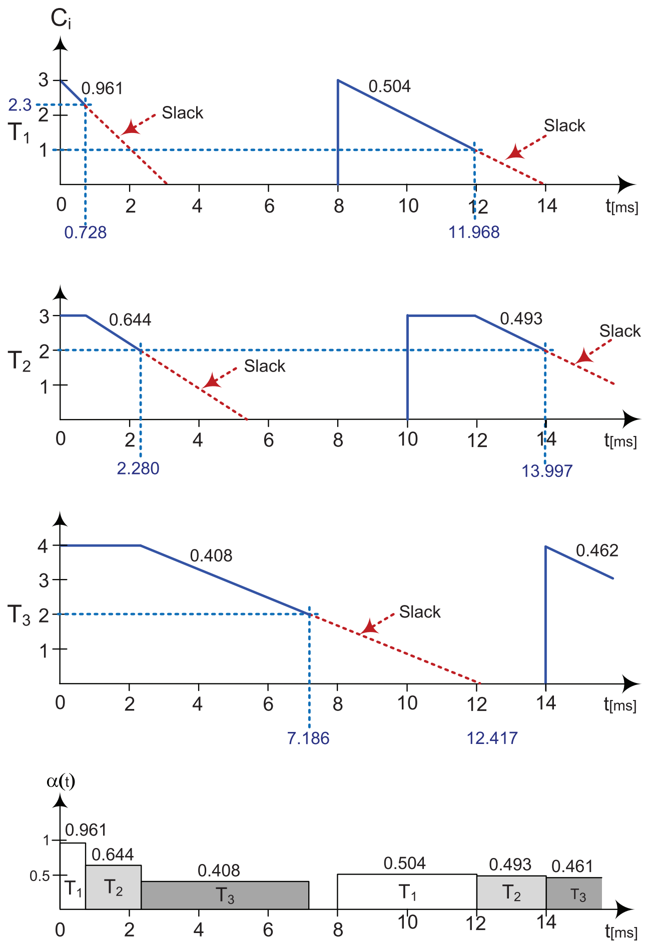


**Fig. 6** Example of EccEDF.

implemented by subtracting the saved utilization from the sum of the worst-case utilizations at each scheduling point. In practical implementations, the hardware voltage switching times may be the significant overhead. The proposed algorithm requires only two voltage transitions per task because calculation of $U_i$ is performed twice an invocation at task release and completion. These switching times also can be accounted for, and added to, the worst-case execution times for a practical implementation. The time complexity of the algorithm is $O(n)$, so we can easily incorporate it into a real-time operating system without incurring any significant scheduling overhead.

### 3.3 An Illustrative Example

Consider a task set shown in Table 1 and Table 2, where Table 1 indicates each task's period and the worst-case execution time. Although real-time tasks are specified with WCET on a scheduling point, they generally use much less than the worst case [1]. Table 2 shows each invocation's actual execution times which can be known to the system after the task completes execution. All tasks of the example task set are released at t = 0.

Figure 5 shows the result of ccEDF scheduling with the example task set. In Fig. 6, we illustrate how the EccEDF algorithm works with the task set. At time t = 0, the highest-priority task $T_1$ is released and starts to run with $\alpha = 0.961$, since $U_{EccEDF} = 3/8 + 3/10 + 4/14 = 0.961$ from Eq. (6). At time t = 0.728, $T_1$ is completed earlier than its WCET and the pended task $T_2$ starts to run with $\alpha = 0.644$, since $U_{EccEDF} = 0.961-(3-0.7)/(8-0.729)$. At time t = 2.280, $T_2$ is also completed earlier than its WCET and the pended task $T_3$ starts to run with $\alpha = 0.408$, since $U_{EccEDF} = 0.961 - (3-0.7)/(8-0.729)-(3-1)/(10-1.552)$. In this way, $T_1$ runs with $\alpha = 0.504$ at time t = 8, $T_2$ runs with $\alpha = 0.493$ at time t = 11.968 and $T_3$ runs with $\alpha = 0.461$ at time t = 14. When compared to ccEDF (Fig. 5), EccEDF (Fig. 6) could offer energy savings by more than 23% for this specific example.

## 4. Performance Comparison

### 4.1 Theoretical Comparison

In this section, we theoretically compare EccEDF to ccEDF before performing simulations in order to evaluate the proposed algorithm. From Eq. (2), we can know the total utilization calculated by ccEDF at a scheduling point is

$$U_{ccEDF} = \sum_{i=1}^{N}\left\{\frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i}\right\}$$

and from Eq. (6), the total utilization calculated by EccEDF is

$$U_{EccEDF} = \sum_{i=1}^{N}\left\{\frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i - E_i}\right\}.$$

$$U_{ccEDF} - U_{EccEDF} = \sum_{i=1}^{N}\left\{\frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i}\right\}$$

$$- \sum_{i=1}^{N}\left\{\frac{C_i}{P_i} - \frac{C_i - cc_i}{P_i - E_i}\right\}$$
$$= \sum_{i=1}^{N}\left\{\frac{C_i - cc_i}{P_i - E_i} - \frac{C_i - cc_i}{P_i}\right\}$$

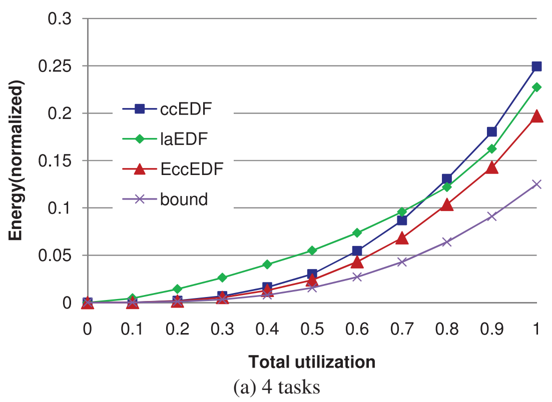
Since $E_i \geq 0$, $U_{ccEDF} - U_{EccEDF} \geq 0$. Therefore, we can know EccEDF always outperforms ccEDF in the energy savings from voltage scaling.

## 4.2 Simulation Results

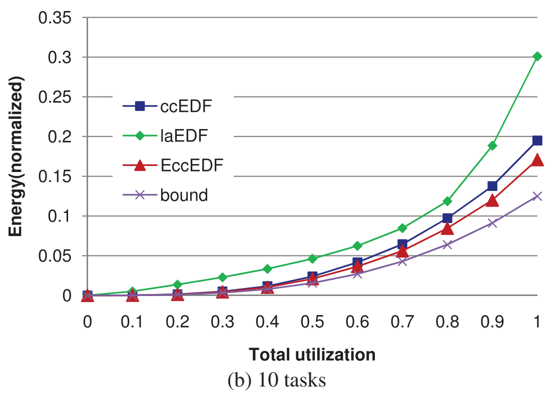
We have performed simulations to evaluate the EccEDF algorithm using RTSIM (Real-Time system SIMulator [15]) which is a real-time simulator. The simulation assumes that the energy consumption for each cycle is constant at a given voltage level. Only the energy dissipated by the processor is considered. This simplification allows that the task execution modeling can be reduced to computing cycles of execution without execution traces [5]. The simulation also assumes a perfect machine in that (i) the clock speed and the supply voltage can be varied continuously within its operational ranges $[f_{min}, f_{max}]$ and $[v_{min}, v_{max}]$, respectively; (ii) a perfect software-controlled halt feature is provided by the processor, so idle time consumes no energy, as in [1], [5]; (iii) the voltage and frequency scaling is performed only at each scheduling point; (iv) the task switch overheads, the preemption overheads and the time required for switching operating frequencies are not considered.

The periodic real-time task sets are specified by their period, worst-case execution time, and actual execution time. The task sets are randomly generated with periods ranging from 1 ms to 100 ms. It simulates various tasks found in real-time systems, such as a real-time computer embedded on a missile system, which is a typical multi-rate system requiring multiple sampling rates (e.g., 600 Hz for a control-law computation, 100 Hz for a guidance-law computation, and 10 Hz for inertial aiding data). Within the range, task periods are uniformly distributed. In order to see how well the proposed algorithm takes advantage of task sets that do not consume their worst-case execution times, we defined the load ratios as $cc_i/C_i$ and generated them using Gaussian probability distribution between 0.1 and 0.9. The number of tasks in the task sets are composed of n = 4, 10 and 15 tasks. The number is set based on the task sets commonly observed in video phone, missile, and avionics applications. We simulated 300 task sets for each point in graphs to improve accuracy of evaluation.

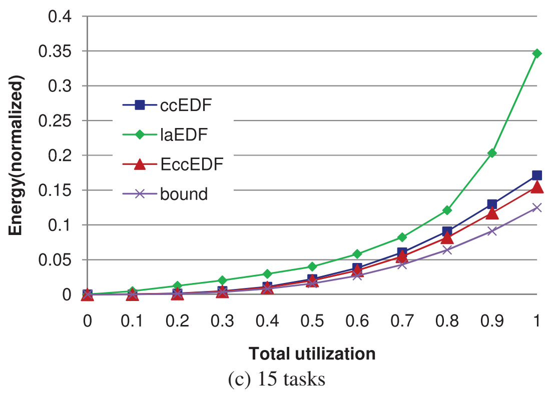
In our first set of simulations, we evaluated the difference in efficiency between ccEDF and EccEDF under conditions of varying the number of tasks in the task sets. Furthermore, we compared the energy efficiency of EccEDF and laEDF [1], one of the most efficient RT-DVS algorithms, to clear the advantage of EccEDF. Figure 7 shows the normalized energy consumption of ccEDF, laEDF, EccEDF and bound for task sets with 4, 10, and 15 tasks when the total utilization is varied between 0.1 and 1.0 and the load ratio is 0.5. Bound is the theoretical lower bound on energy consumption. Although bound cannot be implemented as a practical algorithm, it is used as a criterion in our simulations. As expected from the previous theoretical comparison, we can know that EccEDF outperforms ccEDF in all ranges and approaches bound. These simulations show that EccEDF consumes 22%, 14%, and 10% less energy at 4, 10, and 15 tasks respectively than ccEDF, still meeting the deadline guarantees of a real-time system. From the results, we can notice that EccEDF provides higher efficiency compared to ccEDF when the number of tasks decreases. Figure 7 also shows that EccEDF outperforms the laEDF algorithm in all ranges. In the figure, the energy efficiency of laEDF deteriorates as the number of tasks increases. Since tasks tend to use less execution time than their WCET, laEDF has



(a) 4 tasks

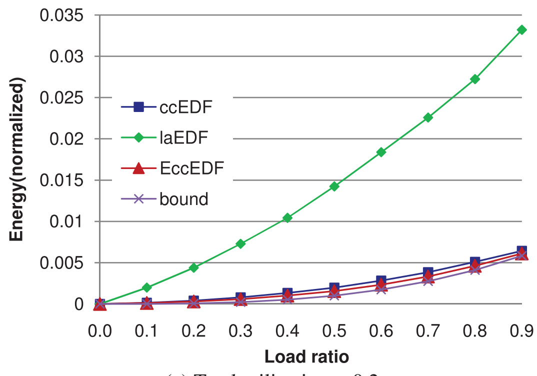


(b) 10 tasks

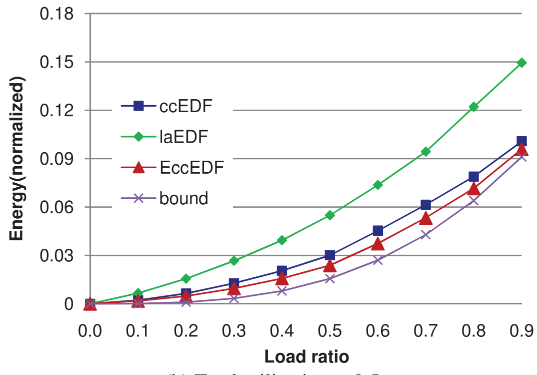


(c) 15 tasks

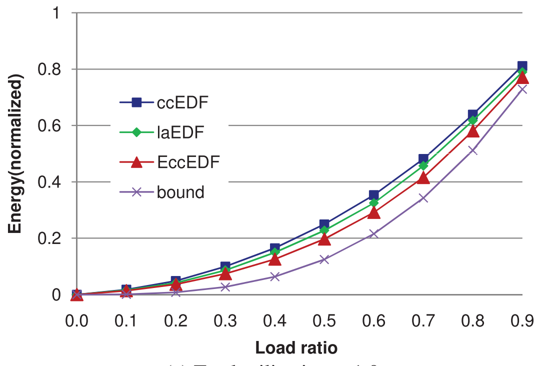
**Fig. 7** Energy consumption with 4, 10 and 15 tasks when load ratio = 0.5.



(a) Total utilization = 0.2

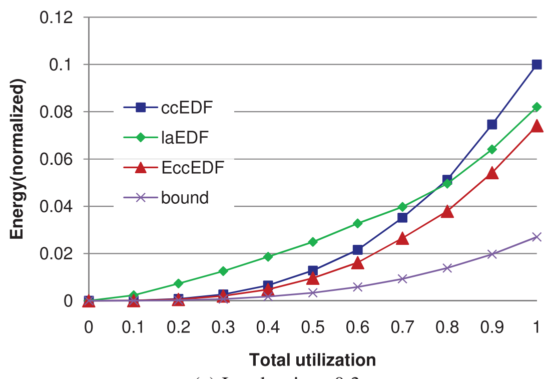


(b) Total utilization = 0.5

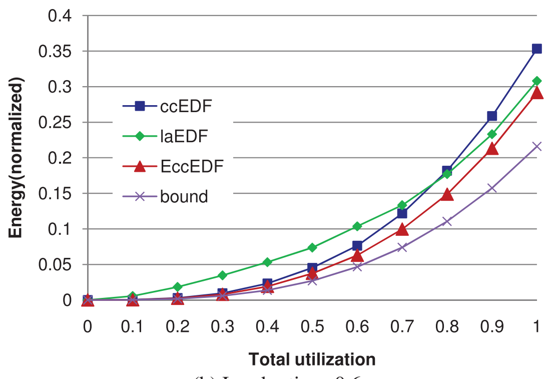


(c) Total utilization = 1.0

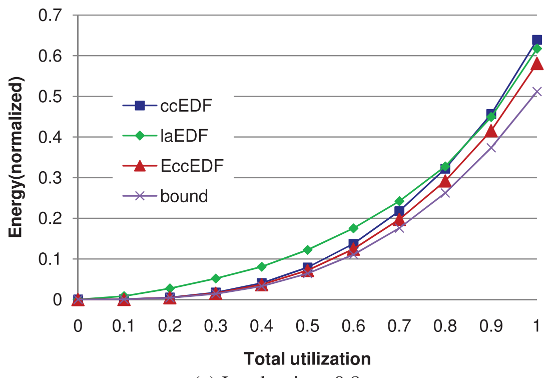
**Fig. 8** Normalized energy consumption with $U_t$ of 0.2, 0.5, and 1.0.



(a) Load ratio = 0.3



(b) Load ratio = 0.6



(c) Load ratio = 0.8

**Fig. 9** Normalized energy consumption with load ratios of 0.3, 0.6, and 0.8.

more slack sources for lowering the frequency setting as the number of task increases. With more settings like the machine considered in this paper, if the low frequency setting is closely matched, the performance of laEDF is degraded as observed in [1].

To see how the total utilization affects the energy efficiency of EccEDF compared to ccEDF and laEDF, we performed several simulations varying $U_t$ between 0.1 and 1.0. Figure 8 shows the normalized energy consumption of ccEDF, laEDF, EccEDF and bound with $U_t$ of 0.2, 0.5, and 1.0 when the load ratio is varied between 0.1 and 0.9 and the number of task is 4. In these simulations, EccEDF outperforms ccEDF, and a result shows that EccEDF consumes 27% less energy than ccEDF. While the number of tasks affects the performance of EccEDF in the preceding simulations, the variance of the total utilization has very little effect when compared to ccEDF's results. Figure 8 also shows that EccEDF offers better performance than that of laEDF, of which the energy efficiency is degraded as the total utilizations of tasks decrease. This can be explained by the fact that, as the utilizations decrease, laEDF can defer much work and lower the operating frequency, which results in high-frequency processing later in order to complete all of the deferred work, hurting performance.

In this set of experiments, we varied $cc_i/C_i$ between 0.1 and 0.9 to see how the load ratio affects the energy efficiency. Figure 9 shows the simulation results for task set with load ratios of 0.3, 0.6, and 0.8 when the total utilization is varied between 0.1 and 1.0 and the number of task is 4. In these simulations, we can observe that EccEDF outperforms ccEDF and EccEDF is more efficient compared to ccEDF when the load ratio is small. Figure 9 also shows that EccEDF works better than laEDF in all ranges.

## 5. Conclusions and Future Directions

We presented an efficient algorithm called EccEDF for real-time dynamic voltage scaling, which is designed based on ccEDF. While ccEDF is one of the most simple but efficient RT-DVS algorithms, it overlooked the time needed to run the task in calculating the available slack. In order to solve the issue, we developed the EccEDF algorithm; it can precisely calculate the available slack with consideration of the elapsed time while keeping the structural simplicity of ccEDF. The main contribution of this paper is that the proposed algorithm can find the maximum unused cycles transferred to the remaining tasks to rescale the frequency on completion of the task and the feasibility of the algorithm is analytically established. Our simulation results show that the proposed algorithm achieves great energy savings and outperforms ccEDF in all ranges. The performance gap becomes much larger as the number of tasks decreases or the load ratio decreases. A simulation shows that EccEDF consumes 27% less energy than ccEDF. In addition, we evaluated the energy efficiency of EccEDF and laEDF to clear the advantage of EccEDF. In the simulations, EccEDF shows better performance than laEDF in all ranges.

In future, we would like to implement the proposed algorithm. Generally, processing power of a system is inconsistent with energy efficiency. Frequent changes in the operating frequency cause significant switching overheads which resulted from the physical limitation of real systems. Therefore, evaluating how the switching overheads affect the energy efficiency of EccEDF is required by incorporating the algorithm into a real-time operating system.

### References

[1] P. Pillai and K.G. Shin, "Real-time dynamic voltage scaling forlow-power embedded operating systems," Proc. 18th ACM Symposium on Operating System Principles (SOSP'01), pp.89–102, 2001.
[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," IEEE Trans. Comput., vol.53, pp.584–600, 2004.
[3] P. Holman and J.H. Anderson, "Adapting Pfair scheduling for symmetric multiprocessors," J. Embedded Computing, vol.1, no.4, pp.543–564, May 2005.
[4] T.D. Burd and R.W. Brodersen, "Energy efficient CMOS microprocessor design," Proc. 28th Hawaii Int'l Conf. on System Sciences, pp.288–297, 1995.
[5] C.-H. Lee and K.G. Shin, "On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm," Proc. 25th IEEE Int'l Real-Time System Symposium (RTSS'04), pp.319–327, 2004.
[6] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," J. ACM, vol.20, no.1, pp.46–61, 1973.
[7] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling," ACM Conference on Languages, Compilers, and Tools for Embedded Systems 2005 (LCTES'05), pp.203–212, 2005.
[8] S. Saha and B. Ravindran," An experimental evaluation of real-time DVFS scheduling algorithms," SYSTOR, 2012.
[9] J.W.S. Liu, Real-Time Systems, Prentice-Hall, 2000.
[10] H. Cho, B. Ravindran, and E.D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," Proc. 27th IEEE Real-Time Systems Symposium, pp.101–110, Dec. 2006.
[11] H. Aydin, et al., "Dynamic and aggressive scheduling techniques for power-aware real-time systems," Proc. IEEE Real-Time Systems Symposium (RTSS'01), 2001.
[12] W. Kim, J. Kim, and S.L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," Proc. Design Automation and Test in Europe (DATE'02), pp.788–794, 2002.
[13] T. Pering and R. Broderson, "Energy efficient voltage scheduling for real-time operating systems," Proc. 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, (Work in Progress Session).
[14] V. Swaminathan and K. Chakrabarty, "Real-time task scheduling for energy-aware embedded systems," Proc. IEEE Real-Time Systems Symposium (Work-in-Progress Session).
[15] RTSIM: Real-time system simulator, http://rtsim.sssup.it.

**Min-Seok Lee** received the B.S. and M.S. degrees in Information and Telecommunication Engineering from Chonbuk National University, Korea in 1994 and 1996, respectively. Since 1996, he has been with Agency for Defense Development, where he is currently a principal researcher. His research interests are in the areas of embedded systems, real-time scheduling, and power-aware computing.

**Cheol-Hoon Lee** received the B.S. degree in Electronics Engineering form Seoul National University, Seoul, Korea, in 1983 and the M.S. and Ph.D. degrees in computer engineering from KAIST, Daejeon, Korea, in 1988 and 1992, respectively. From 1983 to 1994, he worked for Samsung Electronics Company in Seoul, Korea, as a researcher. From 1994 to 1995 and from 2004 to 2005, he was with the University of Michigan, Ann Arbor, as a research scientist at the Real-time Computing Laboratory. Since 1994, he has been a professor in the Department of Computer Engineering, Chungnam National University, Daejeon, Korea. His research interests include parallel processing, operating system, real-time system, and fault tolerant computing.