Core State Aware Slack Gathering Scheduling for Embedded Real-Time Systems

Kentaro Matsui[1,a)] Hideki Takase[1,b)] Kazuyoshi Takagi[1] Naofumi Takagi[1]

Abstract:
In recent embedded real-time systems, it is important to reduce energy consumption and achieve high performance. One feature often available in modern processors is low power states with varying transition cost for entering and leaving said low power states. In this paper, we study the problem on how to determine an energy management policy for a singlecore platform that supports low power states. To solve this problem, we first develop a novel DVFS method to accurately calculate task's slack time while guaranteeing system's deadline constraint. We then present an approach to decide the operation frequency and power state transition that reduce energy consumption. Finally, we develop a scheduling method to reduce energy consumption and guarantee the deadline constraint of a real-time system. Simulation results showed that our proposed method can reduce energy consumption by 20% at the maximum and 8% at the average compared with an existing method.

Keywords: Real-time system, Dynamic voltage/frequency scaling(DVFS), dynamic power management(DPM), energy efficiency, earliest deadline first scheduling

1. Introduction

In recent embedded real-time systems, it is important to achieve high performance and minimize energy consumption. To achieve these objectives, recent processors have introduced efficient supports for low power states that can reduce the power consumption to near zero. When using these processors, dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM) are typical power management techniques. DVFS reduces dynamic power consumption by scaling the operating frequency and voltage when the CPU is active and busy doing something, and DPM reduces stand-by power consumption by transitioning into low power states when the CPU is idle.

Previous work (explained in section 2) has reported results that demonstrate DVFS and DPM's ability to achieve energy savings while keeping the performance degradation under acceptable limits. However, it is still not clear what operating frequency and power state transition are optimal for the reduction of total power consumption because the relation between DVFS that reduces energy by minimizing system idle time and DPM that reduces energy by maximizing system idle time is a trade-off relation. Hence, for further energy saving, it is necessary to analyze tradeoff between DVFS and DPM's energy reduction effect and find an optimal power management policy.

In this paper, we propose an energy-efficient real-time task scheduling algorithm for a singlecore platform that supports multiple power states. Our proposed method consists of Slack Gathering laEDF (SGlaEDF) and Core State Aware Scheduling (CSAS). SGlaEDF is an algorithm that calculates task's slack time more accurately and guarantees the deadline constraint by updating the next deadline and remaining execution time when a periodic task execution is completed. CSAS is a scheduling algorithm that decides the operating frequency and power state transitions so as to reduce overall energy consumption of the system. In CSAS, the operating frequency that is calculated by SGlaEDF is used to guarantee the deadline constraint of the real-time system. We demonstrate that our proposed methods based on analysis of tradeoff between DVFS and DPM provide better system energy savings/performance tradeoffs than an existing method.

The rest of the paper is organized as follows. We discuss the related work in section 2 followed by the some preliminary background closely related to this paper in section 3. Section 4 presents our approach to reduce energy followed by the experimental setup and results given in Section 5. We conclude and provide future directions in Section 6.

2. Related Works

To improve the energy consumption reduction effect of DVFS and DPM, it is important to calculate task's execution time and slack time accurately. To solve this problem, various methods have been proposed.

Hayashi *et al.* [1] and Woonseok *et al.* [2] proposed an approach to calculate a task's slack time by using task priorities. Sung *et al.* [3] proposed an approach to estimate task's actual execution time and slack time by using Kalman filter. Iwata *et al.* [4] proposed a method that reduces dynamic energy consumption by using task's actual execution time and slack time while guaranteeing the deadline constraint.

There are various researches exploring the stand-by energy consumption. Kathleen *et al.* [5] proposed that energy consumption greatly increase if system does not control energy appropri-

[1] Kyoto University, Sakyoku, Kyoto 606-8501, Japan
[a)] matsui@lab3.kuis.kyoto-u.ac.jp
[b)] takase@i.kyoto-u.ac.jp

ately when the CPU is idle. Muhammad *et al.* [6] proposed a method to find a power state transition using break-even time. Qian *et al.* [7] proposed to find a power state transition by using machine learning. Xin *et al.* [8] focused on the time and energy overhead caused by a power state transition and reduced the stand-by energy consumption while suppressing frequent transition.

3. Preliminary

3.1 System Model

We assume periodic task-model with I independent tasks $\tau = \tau_1, \tau_2, ..., \tau_I$. Each task τ_i is represented as a tuple $\tau_i = \langle T_i, D_i, C_i \rangle$, where T_i is the period and D_i is the deadline. C_i is the worst-case execution time when τ_i is executed at the highest frequency. For the sake of simplicity, we assume implicit deadline meaning $T_i = D_i$ for τ_i. Each independent task will release a sequence of unlimited jobs $j_i = \langle r_i, c_i, d_i \rangle$, where r_i, c_i, d_i are the absolute release time, actual execution time and absolute deadline respectively. For the sake of simplicity, we assume that the actual execution time of jobs of the same task is constant and equal to the worst execution time. The priority of each instance of a task is determined by its deadline. The highest priority is given to the task with the earliest deadline and we assume d_i always satisfies $d_i \leq d_{i+1} (i = 1, 2, ..., I-1)$. We defined the load $u_i = \frac{C_i}{T_i}$ of τ_i and defined the CPU utilization $U = \sum_{i=0}^{I} u_i$.

We assume that the processor core has J number of configurable frequencies and $K+1$ power states. Power modes consists of one operation state and K number of low power states. Let settable frequencies in the operation state ($k = 0$) be $f = f_1, f_2, ..., f_j, ..., f_J (f_1 < f_2 < ... < f_j < ... < f_J)$ and let the dynamic power of each frequency be $DP = DP_1, DP_2, ..., DP_j, ..., DP_J (DP_1 < DP_2 < ... < DP_j < ... < DP_J)$. We assume that the overhead associated with change of the operating frequency and supply voltage in the operation state can be neglected. Let the stand-by power of each low power state ($k \neq 0$) be $SP = SP_1, SP_2, ..., SP_k, ..., SP_K (SP_1 < SP_2 < ... < SP_k < ... < SP_K)$. We assume that time and energy consumption overhead are required for entering and leaving low power states, where time overhead is TO_k and energy overhead is EO_k. If there is no task to be executed and CPU is in the operation state, the system's power is DP_1. When a core frequency is f_j and its power state is k, total power of the processor core $P_{j,k}$ is given as formula (1).

$$P_{j,k} = \begin{cases} DP_j & (k = 0) \\ SP_k & (k \neq 0) \end{cases} \tag{1}$$

We discuss the break-even time [6]. The break-even time is the minimum value of the slack time that can be expected to reduce the total energy consumption due to the power state transition. We denoted the break-even time $BET_{j,k}$ by the time of transition from the operation state that operates at the frequency f_j to the low power state $k (\neq 0)$ and returning is expressed by the following formula (2).

$$BET_{j,k} = \max[\frac{EO_k - SP_k * TO_k}{RP_j - SP_k}, TO_k] \tag{2}$$

Algorithm 1 cal_laEDF_freq(t, τ)

Input: current time t, U, d_i, u_i, c_rem_i
Output: frequency calculated by laEDF f_{laEDF}

```
1: $U' \Leftarrow U$
2: $s \Leftarrow 0$
3: for $i = I$ to 1 do
4:     $U' \Leftarrow U' - u_i$
5:     $x \Leftarrow \max\{0, c\_rem_i - (1 - U)(d_i - d_1)\}$
6:     $U' \Leftarrow U' + (c\_rem_i - x)/(d_i - d_1)$
7:     $s \Leftarrow s + x$
8: end for
9: $f_{laEDF} \Leftarrow f_J * s/(d_1 - t)$ ;
```

3.2 look ahead EDF（laEDF）

laEDF [9] is a common DVFS algorithm that guarantees system's deadline constraint and achieve high energy savings. Algorithm1 shows how laEDF algorithm works. In laEDF, we keep track of the worst-case remaining computation c_rem_i for the current invocation of task τ_i. This is set to C_i on task release, decremented as the task executes, and set to 0 on completion. The major step in this algorithm is the deferral function.

Here, we look at the interval until the next task deadline, try to push as much work as we can beyond the deadline, and compute the minimum number of cycles, s, that we must execute during this interval in order to meet all future deadlines. The operating frequency is set just fast enough to execute s cycles over this interval. To calculate s, we look at the tasks in reverse EDF order (i.e., latest deadline first). Assuming worst-case utilization by tasks with earlier deadlines (effectively reserving time for their future invocations), we calculate the minimum number of cycles, x, that the task must execute before the closest deadline, d_1, in order for it to complete by its own deadline. A cumulative utilization U is adjusted to reflect the actual utilization of the task for the time after d_1. This calculation is repeated for all of the tasks, using assumed worst-case utilization values for earlier-deadline tasks and the computed values for the later-deadline ones. s is simply the sum of the x values calculated for all of the tasks, and therefore reflects the total number of cycles that must execute by d_1 in order for all tasks to meet their deadlines. Although this algorithm very aggressively reduces processor frequency and voltage, it ensures that there are sufficient cycles available for each task to meet its deadline after reserving worst-case requirements for higher-priority (earlier deadline) tasks.

In some cases, laEDF pessimistically estimates the amount of time that a task can be executed. (explain in section 4). As a result, it may negatively effect DVFS and DPM because they use the task's slack time to reduce enegy consumption.

4. Proposed Method

In this section, we introduce an energy-efficient real-time task scheduling algorithm for a singlecore platform that supports multiple power states. Our proposed method consists of Slack Gathering laEDF (SGlaEDF) and Core State Aware Scheduling (CSAS). We first introduce how SGlaEDF calculates task's slack time accurately and guarantee system's deadline constraint. Then we present how CSAS decides the operation frequency and power state transition. Finally, we present an efficient method to achieve reduction of energy consumption while guaranteeing system's deadline constraint.

4.1 Slack Gathering laEDF (SGlaEDF)

We first developed a novel DVFS method laEDF to accurately calculate task's slack time while guaranteeing system's deadline constraint.

The reason for estimating the slack time pessimistically in laEDF is that a task's executable time (consisits of the task's execution time and slack time) may exceed the interval from the current time to the latest deadline in some cases. This occurs when the closest deadline is the deadline of a task that has already been completed. For example, at the current time t, we assume τ_1(i.e., closest deadline task) is already completed task and $\tau_i (\neq \tau_1)$ is selected as the task to be executed and been executed until the latest deadline d_1. When τ_1 is released at time $t'(= d_1)$, if τ_i becomes the latest deadline, τ_i will be executed again. As a result, the executable time of τ_i is devided into two or more parts. This problem does not occur when τ_1 is not completed because the closest deadline's τ_1 is selected as the task to be executed. Therefore, due to the deadline constraint, the task's executable time does not exceed the latest deadline.

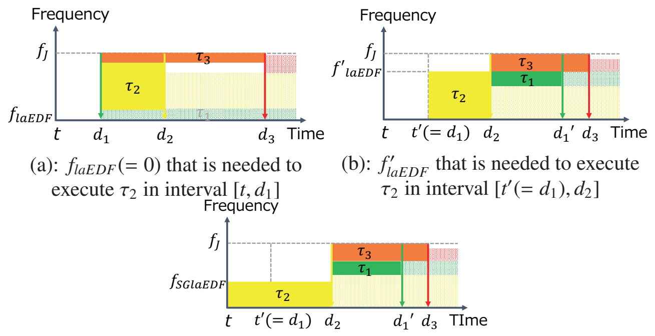
By using the nature of the periodic task, SGlaEDF updates the task's next absolute deadline and worst-case remaining computation when task execution is completed. As a result, it guarantee that the interval until the next task deadline always contain any task's executable time. Therefore, in SGlaEDF, d_i is defined as follows:

$$d_i = \begin{cases} d_i + P_i & (c\_rem = 0) \\ d_i & (c\_rem \neq 0) \end{cases} \qquad (3)$$

According to (3), the latest deadline d_1 at a current time t is the task's deadline that has not been completed including tasks that have not been released at the time t. From the deadline constraint, τ_1 must be started to execute from $d_1 - c\_rem_1$ at the latest time. The maximum executable time of τ_i is $[t, d_1 - c\_rem_1]$, that is included in the interval between the current time and the latest deadline $[t, d - 1]$. Therefore, the executable time of τ_i is always included in the interval from the current time to the latest deadline, that is, our SGlaEDF always guarantees the deadline constraint.

Similarly to laEDF, we look at the interval until the next task deadline, try to push as much work as we can beyond the deadline. In this way, we can calculate the minimum operating frequency in the executable time of the task to be executed and guarantee the deadline constraint.

Fig.1 is an example that SGlaEDF can reduce energy consumption. At time t, τ_2 is the task to be executed, and τ_1 is the task that has already been completed. Fig.1(a) and Fig.1(b) are executed with laEDF and Fig.1(c) is executed with SGlaEDF. In Fig.1(a), at the time of t, laEDF calculated the frequency $f_{laEDF}(= 0)$ and the interval $[t, d_1]$ is the slack time. In Fig.1(b), at the time of $t'(= d_1)$,τ_1 is released and task information is updated. Then, τ_2 becomes the task to be executed again, and it is executed with the significantly increased frequency f'_{laEDF} in the interval $[t', d_2]$. As a result, laEDF divides τ_2's executable time into $[t, d_1]$ and $[d_1, d_2]$. On the other hand, in Fig.1(c), SGlaEDF can include



(a): $f_{laEDF}(= 0)$ that is needed to execute τ_2 in interval $[t, d_1]$

(b): f'_{laEDF} that is needed to execute τ_2 in interval $[t'(= d_1), d_2]$

(c): f'_{laEDF} that is needed to execute τ_2 in interval $[t, d_2]$

Fig. 1 An example that SGlaEDF can include the overall executable time of a task

τ_2's overall executable time.

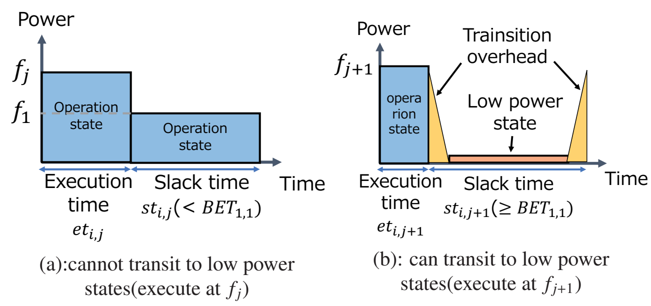
4.2 Core State Aware Scheduling (CSAS)

In this section, we introduce a scheduling method that decides the operating frequency and power state transition so as to reduce total energy consumption of the system. In this method, we compare the increased amount of dynamic power consumption and the reduced amount of stand-by power consumption when we raise the operating frequency. This determines the operating frequency and power state transition to minimize the total energy cumsumption.

To analysis the tradeoff between DVFS and DPM, we use the break-even time $BET_{j.k}$ (explained in section 3) and the task slack time $st_{i,j}$. If $st_{i,j} > BET_{1,k}$, total energy consumption can be reduced by transitioning into low power state k. Furthermore, CSAS compares the dynamic energy consumption increase effect and stand-by energy consumption reduction effect by increasing the operating frequency, and determines the appropriate operating frequency and power state transition.

Fig.2 is an example that CSAS can reduce total energy consumption. For simplicity of explanation, We assume that the processor core has two core states of one operation state and one idle state. We denote $et_{i,j}$ and $st_{i,j}(< BET_{1,1})$ by the execution time and slack time respectively when τ_i is executed with the frequency f_j. $et_{i,j+1}$, and $st_{i,j+1}(\geq BET_{1,1})$ are the execution time and slack time respectively when τ_i is executed with the frequency f_{j+1}, one higher than f_j. Fig.2(a) and Fig.2(b) show the energy comsumption at each operating frequency. In Fig.2(a), system is still active state in the slack time because $st_{i,j} < BET_{1,1}$. In contrast, in Fig.2(b), system can transit to low power states in the slack time because $st_{i,j+1} \geq BET_{1,1}$. Although the dynamic energy consumption in the execution of τ_i increases, the stand-by energy consumption decreases. Therefore, if the amount of stand-by energy consumption reduction exceeds the amount of dynamic energy consumption increase, the total energy consumption can be reduced.

Algorithm 2 shows the algorithm of CSAS. Lines 1–2, at time t, we denote the earliest release time $r_{highpri}$ by the release time of task among the task group with a higher priority than τ_i, and the earliest release time r_{lowpri} by the release time of task among the task group with a lower priority than the task τ_i. Lines 5–6, $et_{i,j}$ and $st_{i,j}$ are calculated from the worst remaining execution time c_rem_i, f_j, $r_{highpri}$ and $r_{highpri}$. Lines 7–15, We call $energy_{i,j,k}$ by



Power
f_j
f_1
Operation state
Operation state
Execution time $et_{i,j}$
Slack time $st_{i,j}(< BET_{1,1})$
Time
(a):cannot transit to low power states(execute at f_j)
Power
f_{j+1}
Trainsition overhead
opera rion state
Low power state
Execution time $et_{i,j+1}$
Slack time $st_{i,j+1}(\geq BET_{1,1})$
Time
(b): can transit to low power states(execute at f_{j+1})

Fig. 2 An example that increase frequency can reduce total energy consumption.

Algorithm 2 cal_CSAS_freq(f, t, τ)

```
Input: current time $t$, $r_i$, $d_i$, $c\_rem_i$, $f_a$
Output: frequency $f_{CSAS}$, power state transition$k_{CSAS}$
1:  $r_{highpri} \Leftarrow \{\min(r_1, r_2, ..., r_{i-1}) | d_1 \leq d_2 \leq ... \leq d_{i-1}\}$
2:  $r_{lowpri} \Leftarrow \{\min(r_{i+1}, r_{i+2}, ..., r_I) | d_{i+1} \leq d_{i+2} \leq ... \leq d_I\}$
3:  for $k = 1$ to $K$ do
4:      for $j = a$ to $J$ do
5:          $et_{i,j} \Leftarrow \min(c\_rem_i / f_j, r_{highpri} - t)$
6:          $st_{i,j} \Leftarrow \max(0, min(r_{highpri}, r_{lowpri}, d_i) - t - c\_rem_i / f_j)$
7:          if $st_{i,j} \geq BET_{1,k}$ then
8:              $energy_{i,j,k} \Leftarrow et_{i,j} * DP_j + (st_{i,j} - TO_k) * SP_k + EO_k$
9:              $state_{i,j,k} \Leftarrow k$
10:         else
11:             $energy_{i,j,k} \Leftarrow et_{i,j} * DP_j + st_{i,j} * DP_1$
12:             $state_{i,j,k} \Leftarrow 0$
13:         end if
14:     end for
15: end for
16: $energy_{i',j',k'} \Leftarrow \min(energy_{i,j,k})$
17: $f_{CSAS} = f_{j'}$
18: $k_{CSAS} = state_{i',j',k'}$
```

the energy consumption of the executable time of τ_i. $energy_{i,j,k}$ is calculated in the combination of f_j and the transition power state $state_{i,j,k}$. By using $st_{i,j,k}$, we decide whether or not to transition to low power states k using break-even time $BET_{1,k}$. Perform the above operations for all core states and combinations of configurable frequencies above f_a, we calculate the frequency f_{CSAS} and the transition k_{CSAS} as a solution so that $energy_{i,j,k}$ is the lowest.

4.3 Proposed Scheduler Procedure

Algorithm 3 shows the procedure of the proposed task scheduler. Lines 1–8 denote procedure when the system operation starts, lines 9–12 and lines 13–16 denote procedure when the task is released or dispatched, lines 17–20 denote procedure when the task is completed. For procedures other than when the task is completed, first SGlaEDF calculates the minimum frequency $f_{SGlaEDF}$ that guarantees the deadline constraint. Next, CSAS determines the operating frequency f_{CSAS} in task execution time and power state transition k_{CSAS} in slack time. Deadline constraints are guaranteed by limiting CSAS's configurable minimum frequency f_a to a frequency above $f_{SGlaEDF}$. Also, at the start of system operation, the deadline of all tasks and the worst remaining execution time are updated first. In the proposed method, since the optimum frequency and power state transition in the executable time of the task are simultaneously calculated, it is not necessary to call it when the task is completed. However, for SGlaEDF, update the task's deadline and worst remaining execution time.

Algorithm 3 Scheduling SGlaEDF + CSAS

```
1:  procedure THE SYSTEM IS INITIALIZED
2:      for $i = I$ to 1 do
3:          $c\_rem_i \Leftarrow C_i$
4:          $d_i \Leftarrow T_i$
5:      end for
6:      $f_{SGlaEDF}$ = cal_SGlaEDF_freq($0, \tau$)
7:      $f_{CSAS}, k_{CSAS}$ = cal_CSAS_freq($f_{SGlaEDF}, 0, \tau$)
8:  end procedure
9:  procedure ONE OR MORE TASKS ARE RELEASED AT $t$
10:     $f_{SGlaEDF}$ = cal_SGlaEDF_freq($t, \tau$)
11:     $f_{CSAS}, k_{CSAS}$ = cal_CSAS_freq($f_{SGlaEDF}, t, \tau$)
12: end procedure
13: procedure A TASK IS DISPATCHED AT $t$
14:     $f_{SGlaEDF}$ = cal_SGlaEDF_freq($t, \tau$)
15:     $f_{CSAS}, k_{CSAS}$ = cal_CSAS_freq($f_{SGlaEDF}, t, \tau$)
16: end procedure
17: procedure THE EXECUTION OF TASK $\tau_i$ IS COMPLETED
18:     $d_i \Leftarrow d_i + T_i$
19:     $c\_rem_i \Leftarrow C_i$
20: end procedure
```

5. Evaluation

5.1 Setup

We evaluated the effectiveness of the proposed method on the developed task scheduling simulator. The task set used a random taskset that was generated based on the input parameters. The period of each task is randomly selected from 1, 5, 10, 20, and 50 ms. The worst-case execution time of the task was determined using the taskset's load U and the normal distribution. The processor core to be evaluated was Cortex-A15 [10] core installed in ODROID-XU3 [11]. In this processor core, the frequency can be set in increments of 100 MHz from 200 MHz to 2000 MHz. The frequency in the operation state of the Cortex-A15 core and the corresponding value of power consumption are given in [4]. The measured power consumption when only one core of Cortex-A15 core on ODROID-XU3 was operated with each frequency set value was used. In order to simplify the evaluation experiments, we assumed that the number of power states of the processor was set as two, where one operation state and one low power state. The stand-by power consumption SP_k in the power state $k(\neq 0)$ is set by the following formula (4).

$$SP_k = \frac{DP_1}{10} \tag{4}$$

We set 600μs and $5 * SP_k$ for TO_k and EO_k, that are time and energy overhead during power state transition. We assumed that the time and energy overhead involved in running the algorithm can be ignored.

There are four scheduling methods to be evaluated.

(1) laEDF
(2) SGlaEDF
(3) laEDF+CSAS
(4) SGlaEDF+CSAS（Proposed）

In this experiment, the load U and the number of tasks I were used as parameters for automatic generation of a taskset. U ranged from 0.1 to 0.9 in 0.2 increments and $I = 10, 30, 60, 90$, for each U, totaling 20 combinations were tested. In the exper-

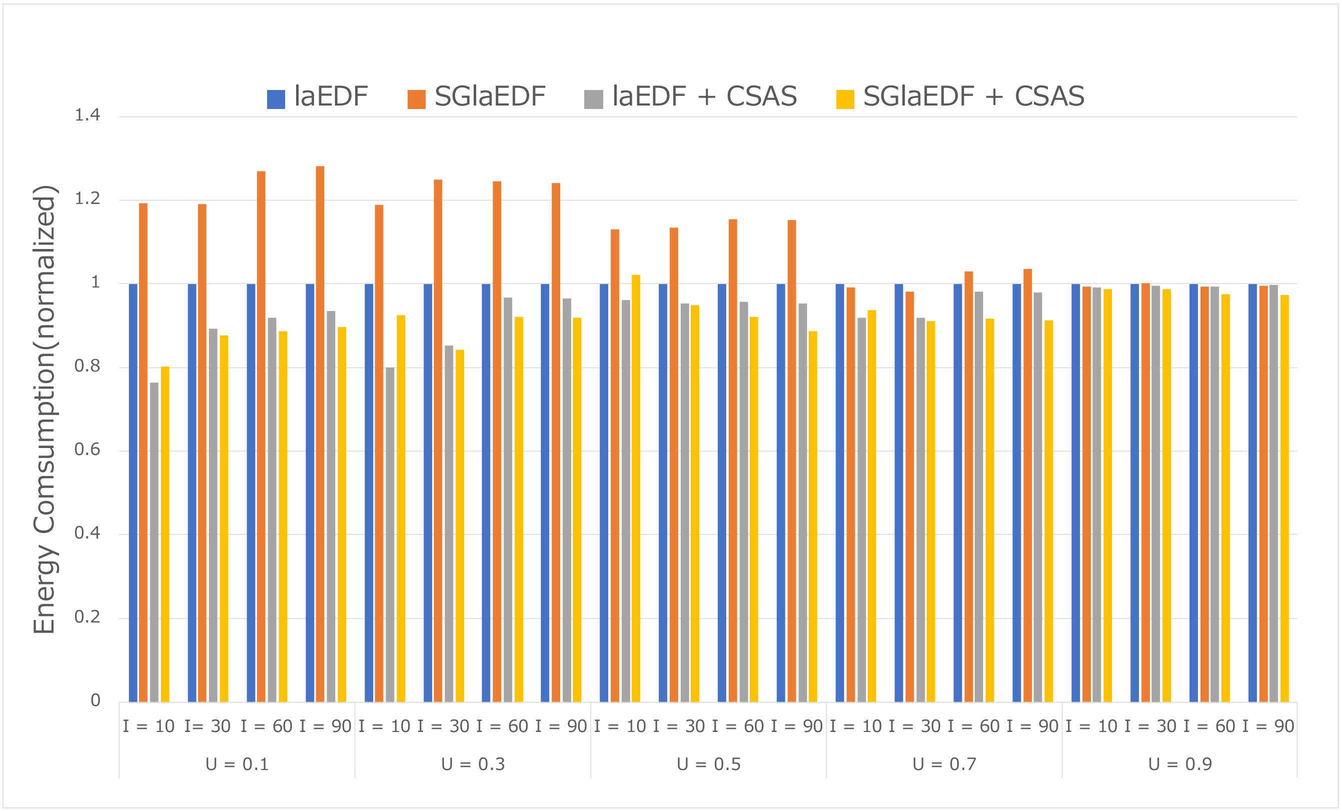


Fig. 3 Normalized energy consumption with CPU utilization 0.1, 0.3, 0.5, 0.7, 0.9, and the number of tasks 10, 30, 60, 90

iment, in order to compare the effectiveness of laEDF and other task scheduling, energy consumption of other methods was normalized by energy consumption of laEDF.

5.2 Results

Fig.3 shows the evaluation result of each task set in the hyperperiod. The ordinate shows energy consumption normalized with respect to laEDF, and the abscissa shows the number of tasks. As a result of the evaluation, the proposed method had highest energy consumption reduction effect when $I = 30 \sim 90$, it can reduce energy consumption by 20% at maximum and 8% on average compared with laEDF.

5.3 Discussion

When applying SGlaEDF alone, the effect of reducing energy consumption was only recognized for $U = 0.7, 0.9$. and when U is low, SGlaEDF most consumed energy. It is considered that updating the next deadline and remaining execution time when a periodic task execution may cause an apparent task load and especially it has negative effects to increase dynamic energy when the system has many slacks.

The proposed method SGlaEDF + CSAS has higher energy saving effect than laEDF + CSAS when $I = 30 \sim 90$. Especially when the number of tasks is large, the advantage of the proposed method got higher. This is because in the case of laEDF + CSAS, when the number of tasks is large, it is considered that the number of times the executable time exceeds the latest deadline increases. On the other hand, when the number of tasks is small, few number of times the executable time exceeds the latest deadline and decrease energy saving effect.

6. Conclusion

In recent embedded real-time systems, it is important to reduce energy consumption and achieve high performance. This paper studies the problem on how to determine an energy management policy for a singlecore platform that supports low power states. To solve this problem, we first develop a novel DVFS method to accurately calculate task's slack time while guaranteeing system's deadline constraint. We then present an approach to decide the operation frequency and power state transition that reduce energy consumption. Finally, we develop a scheduling method to reduce energy consumption and guarantee the deadline constraint of a real-time system.

The future task is to consider the verification on an actual system and the overhead of execution of an algorithm.

Acknowledgments This work was supported by JSPS KAKENHI Grant Number JP26870303.

References

[1] Hayashi, K. and Namiki, M.: A energy saving method in EDF scheduling, *IPSJ (OS)*, Vol. 2010-OS-113, No. 15, pp. 1–8 (2010).

[2] Kim, W. et al.: Dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis, *Proc. of DATE*, p. 788 (2002).

[3] Bang, S. Y. et al.: Run-Time Adaptive Workload Estimation for Dynamic Voltage Scaling, *IEEE Trans. on CAD*, Vol. 28, No. 9, pp. 1334–1347 (2009).

[4] Iwata, A. et al.: Task Scheduling for Reducing Energy Consumption on Single-ISA Heterogeneous Multi-core Systems, *IPSJ*, Vol. 2015-EMB-36, No. 33, pp. 1–6 (2015).

[5] Baynes, K. et al.: The performance and energy consumption of em-

bedded real-time operating systems, *IEEE Trans. on Comp*, Vol. 52, No. 11, pp. 1454–1469 (2003).

[6] Awan, M. A. and Petters, S. M.: Energy-aware partitioning of tasks onto a heterogeneous multi-core platform, *Proc. of RTCSA*, pp. 205–214 (2013).

[7] Diao, Q. and Song, J.: Prediction of CPU idle-busy activity pattern, *Proc. of HPCA*, pp. 27–36 (2008).

[8] Zhan, X. et al.: CARB: A C-State Power Management Arbiter For Latency-Critical Workloads, *IEEE CAL*, No. 99 (2016).

[9] Pillai, P. and Shin, K. G.: Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, *ACM SIGOPS Oper. Syst. Rev.*, pp. 89–102 (2001).

[10] ARM: Cortex-A15, `https://www.arm.com/ja/products/processors/cortex-a/cortex-a15.php`.

[11] Hardkernel: ODROID-XU3, `http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127`.

[12] Liu, C. L. and Layland, J. W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of ACM*, Vol. 20, No. 1, pp. 46–61 (1973).

[13] Lu, Y. H. and De Micheli, G.: Comparing System-Level Power Management Policies, *IEEE Design & Test of Computers*, Vol. 18, No. 2, pp. 10–19 (2001).

[14] : ACPI Advanced Configuration & Power Interface, `http://www.acpi.info/`.