PAPER

An Energy-Efficient Task Scheduling for Near Real-Time Systems on Heterogeneous Multicore Processors

Takashi NAKADA[†a)], ***Member***, **Hiroyuki YANAGIHASHI**[††], **Kunimaro IMAI**[†††], **Hiroshi UEKI**[††††], **Takashi TSUCHIYA**[††††], **Masanori HAYASHIKOSHI**[††††], ***Nonmembers***, ***and*** **Hiroshi NAKAMURA**[††], ***Senior Member***

SUMMARY Near real-time periodic tasks, which are popular in multimedia streaming applications, have deadline periods that are longer than the input intervals thanks to buffering. For such applications, the conventional frame-based schedulings cannot realize the optimal scheduling due to their shortsighted deadline assumptions. To realize globally energy-efficient executions of these applications, we propose a novel task scheduling algorithm, which takes advantage of the long deadline period. We confirm our approach can take advantage of the longer deadline period and reduce the average power consumption by up to 18%.

key words: *adaptive task scheduling, near real-time processing, energy efficiency, heterogeneous multicore processors*

1. Introduction

In most embedded systems, jobs arrive periodically. *Input interval length* is defined as the distance between arrival times of successive jobs. In a recent evolving information society, processors are required to execute a wide variety of *tasks* with different input interval and execution time. For instance, an IoT sensor, which monitors several kinds of sensors such as temperature, vibration, and image, executes the same number of tasks as that of the sensors. Each task is invoked suitable interval of the corresponding sensor. An example is shown in Fig. 1. Each task consists of periodic jobs.

Meanwhile, minimizing the energy consumption of embedded systems is a very critical concern. To adapt to such a situation, heterogeneous multicore and DVFS (Dynamic Voltage and Frequency Scaling) [1]–[3] can be very effective. In Fig. 1, there are two processors; MCUhigh, which is a high-performance MCU (Micro Controller Unit) and MCUlow, which is an energy efficient MCU. In this figure, the y-axis shows the relative power consumption. However, ASAP (As Soon As Possible) scheduling, which is the most straightforward algorithm, only uses MCUhigh and cannot take advantage of heterogeneous multicores (Fig. 1 (a)).

Manuscript received April 5, 2019.
Manuscript revised August 8, 2019.
Manuscript publicized November 1, 2019.
†The author is with Nara Institutet of Science and Technology, Ikoma-shi, 630–0192 Japan.
††The authors are with the University of Tokyo, Tokyo, 113–8656 Japan.
†††The author is with Proassist Ltd, Osaka-shi, 540–0031 Japan.
††††The authors are with Renesas Electronics Corporation, Kodaira-shi, 187–8588 Japan.
a) E-mail: nakada@is.naist.jp
DOI: 10.1587/transinf.2019EDP7101

Therefore, adaptive task scheduling, which includes execution timing management and adaptive active core selection, is indispensable for low power embedded systems. To cope with this challenge, several energy efficient algorithms have been proposed [4], [5]. However, most of them assume that the deadline period is the same as the input interval and schedule jobs within only one *hyper period*. The hyper period is defined as the least common multiple (L.C.M.) of the input intervals. As a result, they can independently minimize the energy consumption only in each hyper period and realize energy-efficient scheduling within each interval as shown in Fig. 1 (b). Therefore, this scheduling cannot take advantage of the deadline period that is longer than the input interval.

To solve this problem a continuous task scheduling algorithm that can take advantage of the longer deadline period has been proposed [6] as shown in Fig. 1 (c). In this execution, two MCUs are used exclusively to improve energy efficiency during execution and the number of core switching is reduced to minimize the energy overhead.

Our approach here is to propose an energy-aware task scheduling algorithm that can take advantage of the longer deadline period as shown in Fig. 1 (d) [7]. In this execution, (A) the usage of energy efficient core (MCUlow) is maxi-

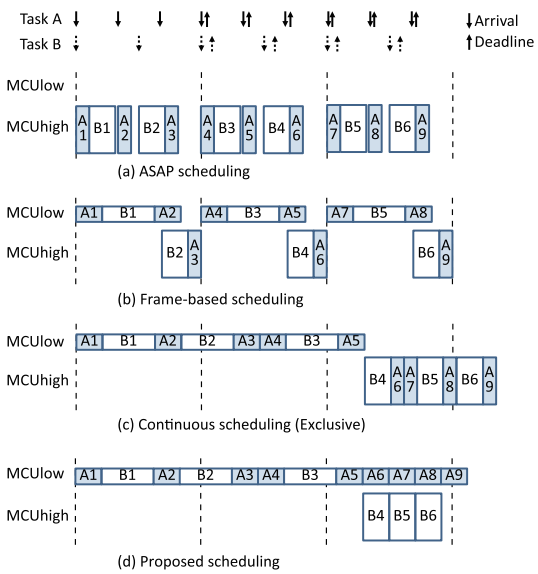


Fig. 1 Scheduling for periodic tasks

mized, namely MCUlow is always on to maximize energy efficiency during execution and (B) the energy overhead is also minimized.

To realize energy-efficient task scheduling, the primary contributions of this paper are as follows.

- We propose an energy-aware scheduling that takes advantage of the deadline period that is longer than the input interval.
- We try to maximize the usage of energy efficient core and achieve the highest energy efficiency.
- We try to minimize the number of core switching and achieve significant energy reduction.

The remaining parts of this paper are organized as follows. Sections 2 and 3 introduce background and related work respectively. Section 4 presents the target problem and proposed task scheduling. Experimental results appear in Sect. 5. Finally, Sect. 6 concludes this paper.

2. Background

2.1 Embedded Systems

In this paper, we assume that a system that has heterogeneous multicores and any core can be used at any time. We also assume a set of near real-time tasks as shown in Fig. 1. In each task, jobs are arrived periodically, nonpreemptive and their sizes are known and fixed. The execution times can be calculated by the job sizes and the processor performances. The jobs are independent of each other. Within a task, the jobs are invoked FCFS (First Come First Serve) policy, but they may finish different order due to performance differences of the assigned cores. Their deadline periods of near real-time tasks are longer than their input intervals.

2.2 Basics of Energy Model

In general, the relation between energy, voltage and clock frequency can be modeled by following known equation [8].

$$E_{proc} = \alpha_1 T_1 C V^2 f + T_2 V I_{leak}. \quad (1)$$

Here, E_{proc} represents the energy consumption of the microprocessor. α_1, T_1, C, V and f represent a constant value, the execution time, the circuit capacity, the supply voltage, the operating frequency respectively. T_2 and I_{leak} represent the total time that includes idle period and the leakage current respectively.

The first and the second terms represent the dynamic and the static energy respectively. The former is caused by switching activities of transistors and essentially consumed by computing. On the other hand, the latter is caused by leakage current and always consumed whenever power is supplied.

The higher performance is realized by higher voltage, frequency and the larger circuit that causes the larger circuit capacity.

In general, more powerful processor core consumes more energy. There exists an empirical model between them called Pollack's Rule [9]. The performance is roughly proportional to the square root of a processor's area. The static power is proportional to the area while the dynamic power is more complex and it can be regarded as being roughly proportional to the performance since switching rates differ between functional units (FUs) and other parts (in general, they switch less frequently than FUs). Therefore, using smaller core is better from the viewpoint of energy efficiency.

3. Related Work

In this section, we introduce existing energy efficient task scheduling algorithms.

Frame-based scheduling is widely studied [4], [5], [10]–[13]. Their target applications consist of multiple tasks. However, the main drawback of these frame-based optimization algorithms is they often assume that the deadline period is the same as the input interval even if the deadline periods are longer than input intervals. In other words, they always complete any job before the next job that belongs the same task arrives. Even though some of them can manage the longer deadline directory, they consider energy-efficient scheduling within a hyper period only and same scheduling is applied repeatedly. Namely, all of the jobs must be finished before the boundary of the hyper periods. As a result, the scheduling is optimized within a hyper period and the improvement of the energy efficiency is limited.

For example, there are five jobs in a hyper period in Fig. 1 (b). These five jobs are optimized in a hyper period. However, due to scheduling flexibility is limited in the hyper period, both cores are powered on and off very frequently. Scheduling multiple hyper periods at the same time can be a solution to this limitation. However, scheduling cost grows exponentially with the larger scheduling flexibility. To alleviate this issue, a heuristics approach is proposed [14].

Another approach that can take advantage of the longer deadline period has been proposed [6]. This scheduling can schedule jobs across multiple hyper periods and active core performance is chosen dynamically based on a slack time, which is defined as the difference between the current time and the deadline of the next job. This approach realized better energy efficiency than the frame-based schedulings. However, they assume only one core is active at the same time such as DVFS processors. Due to this limitation, this approach cannot take advantage of heterogeneous multiprocessors.

Another approach is a prediction based scheduling [15]. The effectiveness of this approach relies on the accuracy of its execution time predictor. Additionally, such run-time prediction must need additional computation cost and a waste of energy.

4. Energy-Aware Task Scheduling

4.1 Problem Definition

Firstly, we introduce input variables, which are related to hardware and software, as shown in Table 1. These variables are given or are easily computed from other given parameters.

A task consists of periodical and homogeneous jobs. N represents the number of tasks. The tasks have IDs $j(j = A, B, C, \dots)$ and characterized by an Input interval $I(j)$, a deadline period $d(j)$ and a size $W(j)$. $l(j)$ represents a load factor of task j ($l(j) = W(j)/I(j)$). We assume the higher load factor task has the later ID j.

The cores also have IDs c ($c = c_1, c_2, c_3, \dots, c_M$). For heterogeneous multicore systems, each ID corresponds to a logical core. On the other hand, for a processor that has DVFS technology, each ID corresponds to each performance mode. M represents the number of available cores and $p(c)$ represents the performance of core c. We assume the higher performance core has a larger ID. $et(j, c)$, $P(c)$, $P_s(c)$ represent the execution time of task j on core c and the dynamic and static power of core c respectively. Here, $et(j, c)$ is defined as follows.

$$et(j, c) = \frac{W(j)}{p(c)}. \tag{2}$$

$E_{OV}(c)$ and $T_{OV}(c)$ represent total energy and latency overhead of power on and off transition on core c. This overhead can contain cold start overhead of assigned tasks.

We assume that the higher performance core executes any jobs faster with larger power consumption. Therefore, the following inequalities are satisfied.

$$\forall j, For\ p_x < p_y :$$
$$et(j, c_x) > et(j, c_y) \tag{3}$$
$$P(c_x) < P(c_y) \tag{4}$$
$$P_s(c_x) < P_s(c_y) \tag{5}$$

Additionally, we also introduce a hyper period HP, which is defined as L.C.M. of $I(j)$. In an HP, there are L jobs. L is given as follows.

$$L = \sum_j \frac{HP}{I(j)}. \tag{6}$$

We also define L_c as the number of jobs on core c in an HP.

Our ultimate goal is to minimize the energy consumption under performance constraint. As a result, objective function and the constraint condition are as follows. In the following sections, we solve this optimization problem.

$$min\ (Average\ energy\ consumption\ of\ cores\ per\ input)$$
$$s.t.\qquad (Satisfy\ deadline\ constraints)$$

Table 1 Input variables

Variables	Definition
N	Number of tasks
$I(j)$	Input Interval of task j
$d(j)$	Deadline period of task j
$W(j)$	Size of task j
$l(j)$	Utilization of task j ($l(j) = W(j)/I(j)$)
M	Number of cores
$p(c)$	Performance of core c
$et(j, c)$	Execution time of task j on core c
$P(c)$	Dynamic power of core c
$P_s(c)$	Static power of core c
$E_{OV}(c)$	Overhead energy of power state transitions of core c
$T_{OV}(c)$	Overhead latency of power state transitions of core c
HP	Hyper period
L	Number of jobs in HP

4.2 Core Selection

When there are M available cores and their performance is different from each other, the number of possible core selection is $2^M - 1$. If some cores have DVFS capability, each frequency mode is regarded as independent core and some physically impossible combinations are removed from $2^M - 1$ candidates beforehand. Here we define a set of selected cores as $C'_{m'}$ and its total throughput $tp(C'_{m'})$ is also defined as follows.

$$tp(C'_{m'}) = \sum_{c \in C'_{m'}} p(c). \tag{7}$$

We assume the higher performance set of cores has the larger ID.

To minimize energy consumption, some of them are obviously removed from the candidates. Namely, a set that has lower performance but consumes more energy than other sets should be removed. To find valuable sets, we introduce power efficiency pe. The power efficiency of set $C'_{m'}$ is defined as follows.

$$pe_{m'} = \frac{\sum_{c \in C'_{m'}} (P(c) + P_s(c))}{tp(C'_{m'})} \tag{8}$$

When x-axis and y-axis are throughput (tp) and power consumption (P) respectively as shown in Fig. 2.

From these sets, the first valuable set is the most power efficient set, namely line that has the smallest gradient. In

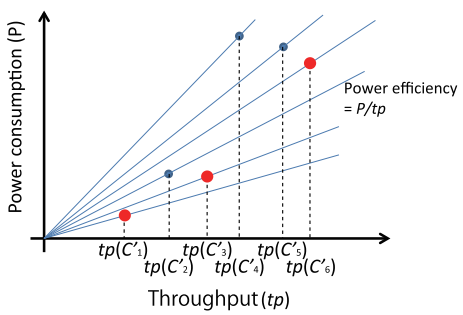


Fig. 2 Power efficiency

this example, C'_1 is chosen. If there are lower performance sets than that, they are marked as useless sets. Then the same procedure is applied for remaining sets until all sets are classified. Finally, there are three valuable sets (C'_1, C'_3, C'_6), which are marked with red color, in this example. These valuable sets are defined as C_m ($m = 1, 2, \dots$). C_m is obviously Pareto optimal. Namely, any performance between $tp(C'_1)$ and $tp(C'_6)$ can be realized by a combination of C'_1, C'_3 and C'_6 with the minimum power consumption.

Utilization on core set C_m is defined as follows.

$$U(C_m) = \sum_j \frac{l(j)}{tp(C_m)} = \sum_j \frac{W(j)}{I(j) \cdot tp(C_m)} \tag{9}$$

If $U(C_m) > 1$, such core set C_m cannot execute all tasks. Therefore, a core set that has $U(C_m)$ less than or equal to 1 is required. If $U(C_m) = 1$, such core set can execute all tasks continuously and core set C_m can be the optimal core set. In this case, further discussion is not required. Hereafter, we focus on other cases.

To minimize the average energy consumption, dynamic core selection is the key. First, we should choose a core set C_{low} that have the smallest $U(C_{low})$ greater than 1 and a core C_{high} that have the largest $U(C_{high})$ less than 1. Then, $(1-U(C_{high}))/(U(C_{low})-U(C_{high}))$ of total tasks are executed on set C_{low} and the remaining tasks are executed on set C_{high}. The execution will continue endlessly on either core set.

If total task size is too small and C_{low} is not found, heterogeneous multicores are not required and intermittent scheduling algorithms such as [16] should be considered.

4.3 Task Assignment

In this section, we introduce how to assign tasks to each core in a core set.

Now, there are M' cores in a core set C_x, and these cores are defined as $c^x_m (m = 1, 2, \dots, M')$ and utilization of each core is defined as $u(c)$. To minimize core set switching, the utilization of each core should be as same as possible. To realize such a task assignment, we introduce the following algorithm.

When core utilization is ideally balanced, the ratio of assigned job size is same as that of core performance $p(c_m)$. Therefore, the ideal utilization of core c_m is defined as follows.

$$u_{ideal}(c_m) = \frac{p(c_m) \sum_j l(j)}{tp(C_x)}. \tag{10}$$

First, the smallest task is assigned to the lowest performance core (c_1). Then, if its utilization does not exceed $u_{ideal}(c_1)$, the next smallest task is assigned to c_1. When the utilization exceeds $u_{ideal}(c_1)$, there are two possibilities, whether the last assigned task is assigned to the current core (c_1) or the next core (c_2). To find the best assignment, we search for both possibilities concurrently. As a result, we can get $2^{M'-1}$ combinations.

Finally, we calculate sum of the difference from ideal utilization and the best assignment, which minimizes the error, as follows.

$$\min \sum_m |u_{ideal}(c_m) - u(c_m)|. \tag{11}$$

In the above discussion, we assume every job that belongs to the same task is assigned to the same core, in other words, this is per-task basis assignment. To realize more flexible assignment, per-job basis assignment is possible. In this case, the jobs in a hyper period should be considered independently at the same time. In the rest of this paper, to simplify the discussion, we use per-task assignment.

Additionally, the utilization of every core in C_{high} must satisfy the following condition.

$$\forall c \in C_{high}, u(c) \leq 1. \tag{12}$$

Otherwise, deadline violation will happen. If this condition is not satisfied, we should choose a higher performance core set as C_{high}. If there does not exist such core, there is no solution.

Meanwhile, the execution time of any task on a core in set C_{low} and C_{high} must shorter than the deadline period $d(j)$. This condition is given as follows.

$$\forall j, \forall c \in (C_{high} \cup C_{low}), et(j, c) < d(j). \tag{13}$$

4.4 Job Scheduling

We adopt a lumped execution [17], which executes multiple jobs continuously. To realize this execution, a job is not executed immediately after it arrives but postponed until several jobs arrive. Then, these ready jobs are executed continuously on the core set C_{low}. In case the deadline violation is predicted during execution, the working core set is changed to the core set C_{high}. After a while, in case that the lumped execution is not possible on C_{high} due to lack of ready jobs, then the core set is switched to C_{low}.

In the end, it comes down to a problem that when the execution starts and how many jobs are executed on each core set C_{high} and C_{low}.

4.4.1 Scheduling Algorithm

The proposed task scheduling is based on the lumped execution. In the following part, to apply lumped execution, we assume $d(j) \geq HP$. As long as this condition is satisfied, C_{high} can guarantee to meet deadline constraint. Otherwise higher performance core set than C_{high} may be required. The detail of the scheduling when $d(j) < HP$ will be discussed in Sect. 4.4.3.

Here, we explain the details of the proposed task scheduling algorithm. The initial state is S_0, which indicates that all of the processors are off. The other states are S_{high} and S_{low}. They correspond to C_{high} and C_{low} are active respectively.

4.4.2 Obtaining Energy-Efficient Scheduling

In this section, we explain how to obtain energy-efficient scheduling. Every job has been assigned to suitable core on each core set in the previous section. In each core, jobs that assigned the same core will be executed FCFS policy.

As an example, we assume two tasks (denoted as A and B) are executed on two heterogeneous cores (denoted as MCU_{high} and MCU_{low}). The input interval of task A is 2/3 of that of task B. As a result, *HP* is same as the double of the input interval of task B. In a *HP*, there are 5 jobs ($L = 5$).

To realize continuous execution with two different performance core set, periodical core switching is necessary. To minimize energy consumption, the switching interval should be maximized.

First, we start from S_0, which is power off state. Now, we explain how to obtain *start*, which indicates when the first job should start. When first L jobs are executed on S_{low}, if any job will not be ready, the execution should not be started yet to avoid unnecessary idle time.

To obtain *start*, the current time is set to be 0 and then the first job in *HP* has just arrived now. In each core which belongs to C_{low}, begin time of ith job is denoted as b_i and given by follows.

$$b_i = b_1 + \sum_{k=1}^{i-1} et(id(k), c) \ (i = 2, 3, \ldots, L_c). \tag{14}$$

Here, $id(i)$ represents a task ID of ith job. Since the first job just arrives, b_1 is 0. If any b_i is earlier than its arrival time, the arrival violating time is denoted as v_i. The maximum value of them $\max_i(v_i)$ gives the threshold *start*. Finally, *start* is given by follows.

$$start = d - \max_i(v_i) * N. \tag{15}$$

Here, d is a sum of slack times in C_{low} when the first job arrives. When just before execution, the sum of slack times is calculated and compared with the threshold *start*. If the sum of slack times is larger than *start*, the execution should be delayed until the condition will be satisfied.

Next, we discuss when core performance should increase from S_{low} to S_{high}. When the next job is executed on S_{low} and then next L jobs are executed on S_{high}, if any deadline violation occurs, the working state must be changed to S_{high} before executing the next job to meet deadline constraint.

Now, we explain whether the next job should be executed on C_{low} or C_{high} with Fig. 3. In this example, C_{low} consists of $\{MCU_{low}\}$. MCU_{low} executes both task A and B. C_{high} consists of $\{MCU_{high}, MCU_{low}\}$. MCU_{low} executes task A. MCU_{high} executes task B. If multiple tasks are arrived at the same time, we will schedule task A first.

Let's assume the current state is S_{low} and the next job is *B*4. Now, we simulate the execution of the next job (B4) on S_{low}. After this execution, we also simulate next L jobs

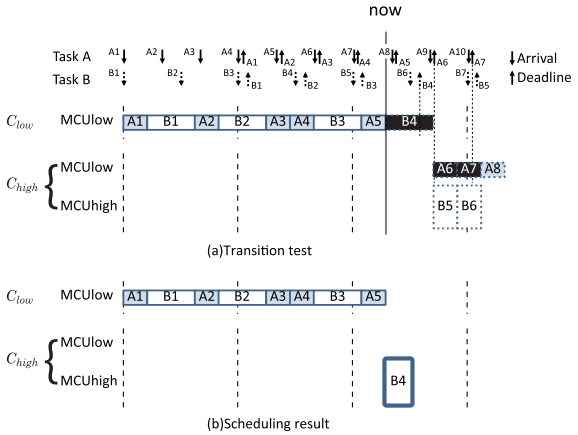


Fig. 3 Scheduling from low to high

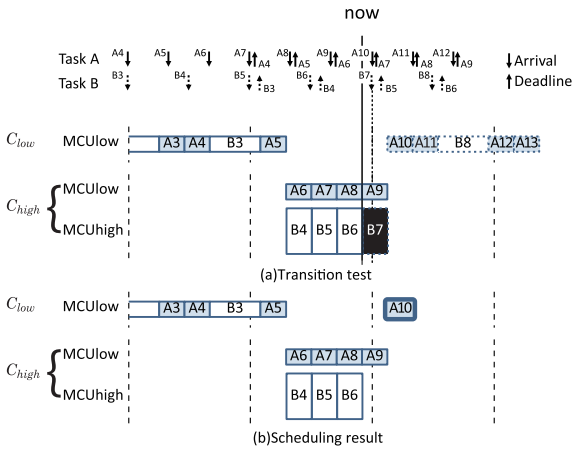


Fig. 4 Scheduling from high to low

(A6, A7, A8, B5 and B6) will be executed on S_{high} as shown with dashed boxes in Fig. 3 (a). Then the completion times of these jobs are easily expected. If any job violates deadline constraint, the next job (B4) should be executed on S_{high}. In this example, since the jobs B4, A6 and A7 violate its deadline constraint as shown with black boxes, the next job (B4) must be executed on S_{high} as shown with a thick box in Fig. 3 (b).

This procedure is repeatedly applied until any deadline violation is expected. When it is expected, the current state is changed to S_{high}.

Next, we discuss when core performance should decrease from S_{high} to S_{low}. When the next job is executed on S_{high} and then next L jobs are executed on S_{low}, if any job will not be ready and any deadline violation is not expected, the working state must be changed to S_{low} before executing the next job.

Now, we explain whether the next job should be executed on S_{high} or S_{low} with Fig. 4. Let's assume the current state is S_{high}, job A9 has been scheduled on S_{high} and the next job is B7. Now, we simulate an execution of the next job (B7) on S_{high}. After this execution, we also simulate next L jobs (A10, A11, A12, A13, and B8) will be executed

on S_{low} as shown with dashed boxes in Fig. 4 (a). Then begin and completion times of these jobs are easily expected. If any job will not be ready and any deadline violation is not expected, the next job (A10) should be executed on S_{low}. Otherwise, the current state must stay on S_{high}. In this example, since the jobs B7 is not ready as shown with a black box, the next job (A10) must be executed on S_{low} as shown with a thick box in Fig. 4 (b).

This procedure is also repeatedly applied until any job will not be ready. When it is expected, the current state is changed to S_{low}. At this moment, at least one core will be powered off. If any job is running on the core, the core will be powered off after the job is finished.

Consequently, while maintaining deadline restrictions, the switching interval is maximized then the total energy consumption is minimized.

4.4.3 Practical Scheduling

Schedulability of proposed algorithm relies on a condition $d(j) \geq HP$. To make our approach more practical, we explain how to guarantee the schedulability when $d(j) < HP$.

When any $d(j)$ is smaller than HP, proposed scheduling may violate deadline constraints during S_{high}. For example, when the next job X is executed on S_{low} and then next L jobs are executed on S_{high}, if any deadline violation does not occur, the job X is executed on S_{low}. However, when $d(j) > HP$, this algorithm cannot guarantee deadline constraints and the future job may violate its deadline. It means the job X must be executed on S_{high}. If necessary, the previous jobs of the job X must also be executed on S_{high}.

To avoid this kind of rollback in a real execution, offline scheduling is required. To guarantee schedulability for indefinitely long times, fixed scheduling for fixed length must be applied repeatedly. Therefore, if we can find periodicity in our scheduling, the schedulability is strictly guaranteed even when $d(j) < HP$.

To find the periodicity, we simulate proposed scheduling with rollback mechanism. When the state is changed from S_{high} to S_{low}, record next job on the new state and executing jobs and their slack times in other cores. If we find the same transition as the previous transition, these transitions are the beginning and the end of the loop of scheduling. This condition can be expressed as follows.

$$\begin{aligned} \forall c \quad & J(k_i, c) = J(k_j, c) \wedge \\ & s(k_i, c) = s(k_j, c) \wedge i < j. \end{aligned} \tag{16}$$

Here J(k,c) is job ID $(1 \cdots L)$ in HP when kth transition on core c and $s(k, c)$ is a slack time which is defined as the difference between the finish time and the deadline on core c. Once we find i and j, which satisfy the above condition, after jth transition is done, scheduling from i to j can be applied repeatedly.

One concern of this method is the size of the scheduling table which stores scheduling from initial state to jth transition. To alleviate this issue, we can relax the condition as follows.

$$\begin{aligned} \forall c \quad & \Big(J(k_i, c) = J(k_j, c) \wedge \\ & \quad s(k_i, c) \leq s(k_j, c) \ \wedge \ i < j \Big) \wedge \\ & \sum_c \Big(s(k_j, c) - s(k_i, c) \Big) \leq \varepsilon. \end{aligned} \tag{17}$$

Here, ε is a threshold, which is larger than 0. This condition allows for ambiguous matching. When this condition is satisfied, by adding small sleep after jth transition on each core c, from ith to jth scheduling can be applied repeatedly. By adjusting the value of ε, we can manage the trade-off between the size of the scheduling table and the energy reduction.

5. Evaluation

5.1 Evaluation Setup

5.1.1 Target Applications

To evaluate the energy efficiency of the proposed scheduling, we considered the most important variation parameter that is the utilization. Additionally, we focus on applications whose deadline periods are longer than their input intervals, which is a characteristic of near real-time periodical tasks. If their deadline periods are shorter than or equal to their input intervals, our scheduling can still guarantee the deadline constraints.

In this evaluation, we use an application that consists of multiple tasks. Task parameters are synthetically generated and shown in Table 2. This application models a system that has some distance sensors and some cameras. The task for a distance sensor is small and the task for a camera is large. To evaluate with a wide range, the number of task is varied. We assume there are three distance sensor tasks and three camera tasks at maximum.

5.1.2 Hardware Environment

We also measured energy parameters using an evaluation board. The board is equipped with an RL78 [18] Micro Controller Unit (MCU) and an RX63N [19] MCU, some sensors, a communication unit and an external NVM. Sensors on the board can help us to measure the energy consumption of each unit separately. In this evaluation, we collect energy parameters of the MCUs, MCU1 and MCU4 are RL78 and RX63N respectively. MCU2 and 3 are generated by interpolating between parameters retrieved from RL78 and RX63N. Since these MCUs have different ISAs, we should

Table 2 Evaluation settings for tasks

Parameters	Task J_1, J_2, J_3	Task J_4, J_5, J_6
Task ID j	1, 2, 3	4, 5, 6
Input interval $I(j)$	100 ms	50 ms
Deadline period $d(j)$	250 ms	

Table 3 Evaluation settings for the hardware platform

	MCU1	MCU2	MCU3	MCU4
core ID c	c_1	c_2	c_3	c_4
Relative Performance $p(c)$	1.0	2.0	3.0	4.0
Power Consumption in				
Active state $P(c)$ [mW]	15.4	36.8	90.9	231
Sleep state $P_s(c)$ [μW]	0.69	2.07	6.20	18.6
Energy Overhead $E_{OV}(c)$[μJ]	51.0	124	253	430
Exec time of				
task J_1, J_2, J_3 [ms]	2.4	1.2	0.8	0.6
task J_4, J_5, J_6 [ms]	54.6	27.3	18.2	13.65

prepare two execution binaries for each task. If the board is equipped with homogeneous ISA MCUs, we may share the same execution binaries. The collected and assumed parameters are shown in Table 3. The energy parameters are calculated from the power and execution time. The performance ratio of these two MCUs is 4 and the execution time of each task is measured as shown in Table 3.

5.1.3 Implementation

To implement the energy-efficient scheduling on a real system, simple calculations to obtain completion times are required. When a target system starts, our scheduler indicates when the first core set should be invoked by *start*. In each active core, the slack time, which is defined as the difference between the current time and the deadline of the next job, is calculated and compared with the thresholds *start*. If the slack time reaches the threshold, the first job starts.

When any job is completed, completion times of next $(L+1)$ jobs are calculated and deadline or arrival time violations are checked. If any violation is detected, the active core set is switched to the other core set.

If there exists no ready job, the core should be switched to a sleep mode. Since the wake up time is easily calculated from the current time and the arrival time of the next job, the appropriate sleep mode can be determined and a simple timer will wake the core up.

We estimate the energy consumption by software simulation with energy parameters which are measured using an evaluation board. Since the computation cost of these procedures is negligibly small, we assumed the scheduler consumed no energy.

5.2 Energy Efficiency

To clarify the merit of our task scheduling algorithm, we evaluate the average power consumption with an in-house simulator, which refers to collected energy parameters.

To evaluate the average power consumption, we first calculate the energy consumption E in a sufficiently long time period T as follows.

$$E = et(j, c_{a(j,i)})P(c_{a(j,i)}) + \sum_{c \in (C_{high} \cup C_{low})} P_s(c) +$$

Table 4 Core selections

Scheduling		4 tasks (J_1 to J_4)	5 tasks (J_1 to J_5)	6 tasks (J_1 to J_6)
Exclusive	MCU_{low}	c_1	c_2	c_3
	MCU_{high}	c_2	c_3	c_4
Proposed	C_{low}	$\{c_1\}$	$\{c_1\}$	$\{c_1, c_2\}$
	C_{high}	$\{c_1, c_2\}$	$\{c_1, c_2\}$	$\{c_1, c_2, c_3\}$

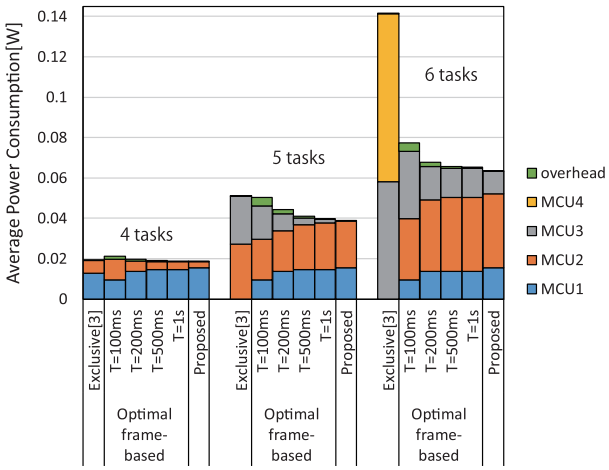


Fig. 5 Average power consumption vs. numbers of tasks with different scheduling algorithms (4,5,6 tasks)

$$\sum_{c \in (C_{high} \Delta C_{low})} (N_{OV}(c) \cdot E_{OV}(c)). \tag{18}$$

Here, $a(j,i)$ indicates core ID which executes ith job of task j, N_{OV} indicates the number of state switching in T. The first and second terms indicate the dynamic and the static energy respectively. The third term indicates the overhead energy of state switching. Here, Δ indicates a symmetric difference of two sets. Then, the average power consumption P is obtained as follows.

$$P = \frac{E}{T}. \tag{19}$$

For comparison, we also calculate the power consumption of Exclusive scheduling [6], which is the same as the proposed scheduling except only one core can be active at the same time. Used cores of each scheduling algorithm are summarized in Table 4.

Any frame-based schedulings repeatedly apply the same scheduling in every HP (100 ms in this evaluation). For comparison, we obtained the Optimal frame-based scheduling, which is the theoretical lower bound, by brute force search. In addition to this, since the optimality depends on the length of the HP, we also evaluate longer HPs, such as 200 ms, 500 ms, 1 s. In these longer HPs, there exist 2, 5 and 10 times jobs respectively.

Figure 5 shows a comparison of the energy consumption and the energy breakdown. The number of tasks N is fixed to 4, 5 and 6 respectively. When N tasks are executed, J_1 to J_N are used. In this figure, each item in the legend represents the dynamic energy of each processor, overhead

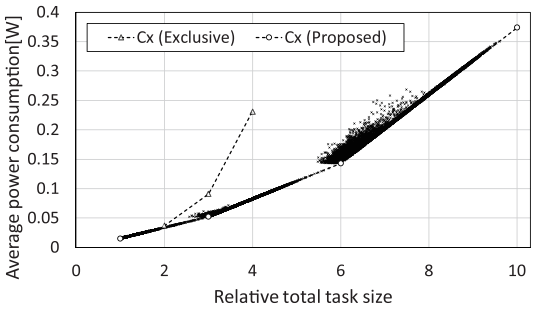


Fig. 6 Average power consumption under different task size

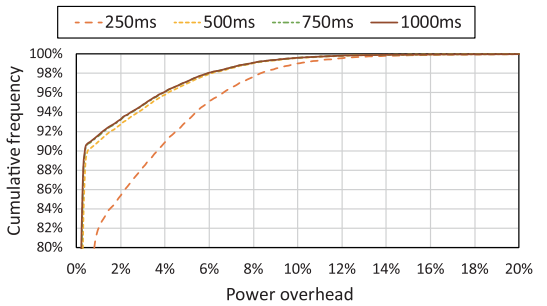


Fig. 7 Average power overhead under different deadline period

is core switching overhead. The y-axis shows the average power consumption.

With 4 tasks, all of Exclusive, Optimal frame-based and Proposed use both MCU1 and MCU2 to minimize dynamic energy. This result also shows the Proposed scheduling achieves the lowest power consumption. Optimal frame-based ($T = 100ms$) consumes non-negligible overhead energy due to their frequent core switching. On the other hand, both Exclusive and Proposed successfully reduce the frequency of core switching. As a result, the energy consumption of overhead is negligibly small.

With 6 tasks, Since, Exclusive uses both MCU3 and MCU4 to meet deadline constraint, the dynamic power becomes significantly high. In contrast, Optimal frame-based and Proposed does not use MCU4 but uses MCU1, 2 and 3. As a result, their schedulings can drastically reduce power consumption. Additionally, Proposed consumes the lowest power and 18% lower power than Optimal frame-based ($T = 100ms$).

When comparing power consumption with different utilization (4, 5, 6 tasks) to evaluate the Proposed scheduling, these results clearly show that MCU1 is always used, because it is the most energy efficient core. Thus, to minimize the dynamic power, the Proposed scheduling successfully uses energy efficient core as much as possible.

5.3 Scalability

To clarify the scalability of the scheduling, we also evaluate power consumption with randomly generated task sets.

Figure 6 shows the result with 20,000 task sets. Each task set consists of from 3 to 40 tasks. Their execution times (i.e. task sizes) are randomly set within the range from 1 to 150 ms. Their intervals are also randomly set within the range from 10 to 200 ms. Their deadlines are fixed to 250 ms. In this figure, the x-axis shows total job size, which normalized by throughput (tp). The y-axis shows the average power consumption. The valuable core sets (C_x) of Proposed and Exclusive, which explained in Sect. 4.2, are also shown with the white circles and triangles respectively. A broken lines connected these circles and triangles show lower limits of power consumption. From these results, in most cases, the energy efficiency is very close to the optimal. These results also show Proposed can successfully reduce power consumption, which is smaller than theoretical lower limit of Exclusive, when relative total task size is larger than 2.

To clarify the energy efficiency, cumulative frequency of power overhead from the lower limit is shown in Fig. 7. In this evaluation, the deadline period is also varied. When the deadline period is 250 ms, more than 90% of cases achieve lower than 4% of power overhead. When the deadline period is longer than 500 ms, more than 90% of cases achieve less than 0.4% of power overhead. This is because of less frequent core switching thanks to longer deadline period. The maximum power overhead is 19.0% when relative total job size and deadline period are 2.56 and 250 ms respectively. This result clearly shows that more energy reduction is possible with longer deadline periods.

6. Conclusion

Near real-time data processing, which is popular in multimedia streaming applications, has a deadline period that is longer than its input interval. Under this situation, energy efficient task scheduling is important while meeting the deadline strictly.

To cope with this challenge, we proposed an energy-aware task scheduling. This scheduler throttles core performance to minimize energy consumption. This scheduling is obtained from hardware and task parameters at design time.

We confirm that our approach can reduce the average power consumption by 18% compared to *optimal* frame-based execution. We conclude that our energy-aware scheduling can drastically reduce the energy consumption of embedded systems while strictly guaranteeing the deadline constraint.

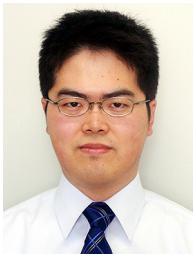
Acknowledgments

This work is supported by Normally-Off Computing Project of NEDO in Japan and JSPS KAKENHI Grant Number JP17H01708.

References

[1] W. Huang and Y. Wang, "An optimal speed control scheme supported by media servers for low-power multimedia applications," Multimedia Systems, vol.15, no.2, pp.113–124, 2009.

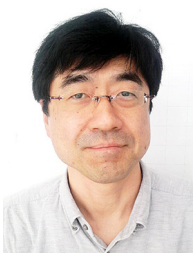
[2] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," Proc. 1st USENIX Conference on Operating Systems Design and Implementation, OSDI '94, 1994.
[3] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," Proc. 36th Annual Symposium on Foundations of Computer Science, pp.374–382, 1995.
[4] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," ACM Trans. Embed. Comput. Syst., vol.13, no.3s, pp.111:1–111:21, 2014.
[5] M.E.T. Gerards and J. Kuper, "Optimal DPM and DVFS for frame-based real-time systems," ACM Trans. Archit. Code Optim., vol.9, no.4, pp.41:1–41:23, 2013.
[6] T. Nakada, H. Yanagihashi, H. Ueki, T. Tsuchiya, M. Hayashikoshi, and H. Nakamura, "Energy-efficient continuous task scheduling for near real-time periodic tasks," IEEE International Conference on Internet of Things, pp.675–681, 2015.
[7] T. Nakada, H. Yanagihashi, K. Imai, H. Ueki, T. Tsuchiya, M. Hayashikoshi, and H. Nakamura, "Energy-aware task scheduling for near real-time periodic tasks on heterogeneous multicore processors," IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 2017), pp.1–6, Oct. 2017.
[8] T.D. Burd and R.W. Brodersen, "Energy efficient CMOS microprocessor design," Proc. Twenty-Eighth Hawaii International Conference on System Sciences, pp.288–297, 1995.
[9] F. Pollack, "Micro32 conference keynote," Intel Corp., 1999.
[10] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "Energy reduction techniques for systems with non-DVS components," 2009 IEEE Conference on Emerging Technologies Factory Automation, pp.1–8, Sept. 2009.
[11] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time DAG tasks," pp.22:1–22:21, 01 2017.
[12] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R.V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp.1–12, April 2016.
[13] J. Rho, T. Azumi, M. Nakagawa, K. Sato, and N. Nishio, "Scheduling parallel and distributed processing for automotive data stream management system," Journal of Parallel and Distributed Computing, vol.109, pp.286–300, 2017.
[14] W. Zhang, E. Bai, H. He, and A. Cheng, "Solving energy-aware real-time tasks scheduling problem with shuffled frog leaping algorithm on heterogeneous platforms," Sensors, vol.15, no.6, pp.13778–13804, June 2015.
[15] C. Xian, Y.H. Lu, and Z. Li, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.27, no.8, pp.1467–1478, 2008.
[16] S. Albers and A. Antoniadis, "Race to idle: New algorithms for speed scaling with a sleep state," ACM Trans. Algorithms, vol.10, no.2, pp.9:1–9:31, Feb. 2014.
[17] T. Nakada, K. Okamoto, T. Komoda, S. Miwa, Y. Sato, H. Ueki, M. Hayashikoshi, T. Shimizu, and H. Nakamura, "Design aid of multi-core embedded systems with energy model," IPSJ Transactions on Advanced Computing Systems, vol.7, no.3, pp.37–46, 2014.
[18] Renesas Electronics Corporation, "RL78 Family." https://www.renesas.com/products/microcontrollers-microprocessors/rl78.html.
[19] Renesas Electronics Corporation, "RX63N, RX631." https://www.renesas.com/products/microcontrollers-microprocessors/rx/rx600/rx63n-631.html.



Takashi Nakada received his M.E. and Ph.D. degrees in Engineering from Toyohashi University of Technology in 2004 and 2007 respectively. He has been an Associate Professor at the Nara Institute of Science and Technology since 2016. His research interests includes Normally-Off Computing, processor architecture and related simulation technologies. He is a member of IEEE, ACM and IPSJ.

Hiroyuki Yanagihashi received his B.E. and M.E. degrees from the University of Tokyo in 2014 and 2016 respectively. His research interests are task scheduling and its analysis.

Kunimaro Imai received his B.S. degree in Law from Kansai University in 1998. In 2000, he joined Proassist Ltd. and currently belongs to Solution Business Div. System Development Dpt. #1. He is software engineer mainly involved in FA and image processing.



Hiroshi Ueki received the B.S. and M.S. degrees in physics and nuclear technology from Kyoto University. Since 1991, he has been involved in microcontroller and SoC design, in Mitsubishi Electric Corporation and Renesas Electronics Corporation. He developed CPU and peripheral circuits for HDD controller, flash-memory control module for microcontroller and power management module for SD-card controller.

Takashi Tsuchiya received his B.E. degree in electronic engineering from Doshisha University. He joined Mitsubishi Electric Semiconductor Software Corporation in 1984. He has been an Application Engineer of Embedded Microcomputer and Soc in Consumer field. He is currently a Senior Principal Engineer of IoT Platform Business Division in Renesas Electronics Corporation.

Masanori Hayashikoshi received the B.S. and M.S. degrees in electronic engineering from Kobe University, Kobe, Japan, in 1984 and 1986, respectively, and the Ph.D. degree in electrical engineering and computer science from Kanazawa University, Ishikawa, Japan, in 2018. In 1986, he joined the LSI Research and Development Laboratory, Mitsubishi Electric Corporation, Hyogo, Japan. Since then he has been engaged in the research and development of non-volatile memories, high density DRAMs, and low power SDRAMs. Now, he is a Senior Principal Specialist of Shared R&D Core Technology Division in Renesas Electronics Corporation. He has been engaged in the research and development of Flash memory, embedded MRAM for MCUs, Normally-off computing architecture for low-power solutions with NVRAMs, and now in-memory computing solutions for future new applications.

Hiroshi Nakamura is a Professor in the Department of Information Physics and Computing at The University of Tokyo. He received the Ph.D. degree in Electrical Engineering from The University of Tokyo in 1990. His research interests include power-efficient computer architecture and VLSI design for high-performance and embedded systems. He is a senior member of IEEE and ACM.