PAPER

# An Energy-Efficient Task Scheduling for Near-Realtime Systems with Execution Time Variation

**Takashi NAKADA**[†a)], ***Member***, **Tomoki HATANAKA**[††], **Hiroshi UEKI**[†††], **Masanori HAYASHIKOSHI**[†††], ***Nonmembers***, **Toru SHIMIZU**[††††], ***Senior Member***, ***and*** **Hiroshi NAKAMURA**[††], ***Member***

**SUMMARY** Improving energy efficiency is critical for embedded systems in our rapidly evolving information society. Near real-time data processing tasks, such as multimedia streaming applications, exhibit a common fact that their deadline periods are longer than their input intervals due to buffering. In general, executing tasks at lower performance is more energy efficient. On the other hand, higher performance is necessary for huge tasks to meet their deadlines. To minimize the energy consumption while meeting deadlines strictly, adaptive task scheduling including dynamic performance mode selection is very important. In this work, we propose an energy efficient slack-based task scheduling algorithm for such tasks by adapting to task size variations and applying DVFS with the help of statistical analysis. We confirmed that our proposal can further reduce the energy consumption when compared to oracle frame-based scheduling.

***key words:*** *adaptive task scheduling, near real-time processing, execution time variation, energy efficiency*

## 1. Introduction

Energy consumption of embedded systems is a very critical concern and minimizing it is always of great importance. In most embedded systems, tasks periodically arrive and are then executed. As information processing in such devices becomes highly sophisticated, execution time of tasks vary significantly. Buffering is effective to mitigate load imbalance in the time domain. The key idea of near real-time systems is to trade off between the latency and required peak performance.

In this paper, *Input interval length* is defined as the distance between arrival times of successive tasks. *WCET* (*Worst Case Execution Time*) is defined as the maximum execution time. The simplest task scheduling algorithms assume the execution time is the same as WCET. However, the execution time is often shorter than WCET due to dynamic behaviors such as input-dependent variations.

To adapt to the variation of the execution time, DVFS (Dynamic Voltage and Frequency Scaling) [1], [2] can be very effective. Therefore, execution time variation aware adaptive task scheduling is indispensable for low power embedded systems. To cope with this challenge, some execution time variation aware algorithms have been proposed. However, these approaches are insufficient for near real-time processing because most of them assume that the deadline period is the same as the input interval and schedule tasks within only one input interval.

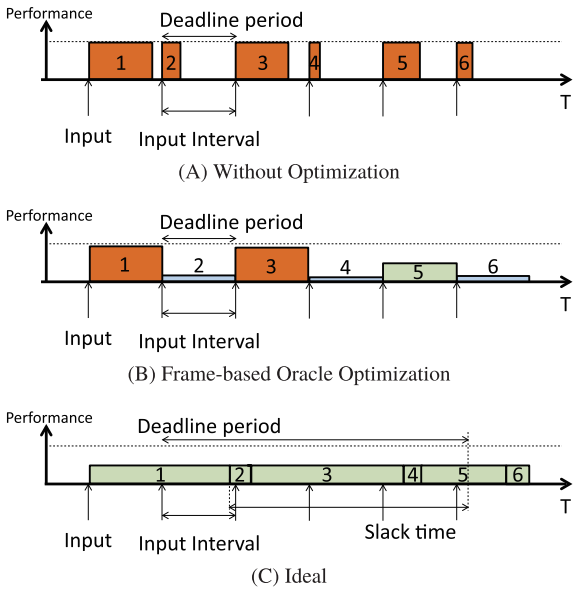



**Fig. 1** Periodic tasks with task size variation

We present a motivational example in Fig. 1. We start from executing periodic dynamic tasks without any optimization shown in Fig. 1 (A). In this execution, to meet the deadline of whatever task size, the tasks are always executed under a high performance mode (Orange). When we apply DVFS technology to these tasks with a *perfect* execution time prediction, the execution is then shown in Fig. 1 (B). In this execution, we assume the deadline period is the same as the input interval. In each interval, optimal execution is achieved locally. Specifically, when the execution time is middle (5th) or short (2nd, 4th, 6th), a middle (Green) or a low (Blue) performance mode can be used respectively. For near real-time systems, whose deadline periods are longer than the input intervals, the ideal scheduling is then shown as Fig. 1 (C). In this execution, middle performance mode (Green) is always used and incurs the smallest energy consumption. As a result, the execution time of large tasks

Manuscript received December 19, 2016.
Manuscript revised May 19, 2017.
Manuscript publicized June 26, 2017.

[†]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 authors are with Renesas Electronics Corporation, Kodaira-shi, 187–8588 Japan.
[††††]The author is with Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: nakada@is.naist.jp

DOI: 10.1587/transinf.2016EDP7497

are longer than the input interval, and the average execution time is the same as the input interval. This scheduling is theoretically the most energy efficient.

Our approach here is to firstly propose a *slack-based* task scheduling method that can adapt to dynamic behaviors and take advantage of the longer deadline period. *Slack time* is defined as the time from the current time to the earliest deadline. For example, the slack time when the first task completes is shown in Fig. 1 (C).

For near real-time systems, the execution time for a task varies and is not fixed in advance. As an example, for image processing or other types of streaming processing, required execution time heavily depends on the characteristics of the input. Therefore, the target of our scheduling algorithm are those tasks whose execution times are known only after completion.

When the slack time is long enough, lower performance is preferable to maximize the energy efficiency. The slack time becomes shorter if some tasks, whose execution time is long, continuously arrive. Then we need higher performance to meet the deadline. In this circumstance, we adaptively adjust the performance to minimize the energy consumption. As the performance adjustment incurs additional time and energy overhead, frequent adjustment may increase the total energy consumption. So, our scheduling should be guided with several thresholds, which are compared with the slack time and indicate when the performance should be changed. Additionally, these values are obtained at design time.

To seek for optimal thresholds, we derive the average energy consumption from a statistical analysis. Firstly, we introduce a finite state machine (FSM) that represents a system controlled by the proposed slack-based task scheduling. The states and the transitions represent which performance is chosen and when the performance should be changed respectively. By doing so, next task will be executed on an optimal performance and the average energy consumption will be minimized. When task and hardware parameters are given, the average energy consumption can be calculated from a statistical analysis. Finally, the optimal thresholds can be obtained with a suitable heuristic search algorithm.

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

- We propose a slack-based scheduling that adapts to execution time variation and takes advantage of deadline periods that are longer than input intervals.
- We formulate our slack-based scheduling with an FSM to properly regulate the performance of tasks and we derive the average energy consumption from statistical analysis as threshold for performance change.
- Through evaluation, we observe significant energy reduction with our proposal when compared to *oracle* frame-based task scheduling.

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

## 2. Background

In this paper, we assume that a system that has a performance adjustable core. Namely, our target system has a core with DVFS technology.

We also assume that target tasks periodically arrive and their sizes are unknown before their executions complete but the probability distribution of the task sizes is given. The execution time, including WCET, can be calculated by the task sizes and the processor performances. This type of tasks is called *dynamic tasks*. The tasks are independent with each other, namely, each task and its task size are independent from the previous tasks. Their deadline period is longer than the input interval.

In general, the relation between energy, voltage and clock frequency can be modeled by following known equation [3].

$$E_{proc} = \alpha_1 T_1 C V^2 f + T_2 V I_{leak}$$

Here, $E_{proc}$ represents the energy consumption of the microprocessor. $\alpha_1, T_1, C, V$ 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. While, the latter is caused by leakage current and always consumed whenever power is supplied.

DVFS has been around for more than a decade [1]. It allows the voltage and the clock frequency to be decreased dynamically to trade time for dynamic energy. DPM (Dynamic Power Management) is also important to reduce the static power when the processor core is in an idle state. An overview of DPM techniques is given in a survey article [4].

To determine the appropriate power state, the length of the next idle period is important. For short idle period, shallower sleep mode is preferable and vice versa. This strategy can be modeled with a function of cost against the length of the idle period. This function turns out to be piecewise-linear, increasing and concave [5]. Even if the task sizes are known, NP-hardness of the optimal scheduling algorithm was proven [6].

In this paper, executed tasks periodically arrive and wake-up time is statically scheduled. So, when an idle state is encountered, the time when the next task can be invoked is definitely predictable. Then the length of the idle period is also predictable. Additionally, since the restart time is predictable, wakeup overheads are easily hidden by a pre-wakeup technique. Therefore, the optimal power management is easily determined by the strategy. To simplify the

discussion, we assume that the optimal power mode is automatically chosen. Namely, after the execution is finished, the length of an idle period is easily calculated. The optimal power mode can be chosen from the length of the idle period.

## 3. Related Work

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

A simple strategy is a straightforward extension of a static task scheduling [7]–[10]. Their target applications consist of multiple dynamic subtasks. These algorithms initially assume the execution times of all subtasks are the same as WCETs. During the execution, when the actual execution time is shorter than WCET, they re-calculate and switch to lower performance mode. However, these algorithms always start from the highest performance mode to meet the deadline period of the largest task, even if larger task rarely appeared.

To solve this problem, more sophisticated scheduling [11], [12] is proposed. This approach assumes a wider range of performance modes and start from medium performance mode. Then, if the execution time is longer than expected, following subtasks should be in higher performance mode, which is higher performance of the previous approaches. As a result, if larger tasks rarely arrived, more power reduction can be obtained. To realize this management, they introduced a scheduling table. At design time, they construct the table that indicates the optimal scheduling. Using this table, run-time overhead is minimized. Similar approaches are proposed for conditional task graph [13], [14].

However, the main drawback of these frame-based optimization algorithms is, they assume that the deadline period is the same as the input interval even it is indeed longer than the input interval. In other words, these approaches are not optimized for near real-time systems. As a result, the scheduling is optimized within each input interval independently and the improvement of the energy efficiency is limited. Only a limited numbers of related work tackled this problem.

A basic approach for near real-time systems is proposed [15]. Since this approach focuses on static tasks, dynamic tasks are not supported.

Approaches for near real-time dynamic processing are a prediction based and a feedback control based schedulings [16], [17]. The effectiveness of these approaches relies on the accuracy of their execution time predictors. When the prediction fails, deadline constraint may be violated. Additionally, such run-time prediction must need additional computational cost and is a waste of energy.

Another approach is buffer based scheduling [18]. This scheduling postpones execution to keep continuous execution and tries to utilize minimum performance core. This greedy approach can realize optimal scheduling only for next task but not globally optimal. For example, when minimum performance core is chosen and execution time of next task is same as WCET, following task should be executed on the highest performance core to meet deadline restriction. Therefore, the greedy approach cannot minimize total energy consumption. Moreover, they did not consider mode switching overhead at all.

On the other hand, our approach can adapt to both the execution time variation and take advantage of the longer deadline. Our scheduling can schedule tasks across multiple input intervals. Namely, multiple tasks are held in an input buffer and then execute multiple tasks continuously. Basic idea of this scheduling has been introduced in [19]. In this paper, we introduce details of its algorithm and further evaluation.

## 4. Slack-Based Task Scheduling

In this section, we propose a novel adaptive energy-efficient task scheduling algorithm for near real-time data processing. Specifically, our approach can adapt to the execution time variation and take advantage of the longer deadline.

In general, a task behavior can be expressed by a task graph. In case of the dynamic task, the task graph includes branches as shown in Fig. 2. Each branch has a probability of execution and each node represents the task size. Note that the number of branches corresponds to the number of task size variations and only one node is selected and executed for each input interval. If original task graph has multiple subtasks through the path, they should be merged before applying our algorithm. We also assume these subtasks are non-preemptive.

### 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.

As we mentioned, the tasks are independent with each other and the deadline $d$ is longer than the input interval $I$. The number of possible patterns is expressed by $M$. In general, the task graph may contain multiple branches as

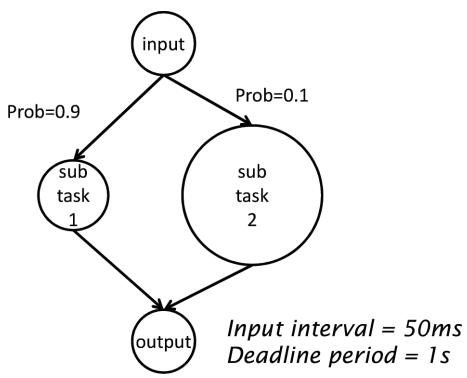



**Fig. 2** An example task graph

**Table 1** Input variables

| Variables | Definition |
|---|---|
| $M$ | Number of possible execution paths |
| $p(a)$ | Probability of path $a$ |
| $I$ | Input Interval |
| $d$ | Deadline |
| $N$ | Number of modes |
| $et(a, c)$ | Execution time of path $a$ on mode $c$ |
| $E_d(a, c)$ | Dynamic energy of path $a$ on mode $c$ |
| $E_D$ | Average dynamic energy per one execution |
| $E_{Sa}$ | Average static energy per one execution |
| $E_{Ss}$ | Static energy during sleep state per input interval |
| $E_{OV}$ | Overhead energy of power state transitions |
| $T_{OV}$ | Overhead latency of power state transitions |

shown in Fig. 3 (A). If a task graph contains two sequential branches, which have two directions, this task graph can be transformed into a task graph that contains only one branch, which has four directions, as shown in Fig. 3 (B). Each branch has only one subtask and nonpreemptive. If these two branches have dependency, the probabilities may differ from the values shown in Fig. 3 (B) but are still easily obtained. In the same way, when the task graph contains some loops, possible patterns of the execution time are enumerable and their probabilities are easily obtained.

Moreover, if there are multiple tasks, we have to transform them into a task graph that contains only one branch. For example, one application is 100ms execution every 1 second and the other is 50ms execution every 2 seconds. The combined application is 100ms or 150ms execution every 1 second and their probability distribution is 50% and 50%. This transformation lose some information of the applications and may reduce energy efficiency.

As a result, each path has an ID $a$ ($a = 1, 2, 3, \ldots, M$). $p(a)$ represents the probability of the subtask on path $a$. If these probabilities are not available in advance precisely, expected probabilities are also acceptable. Even if these expected values are significantly different from actual values, the deadline constraint is still guaranteed as long as the WCET is given correctly. Although, the energy efficiency is degraded.

Additionally, if the probability distribution is changed depended on the execution phase or input data, we can prepare the optimal scheduling for each execution phase or each typical input data at design time. An adaptive scheduling can be realized by monitoring execution behavior and selecting suitable scheduling depending on expectation of current execution phase or input data. In this paper, to simplify the discussion, we assume that the probability distribution is predictable and fixed through execution.

For a processor that has DVFS technology, each performance mode has ID $c$ ($c = 0, 1, 2, 3, \ldots, N$). $N$ represents the number of available modes. $c = 0$ represents sleep mode. We assume the higher performance modes have larger IDs. $et(a, c)$ and $E_d(a, c)$ represent the execution time and the dynamic energy when path $a$ is executed on mode $c$ respectively.

$E_D$ and $E_{Sa}$ are $[(N + 1) \times 1]$ vectors and $i$th entry represents the average dynamic energy and static energy of one execution on mode $i$ respectively. Note that 0th entries are always 0. $E_{OV}$ is an $[(N+1)\times(N+1)]$ matrix and $(i, j)$ entry represents energy overhead of the transition from mode $i$ to mode $j$. $E_{Ss}$ is constant and represents the total static energy during an input interval. $E_{Ss}$ is given by $P_{Ss} \times I$.

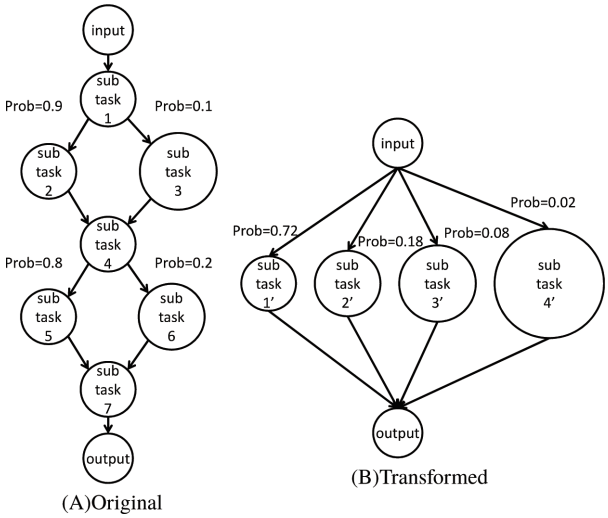



**Fig. 3** Task graph with multiple branches

We assume that the core in the higher performance mode execute any tasks faster with larger energy consumption. Otherwise, the low performance but larger energy consumption mode should be removed. Therefore, the following equations are satisfied.

$$\begin{aligned} For\ c_1 < c_2 : \quad & et(a, c_1) > et(a, c_2) \\ & E_d(a, c_1) < E_d(a, c_2) \end{aligned}$$

Our ultimate goal is to minimize the energy consumption under performance constraint. However, the task size of the target tasks varies probabilistically. The objective function should minimize expected value of the energy consumption without deadline violation.

As a result, objective function and the constraint condition are as follows. We solve this optimization problem in the following sections.

$$\begin{aligned} & min\ (Average\ energy\ consumption\ of\ core\ per\ input) \\ & s.t. \qquad (Satisfy\ deadline\ constraints) \end{aligned}$$

## 4.2 Task Scheduling

### 4.2.1 Overview

In this paper, we adopt a lumped execution [20], which executes multiple tasks continuously. To realize this execution, we do not execute a task just after it is available. We postpone the execution but still make sure the deadline constraint can be met. After several tasks are ready, we start the execution. We can then execute several tasks continuously. When all the ready tasks are done, the execution is

**Table 2** Execution time

| mode ID $c$ | $et(1,c)$ | $et(2,c)$ |
|---|---|---|
| 1 | 0.3 | 3.6 |
| 2 | 0.15 | 1.8 |
| 3 | 0.1 | 1.2 |
| 4 | 0.075 | 0.9 |

terminated and the core switches to the sleep mode.

To adapt to the variation of the task size. The executing mode should be chosen carefully. Firstly, the maximum execution time on mode $c$ ($\max_a et(a,c)$) corresponds to WCET. To ensure the schedulability, the execution time when the longest path is executed on the highest performance mode $N$ must be shorter than the input interval $I$. Meanwhile, the lowest performance mode must execute the longest path shorter than the deadline $d$. Otherwise, such performance modes should be disabled. These conditions are given as follows.

$$\max_a(et(a,N)) < I, \quad \max_a(et(a,1)) < d$$

Notice that WCETs of all modes except mode $N$ are longer than the input interval $I$.

We introduce an overview of the proposed task scheduling algorithm with an example shown in Fig. 2. In this example, the small subtask has large probability and the large subtask has small probability. We assume four processor modes and the execution time is shown in Table 2.

In this case, if we use only mode 4, we can obviously satisfy the deadline constraints. However, the mode 4 seems to be over performance. On the other hand, if we use other mode, the deadline constraints may violated when the large task arrived continuously.

As mentioned in Sect. 1, to realize this task scheduling, we focus on the *slack time*, which is defined as the time from present to the earliest deadline. In the end it comes down to a problem that when the core performance should be changed.

### 4.2.2 Scheduling Algorithm

Here, we explain the details of the proposed task scheduling algorithm. As we mentioned, the working mode is changed based on the slack time. When a task finishes, the slack time for the next oldest task is calculated. Note that the oldest task has the earliest deadline. If the slack time exceeds thresholds, the working mode is changed. The mode is mainly changed to its adjacent performance mode, but if the slack time is largely changed, the core performance should also be largely changed.

As a result, it comes down to a problem that when the working mode should change. The scheduling variables are shown in Table 3. Firstly, the thresholds from mode $i$ to mode $i+1$ are defined as $x^i_{up}$, the threshold from mode $i+1$ to mode $i$ are defined as $x^i_{down}$. Finally, the first working mode is defined as mode $\star$. When current state is $s_i$, while the slack time is between $x^i_{up}$ and $x^{i-1}_{down}$, the tasks are executed on mode $i$ continuously. This range is defined as stage

**Table 3** Scheduling variables

| Param | Definition |
|---|---|
| $x^i_{up}$ | Threshold from $s_i$ to $s_{i+1}$ |
| $x^i_{down}$ | Threshold from $s_{i+1}$ to $s_i$ |
| $\star$ | The first working mode |

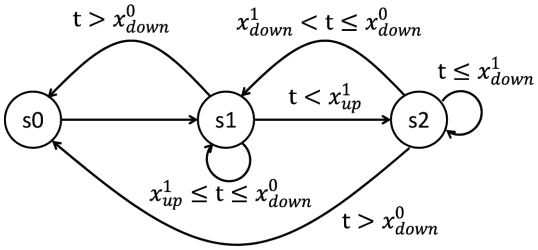



**Fig. 4** State transition diagram and transition conditions

$i$. Therefore, $x^0_{up}$ is the threshold of the wake up from the sleep state. $x^0_{down}$ is the threshold to the sleep state. From a viewpoint of energy efficiency, there is no reason to sleep while any task is available. Thus $x^0_{down} = d$, namely, when the next task is not ready, this system goes to the sleep state.

In this situation, since the wake up time is easily calculated from the current time and the scheduling variable ($x^0_{up}$), the appropriate sleep mode can be determined and a simple timer will wake the core up. If the calculated sleep period is too short even for the shallowest sleep mode, the appropriate mode is idle mode.

The slack time is calculated when a task is finished. The slack time when $n$th task in a lumped execution is finished is defined as $t_n$ and written as follows.

$$t_0 = x^0_{up} \tag{1}$$

$$t_n = t_{n-1} + (I - et(a,c)) \tag{2}$$

From this definition, if the execution time is longer than the input interval, namely $I - et(a,c) < 0$, the slack time is decreased. Otherwise, if the execution time is shorter than the input interval, namely $I - et(a,c) > 0$, the slack time is increased. When $n$th task is executed on mode $i$, if $t_n$ is smaller than $x^i_{up}$, the next state is $s_j$ that satisfies $x^j_{up} \le t_n - T_{OV(i,j)} < x^{j-1}_{up}$. Conversely, if $t_n$ is larger than $x^{i-1}_{down}$, the next state is $s_j$ that satisfies $x^{j+1}_{down} \le t_n - T_{OV(i,j)} < x^j_{down}$. Otherwise, the next task is also executed on mode $i$. This transition when N=2 is shown in Fig. 4.

The threshold $x^i_{up}$, $x^i_{down}$ and the first working mode $s_\star$ satisfy following constraints.

$$x^{(i+1)}_{up} \le x^i_{up} \, (i = 1, 2, \ldots, N) \tag{3}$$

$$x^i_{down} \le x^{(i-1)}_{down} \, (i = 1, 2, \ldots, N) \tag{4}$$

$$x^i_{up} \le x^i_{down} \, (i = 0, 1, \ldots N) \tag{5}$$

$$WCET_i \le x^i_{up} \, (i = 1, 2, \ldots, N) \tag{6}$$

$$x^\star_{up} \le x^0_{up} \quad \le \quad x^{(\star-1)}_{down} \tag{7}$$

Here, $WCET_i$ is $WCET$ of mode $i$. The constraints (3) and

**Table 4** Variables for problem formulation

| Variables | Definition |
|---|---|
| $P$ | Transition probability matrix |
| $\pi$ | Stationary distribution of $P$ |
| $t_{active}$ | Average active time |
| $t_{sleep}$ | Average sleep time |

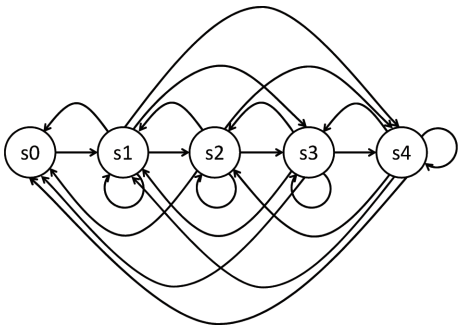



**Fig. 5** State transition diagram (N=4)

(4) keep the performance order. The constraint (5) guarantees that at least one state is available for any slack time. The constraint (6) is required to meet the deadline constraints. The constraint (7) guarantees that the first transition is valid. Note that the $x^0_{up}$ is not included in the constraints (3) and (6). Since, the $x^0_{up}$ is the threshold for wake up, $x^0_{up} < x^1_{up}$ is valid. Whenever these constraints are satisfied, it is guaranteed to meet the deadline for any task sequence.

Next, we explain how the objective function follow from the thresholds $x^i_{up}$, $x^i_{down}$ and the first working mode $\star$.

### 4.3 Energy Analysis Based on FSM

In this section, we introduce a finite state machine (FSM) to analyze proposed scheduling statistically. The introduced variables are shown in Table 4. Each state corresponds to the working mode or the sleep state. Thus, the total number of states is $N + 1$. When the working mode is changed, the state is also changed. The state transition diagram is shown in Fig. 5. The sleep state is $s_0$, the mode $i$ is working at $s_i$. Since the first working mode is fixed in our scheduling, possible transition from the $s_0$ is only one and the first working mode is a scheduling variable.

Then, we can define a transition probability matrix $P$ from this FSM. P is an $[(N+1)\times(N+1)]$ matrix and its row sums are always 1. $(i, j)$ entry of $P$, namely $p_{ij}$, is defined as follows.

$$p_{ij} = (\textit{probability of the transition from } s_i \textit{ to } s_j) = \begin{cases} \textit{from mode i to mode j} & \text{if } ij \neq 0 \\ \textit{from mode i to sleep state} & \text{if } i \neq 0, j = 0 \\ \textit{from sleep state to mode j} & \text{if } i = 0, j \neq 0 \\ 0 & \text{if } i = j = 0 \end{cases} \tag{8}$$

Here $p_{0j}$ also satisfies the following equations.

$$p_{0j} = \begin{cases} 1 & \text{if } \textit{mode j is the first working mode} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

In the example shown in Fig. 4, $p_{01} = 1$.

A stationary distribution $\pi$ is also defined for $P$. $\pi$ is $[1 \times (N + 1)]$ vector and $\pi P = \pi$ is satisfied. Therefore, $\pi$ represents that how often each state is active.

Additionally, an average active time $t_{active}$ and average sleep time $t_{sleep}$ are defined as follows.

$$t_{active} = \textit{average total lumped execution time} \tag{10}$$

$$t_{sleep} = \textit{average time between lumped executions} \tag{11}$$

Using these values, the objective function $J$ is defined as follows.

$$J = \pi(E_D + E_{Sa}) + \parallel \pi(P * E_{OV}) \parallel + \frac{t_{sleep}}{t_{active} + t_{sleep}} E_{Ss} \tag{12}$$

Here, $*$ and $\parallel \cdot \parallel$ represent the products of correspond entries and the sum of all entries respectively. Therefore, the first and second terms represent average dynamic and static energy consumption per task. The third term represents average overhead energy per transition. The fourth term represents static energy per input interval during sleep state. The stationary distribution $\pi$, average sleep time $t_{active}$ and average sleep time $t_{sleep}$ can be obtained from the transition probability matrix $P$.

The initial distribution of the slack time depends on the previous state. Namely, the probability of transition from $s_i$ to $s_j$ depends on the previous state. Therefore, before calculating $P$, we define $N^2 - N + 2$ state as follows and construct their transition probability matrix $\hat{P}$. $\hat{P}$ is an $[(N^2 - N + 2) \times (N^2 - N + 2)]$ matrix.

$$\begin{array}{l} \textit{state } s_0 \\ \sigma(\star, 0) \\ \left.\begin{array}{l} \sigma(1,2) \\ \sigma(1,3) \\ \dots \\ \sigma(1,N) \end{array}\right\} N-1 \\ \left.\begin{array}{l} \sigma(2,1) \\ \sigma(2,3) \\ \dots \\ \sigma(2,N) \end{array}\right\} N-1 \\ \dots \\ \left.\begin{array}{l} \sigma(N,1) \\ \sigma(N,2) \\ \dots \\ \sigma(N,N-1) \end{array}\right\} N-1 \end{array} \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right\} \times N \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right\} N(N-1)+2 \tag{13}$$

Here, $\sigma(j, i)$ means the current state is $s_i$ and the previous state is $s_j$.

To obtain the initial distribution of state $i$, we assume that the distribution of the slack time is uniform at the previous state $j$ just before the transition. Then initial distribution $r_i(t)$ of $s_i$ is given below.

$$r_i(t) = \begin{cases} \int_{-\infty}^{\infty} v_j(\tau) U_j(t-\tau) d\tau & \text{if } x_{up}^i \le t \le x_{down}^{i-1} \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Here, $U_j(t)$ is a uniform distribution that is valid only on the previous stage $j$, $v_j(t)$ means that the probability distribution of slack time variation is $t$, when one task is executed on mode $j$. Notice that $r_i(t)$ is defined only on the stage $i$ ($[x_{up}^i, x_{down}^{i-1}]$) Therefore, the integrated value of $r_i(t)$ within $[-\infty, \infty]$ may be smaller than 1 and it is required to be normalized as follows.

$$\tilde{r}_i(t) = \frac{r_i(t)}{\int_{-\infty}^{\infty} r_i(t) dt} \tag{15}$$

Then the transition probabilities are obtained from the initial distributions. After one task is executed, the probability distribution of the slack time $v_i^1(t)$ can be obtained from a convolution of the initial distribution $r_i(t)$ and the probability distribution of the slack time variation $p_i(t)$.

$$v_i^1(t) = \int_{-\infty}^{\infty} p_i(\tau) \tilde{r}_i(t-\tau) d\tau \tag{16}$$

Here $p_i(t)$ is defined as follows.

$$p_i(t) = \begin{cases} p(a) & \text{if } I - et(a,i) = t \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

In the same way, the probability distribution of slack time after $n$ executions $v_i^n(t)$ is obtained from a convolution of the $v_i^{n-1}(t)$ and $p_i(t)$.

$$v_i^n(t) = \int_{-\infty}^{\infty} p_i(\tau) v_i^{n-1}(t-\tau) A(t-\tau) d\tau \tag{18}$$

Here, $A(t)$ is a window function and given by

$$A(t) = \begin{cases} 1 & \text{if } x_{up}^i \le t \le x_{down}^{i-1} \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

This window function limits $v_i^{n-1}$ on the stage $i$.

At the $n$th transition, the probability of the slack time exceeds $x_{up}^i$ or $x_{down}^{i-1}$ are defined as $p_{left}^{(n)}$, $p_{right}^{(n)}$ respectively. However, the slack time may exceed more than 1 threshold after one execution. Thus, the probability of the slack time exceed $m$ threshold are defined as $p_{left(m)}^{(n)}$, $p_{right(m)}^{(n)}$ and given by

$$p_{left(m)}^{(n)} = \int_{x_{up}^{i+m}}^{x_{up}^{i+m-1}} v_i^n(t) dt - \sum_{k=1}^{n-1} p_{left(m)}^{(k)}$$

$$p_{right(m)}^{(n)} = \int_{x_{down}^{i-m}}^{x_{down}^{i-m-1}} v_i^n(t) dt - \sum_{k=1}^{n-1} p_{right(m)}^{(k)}$$

As a result, the state transition probability $\hat{P}$ is obtained from $p_{left(m)}^{(n)}$ and $p_{right(m)}^{(n)}$ as follows.

$$\hat{p}_{\sigma(j,i),\sigma(i,i+m)} = \sum_{k=1}^{\infty} \frac{p_{left(m)}^{(k)}}{k}$$

$$\hat{p}_{\sigma(j,i),\sigma(i,i-m)} = \sum_{k=1}^{\infty} \frac{p_{right(m)}^{(k)}}{k}$$

$$\hat{p}_{\sigma(j,i),\sigma(j,i)} = 1 - \hat{p}_{\sigma(j,i),\sigma(i,i+m)} - \hat{p}_{\sigma(j,i),\sigma(i,i-m)}$$

Here $\sigma(j,i)$ represents the current state is $i$ and the previous state is $j$. This expression corresponds to the definition of $\hat{P}$ is given by (13). Then an $[(N^2 - N + 2) \times (N^2 - N + 2)]$ matrix $\hat{P}$ is calculated.

Next, we obtain stationary distribution $\hat{\pi}$ from $\hat{P}$. We start from $q_0$, whose 1st entry is set to 1 and others are 0 as follows.

$$q^0 = \begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix} \tag{20}$$

Then, $n$ time state transition is applied to $q^0$ and its probability distribution is defined as $q^n$. Thus, $q^0$ is given by

$$q^n = q^0 \hat{P}^n. \tag{21}$$

When $\| q^n - q^{n-1} \|_2$ becomes negligibly small, this computation is converged and the stationary distribution $\hat{\pi}$ is obtained. This $\hat{\pi}$ corresponds to right eigenvalues of the matrix $\hat{P}$.

Then, using the $\hat{P}$ and the $\hat{\pi}$, the state transition matrix $P$ and its stationary distribution $\pi$ are given by

$$p_{i,j} = \sum_k \hat{p}_{\sigma(k,i),\sigma(i,j)} \times \hat{\pi}_{\sigma(k,i)} \tag{22}$$

$$\pi_i = \sum_k \hat{\pi}_{\sigma(k,i)}. \tag{23}$$

The average active time and the average sleep time are obtained as follows. For each lumped execution, $N$ tasks are executed on average. The total time ($t_{active} + t_{sleep}$) obviously equals to ($N \times I$).

The average number of executed tasks $N$ is given by

$$N = \sum_{n=0}^{\infty} n \cdot u(n) \tag{24}$$

Here, $u(n)$ is the probability of the lumped execution executes exactly $n$ tasks. To obtain $u(n)$, we start from a probability distribution $u_{dist}(0)$, which represents the probability of the first working mode ($s_\star$) is 1. If $s_{star}$ is $s_1$, $u_{dist}(0)$ is given by

$$u_{dist}(0) = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \end{pmatrix}. \tag{25}$$

Then the followings are given by

$$u_{dist}(1) = (0\ 1\ 0\ \cdots\ 0)\hat{P}$$

$$u_{dist}(2) = (0\ u'_{dist}(1))\hat{P}$$

$$\vdots$$

$$u_{dist}(n) = (0\ u'_{dist}(n-1))\hat{P}$$

Here, $u'_{dist}(n)$ is a $1 \times N$ vector and has from 2nd to $(N+1)$th entries of $u_{dist}(n)$. Then, $u(n)$ is the first entry of $u_{dist}(n)$. When $n \cdot u(n)$ becomes negligibly small, $N$ is obtained.

The average sleep time $t_{sleep}$ is also given by

$$t_{sleep} = \int_{-\infty}^{\infty} t\tilde{r}_0(t)dt - x_{up}^0. \quad (26)$$

Here, $\tilde{r}_0(t)$ is initial distribution of $s_0$.

Now, the average active time $t_{active}$ is obviously given by

$$t_{active} = N \times I - t_{sleep}. \quad (27)$$

Finally, when the scheduling variables $x_{up}$, $x_{down}$ and the first working mode $s_\star$ are given, we can obtain the transition probability matrix $P$ and other parameters, then the objective function $J$ can be obtained.

### 4.4 Obtaining Optimal Scheduling

In the previous section, we explain how to obtain the objective function $J$ from the scheduling variables $x_{up}$, $x_{down}$ and the first working mode $s_\star$. In this section, we explain how to seek for the optimal scheduling. In general, the scheduling variables $x_{up}$ and $x_{down}$ can be real numbers and the number of possible combinations is infinity. To simplify the algorithm, we define a unit time and these scheduling variables are limited to integral multiples of the unit time. Then the problem is N+1-dimensional discrete optimization problem. In this paper, we introduce and compare three search algorithms, *Greedy search, Random search, Uniform search and Genetic algorithm*. We explain these algorithms one by one.

#### 4.4.1 Greedy Search

The greedy search is one of the simplest search algorithms. When a set of scheduling parameters is given, this algorithm tries to modify each parameter to the neighbor value and evaluates it. If new parameters are better than the current parameters, the current parameters are replaced with the new parameters. Then the same algorithm is applied repeatedly. When any neighbor parameters are worse than the current parameters, this search is terminated. Obviously, this algorithm can guarantee the local optimum.

In our evaluation, we always apply this algorithm after other algorithm to guarantee the local optimum.

#### 4.4.2 Random Search

The random search is also one of the simplest search algorithms. First, randomly generate valid $N$ sets of the scheduling parameters. Then evaluate all sets and pick the best set of the parameters. Finally, we apply the greedy search for the best one.

#### 4.4.3 Uniform Search

The uniform search is a simple search algorithm. First, we define the search granularity $\Delta t$. For each scheduling parameters, the values every $\Delta t$ are set to search point. Then all possible combinations of all parameters are evaluated and pick the best set of the parameters. Note that the initial working mode is always searched for every possible pattern. Finally, we apply the greedy search for the best one.

#### 4.4.4 Genetic Algorithm

The genetic algorithm (GA) is a well-known search algorithm and is effective for a huge search space. The GA has two parameters, the number of genes $K$ and the number of maximum generations $L$. One gene corresponds to one set of scheduling variables. Initial $K$ genes are randomly generated.

First of each itaration, expected energy consumption of all genes are evaluated. Top 10% of energy efficient genes are simply copied into the next generation. The rest (90%) is generated by crossing. The crossing randomly pick two genes and mix them. We randomly mix them stage by stage. If invalid gene is generated, we normalize or adjust it randomly. Then, 1% of the genes are mutated. Finally, 1% of the genes are mutated. Namely, we randomly pick one parameter and modify it randomly within valid range.

Then repeat this procedure until the number of the loops reaches to the number of maximum generations $L$. After $L$ generations, the best gene is the answer of the genetic algorithm. Same as the other algorithms, the greedy search is applied to the answer to guarantee the local optimum.

## 5. Evaluation

### 5.1 Evaluation Setup

In this section, we introduce software parameters, hardware parameters and their implementation for the evaluation.

#### 5.1.1 Target Applications

In this evaluation, we use a synthetic application which is shown in Table 5 and 6. We also use an H.264 decoder with three kinds of videos, animation (ani), High-motion (high) and Low-motion (low). Their detailed parameters are borrowed from [21] and are shown in Table 7 and 8. In these videos, there are three type of frames, I, P and B. In each frame type, we assume the execution time follows a normal distribution with a standard deviation of (max-min)/2 between min and max.

#### 5.1.2 Hardware Environment

We use an evaluation board, which equipped with heterogeneous MCUs instead of a DVFS capable processor. To realize equivalent environment with a DVFS processor, the maximum number of active core is limited to one.

We measured energy parameters using the evaluation board. The board is equipped with an RL78 [22] Micro Controller Unit (MCU) and an RX63N [23] MCU, some sensors, a communication unit and an external NVM. Sensors

**Table 5** Evaluation settings (Synthetic task)

| Parameters | Values |
|---|---|
| Number of paths $M$ | 2 |
| Size ratio of paths $R$ (=p(2)/p(1)) | 0.1, 0.5, 0.9 |
| Input interval $I$ | 100 ms |
| Deadline $d$ | 1000 ms |
| Execution Time of path 1 on Mode1 $et(1,1)$ | 360ms |

**Table 6** Task probabilities (Synthetic task)

| Pattern ID | Probabilities p(1) | p(2) | Average task size (relative) $R$=0.1 | $R$=0.5 | $R$=0.9 |
|---|---|---|---|---|---|
| P0 | 0.0 | 1.0 | 1.0 | 5.0 | 9.0 |
| P1 | 0.1 | 0.9 | 1.9 | 5.5 | 9.1 |
| P2 | 0.2 | 0.8 | 2.8 | 6.0 | 9.2 |
| : | : | : | : | : | : |
| P9 | 0.9 | 0.1 | 9.1 | 9.5 | 9.9 |
| P10 | 1.0 | 0.0 | 10.0 | 10.0 | 10.0 |

**Table 7** Evaluation settings (Decode task)

| | |
|---|---|
| Input interval $I$ | 50 ms |
| Deadline $d$ | 250 ms to 1000 ms |

**Table 8** Execution time (Decode task)

| Type | I:P:B | Execution Time on Mode1 [ms] I frame min | I frame max | P frame min | P frame max | B frame min | B frame max |
|---|---|---|---|---|---|---|---|
| ani | 1:2:27 | 127.5 | 198.9 | 15.3 | 198.9 | 20.4 | 178.5 |
| high | 1:1:4 | 76.5 | 127.5 | 25.5 | 127.5 | 20.4 | 76.5 |
| low | 1:4:10 | 76.5 | 102.0 | 35.7 | 66.3 | 25.5 | 40.8 |

**Table 9** Evaluation settings (Hardware)

| mode ID $j$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Relative Performance | 1.0 | 2.0 | 3.0 | 4.0 |
| Power Consumption | | | | |
| in Active $P_d(j)$+$P_{Sa}(j)$[W] | 0.025 | 0.066 | 0.120 | 0.194 |
| in Sleep $P_{Ss}(j)$ [$\mu$W] | 0.69 | 2.07 | 6.20 | 18.6 |

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, Mode1 and 4 are RL78 and RX63N respectively. Mode2 and 3 are generated by interpolating between parameters retrieved from Mode1 and 4. The collected and assumed parameters are shown in Table 9. The energy parameters, which are used in the object function, are calculated from the power and execution time. The performance ratio of these two MCUs is 4. To simplify the evaluation, we also assume the execution times of other tasks are in proportion to their task size. If this assumption is not satisfied, all of the execution times ($et(a,c)$) and the energy consumption ($E_d(a,c)$) must be given particularly.

#### 5.1.3 Implementation

To implement the optimal scheduling on a real system, only a small additional table is required. When a task is finished, the slack time is calculated and compared with the thresholds in the table. If the slack time exceeds the threshold, another mode should be selected. The computation cost of this comparison is negligibly small.

To obtain the optimal scheduling at a reasonable cost, we quantize every time-domain parameters with 0.1ms.

**Table 10** Evaluated patterns of uniform search ($d$=1000ms)

| $\Delta t$ [ms] | # of Patterns |
|---|---|
| 500 | 30 |
| 300 | 226 |
| 200 | 1176 |
| 150 | 8950 |
| 100 | 71807 |
| 50 | 5224186 |
| 1 | $> 1 \times 10^{17}$* |

*: expected

### 5.2 Solution Cost and Search Method

To evaluate the search algorithms, we need to estimate the solution cost. The solution cost is defined by counting the number of evaluations. We assume the total computation time is proportional to the evaluation count. Though, the execution time of each evaluation varies depending on the scheduling parameters and execution time of generating scheduling parameters is negligibly small. Current implementation can evaluate roughly 500 patterns per second with dual Xeon E5-2620 v2 (24 threads).

With the uniform search, as a preliminary study, the relationship between $\Delta t$ and the evaluation count is evaluated and is shown in Table 10. In this evaluation, deadline $d$ is fixed to 1000ms. From this result, if we try brute force searching, i.e. for every 1$\mu$s uniform search, the total number of evaluations is expected to be more than $1 \times 10^{17}$. Due to this enormous search space, the brute force search is practically unacceptable. Therefore, efficient search methods are strongly required.

The results are shown in Fig. 6 and 7. Note that the x-axes represent the solution cost and are in log scale. Random, Uniform and GA represent the random search, the uniform search and genetic algorithm respectively. These results are obtained before applying the greedy search.

From these results, the GA reached the minimum energy consumption after evaluate 2 million patterns. The Uniform also achieved comparable energy efficiency. A drawback of the Uniform is that it does not guarantee monotonic decrease (see 200 and 5M with P1 and 5M with P9) in contrast to others that do. Since the search space is multidimensional, if we half the granularity $\Delta t$, the number of patterns becomes more than 50 times larger as shown in Table 10. Thus if we want to search slightly larger number of patterns, $\Delta t$ should be slightly smaller. As a result, due to the resolution mismatching, the search point are shifted and the result may be worse. The Random is about 6% worse than others with P9.

According to these results, we decide to use the GA with $M = 10000$ and $L = 1000$ is sufficient to get the optimal solution and we use this for further evaluations.

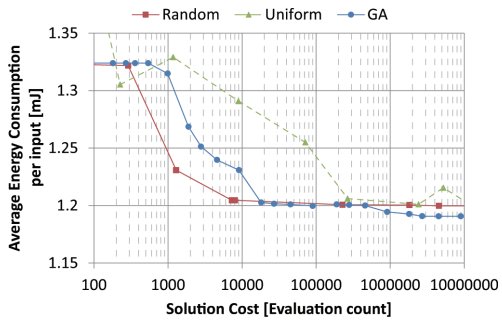



**Fig. 6** Solution cost and quality (Synthetic task, R=0.1, P1)

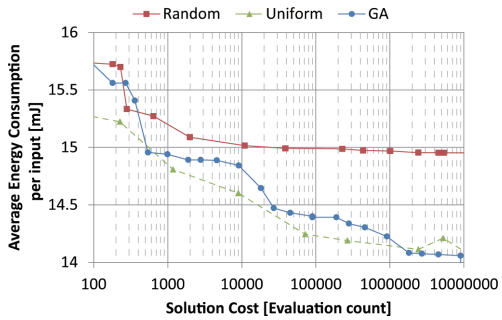



**Fig. 7** Solution cost and quality (Synthetic task, R=0.1, P9)

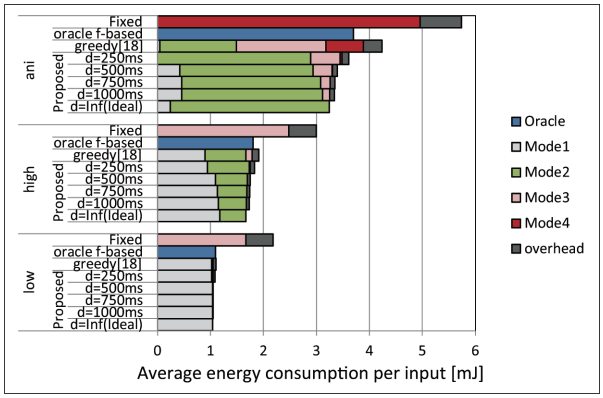



**Fig. 8** Energy breakdown

**Table 11** Obtained schedulings

| Variables | ani | high | low |
|---|---|---|---|
| $x^0_{up}$ [ms] | 249.3 | 248.4 | 190.2 |
| $x^1_{up}$ [ms] | 249.9 | 167.9 | 127.5 |
| $x^2_{up}$ [ms] | 160.7 | 95.9 | 76.5 |
| $x^3_{up}$ [ms] | 116.0 | 74.6 | 72.0 |
| $x^0_{down}$ [ms] | 249.9 | 249.9 | 249.9 |
| $x^1_{down}$ [ms] | 249.9 | 243.8 | 166.6 |
| $x^2_{down}$ [ms] | 211.9 | 117.5 | 108.4 |
| $x^3_{down}$ [ms] | 133.4 | 74.6 | 92.9 |
| ★ | 1 | 0 | 0 |

### 5.3 Validation

To validate our estimation of the average energy consumption based on statistical analysis, we compare the estimated value with the measured value on the evaluation board. We use the synthetic task and only use Mode1 and Mode4, which are actually equipped on the board, and implement the application with proposed slack-based scheduler. The optimal scheduling, which are sought by GA, is set.

The average energy consumption is calculated from the total energy consumption during 90 seconds and the number of executed tasks. In this evaluation, $d$ is fixed to 1000ms. As a result, we confirmed that our estimation is very accurate and even the largest error is less than 6%.

### 5.4 Energy Efficiency

To clarify the merit of our task scheduling algorithm, we evaluate the average energy consumption.

For comparison, we also calculate the energy consumption of fixed scheduling (Fixed), which always assume the largest task and use the highest performance mode, and oracle frame-based scheduling (Oracle f-based), which assumes *perfect* execution time prediction is possible. We also calculate that of ideal scheduling (Ideal), which equals to the minimum energy consumption to achieve required throughput.

Figure 8 shows a comparison of the energy consumption and the energy breakdown, when deadline period $d$ varies from 250ms to 1000ms for proposed and fixed to 250ms for greedy. In Fig. 8, overhead is only calculated for Proposed and greedy because other results are based on abstracted energy models.

Table 11 shows obtained scheduling when deadline period is 250ms. For heavy application, such as ani, every $x^i_{up}$ and $x^i_{down}$ are set to large values to keep high performance core active and minimize core switching overhead. For light application, such as low, these variables are set to small values to keep energy efficient core active and minimize core switching overhead.

In this evaluation, Fixed always assumes WCET and executes all tasks on Mode4(ani) or Mode3(high and low). Oracle f-based always executed on the desired frequency, regardless of four modes, based on perfect execution time prediction. In contrast, Proposed slack-based task scheduling adaptively selects from all of MCUs. Additionally, Ideal scheduling only uses adequate Modes. greedy [18] always select as low performance as possible mode and uses all of MCUs. However, this stratagy is too agressive and cannot minimize utilize ratio of the high performance mode. Additionally, overhead is larger than our scheduling due to frequent mode switching.

This result shows proposed scheduling can achieve up to 9.8% lower power consumption than Oracle f-based, which is a theoretical lower bound of frame-based schedulings. The longer the deadline period, the smaller the energy consumption. When the deadline period is 1000ms, the improvement of the energy efficiency is almost saturated and only 3.6% larger energy consumption than that of the ideal scheduling can be observed.

## 6. Conclusion

Near real-time data processing, such as multimedia streaming applications, has a deadline period that is longer than the input interval and the tasks have dynamic behaviors such as input-dependent variations. In this situation, energy efficient task scheduling is important, especially for meeting the deadline strictly.

To cope with this challenge, we proposed a slack-based task scheduling. This scheduling is carried out by comparing the slack time to several thresholds and the scheduler then throttles core performance when these thresholds are overpassed. These thresholds are obtained from hardware and task parameters at design time. To obtain optimal thresholds, we formulate the scheduling with an FSM. Then, the average energy consumption is derived from statistical analysis. Finally, the optimal scheduling is sought by GA.

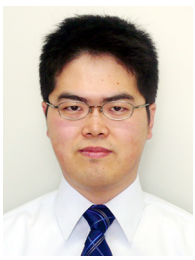
We confirmed that our approach can reduce the average energy consumption by up to 9.8% compared to *oracle* frame-based execution, when the deadline period is 20 times longer than the input interval. We conclude that our slack-based scheduling can drastically reduce the energy consumption of embedded 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 JP16K12405.

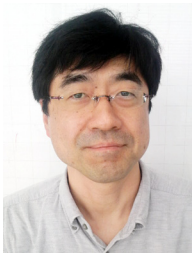
### References

[1] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," Proc. 1st USENIX Conference on Operating Systems Design and Implementation, pp.13–23, 1994.
[2] 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.
[3] T.D. Burd and R.W. Brodersen, "Energy efficient cmos microprocessor design," Proc. 28th Hawaii International Conference on System Sciences, pp.288–297, 1995.
[4] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," IEEE Trans. Very Lagre Scale Integr. (VLSI) Syst., vol.8, no.3, pp.299–316, 2000.
[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] 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.
[7] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo, "Multiprocessor energy-efficient scheduling with task migration considerations," 16th Euromicro Conference on Real-Time Systems, pp.101–108, 2004.
[8] C. Xian, Y.-H. Lu, and Z. Li, "Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time," 44th ACM/IEEE Design Automation Conference, pp.664–669, 2007.
[9] R. Xu, R. Melhem, and D. Mossé, "A unified practical approach to stochastic dvs scheduling," Proc. 7th ACM & IEEE International Conference on Embedded Software, pp.37–46, 2007.
[10] T. Zitterell and C. Scholl, "A probabilistic and energy-efficient scheduling approach for online application in real-time systems," Proc. 47th Design Automation Conference, pp.42–47, 2010.
[11] J. Cong and K. Gururaj, "Energy efficient multiprocessor task scheduling under input-dependent variation," Design, Automation Test in Europe Conference Exhibition, pp.411–416, 2009.
[12] M. Qiu, C. Xue, Z. Shao, and E.H.-M. Sha, "Energy minimization with soft real-time and dvs for uniprocessor and multiprocessor embedded systems," Proc. Conference on Design, Automation and Test in Europe, pp.1641–1646, 2007.
[13] D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," Proc. 2003 International Symposium on Low Power Electronics and Design, pp.408–413, 2003.
[14] M. Lombardi, M. Milano, M. Ruggiero, and L. Benini, "Stochastic allocation and scheduling for conditional task graphs in multiprocessor systems-on-chip," Journal of Scheduling, vol.13, no.4, pp.315–345, 2010.
[15] T. Nakada, H. Yanagihashi, H. Ueki, T. Tsuchiya, M. Hayashikoshi, and H. Nakamura, "Energy-efficient continuous task scheduling for near real-time periodic tasks," The 8th IEEE International Conference on Internet of Things (iThings), pp.675–681, Dec. 2015.
[16] 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.
[17] S. Durand, A.-M. Alt, D. Simon, and N. Marchand, "Energy-aware feedback control for a h.264 video decoder," Int. J. Syst. Sci., vol.46, no.8, pp.1432–1446, 2015.
[18] C. Im, S. Ha, and H. Kim, "Dynamic voltage scheduling with buffers in low-power multimedia applications," ACM Trans. Embed. Comput. Syst., vol.3, no.4, pp.686–705, Nov. 2004.
[19] T. Nakada, T. Hatanaka, H. Ueki, M. Hayashikoshi, T. Shimizu, and H. Nakamura, "An adaptive energy-efficient task scheduling under execution time variation based on statistical analysis," IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp.1–7, Sept. 2016.
[20] 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 Online Transactions, vol.7, no.3, pp.37–46, 2014.
[21] B. Lee, E. Nurvitadhi, R. Dixit, C. Yu, and M. Kim, "Dynamic voltage scaling techniques for power efficient video decoding," J. Syst. Architect., vol.51, no.10-11, pp.633–652, 2005.
[22] Renesas Electronics Corporation, "RL78 Family." http://japan.renesas.com/products/mpumcu/rl78/index.jsp.
[23] Renesas Electronics Corporation, "RX63N, RX631." http://japan.renesas.com/products/mpumcu/rx/rx600/rx63n_631/index.jsp.



**Takashi Nakada** received his M.E. and Ph.D. degrees 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 and ACM.

**Tomoki Hatanaka** 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.



**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. He is now section manager of power module design of System Integration Business Division in Renesas Electronics Corporation.

**Masanori Hayashikoshi** received his B.S. and M.S. degrees in electronic engineering from Kobe University in 1984 and 1986 respectively. In 1986, he joined the LSI Research and Development Laboratory, Mitsubishi Electric Corporation. He is currently a Chief Professional of Core Technology Business Division in Renesas Electronics Corporation. Since 1986, he has been engaged in the research and development of EEPROM's, high density DRAM's, Low power SDRAM's, embedded MRAM's for MCUs, and Normally-Off computing architecture as the challenge for further low-power solution with NVRAM.

**Toru Shimizu** received his Ph.D. degree of Information Science from The University of Tokyo in 1986. Since 1986, he has been involved in microprocessor and micro-controller R&D projects in Mitsubishi Electric and Renesas Electronics. His R&D works include a RISC microprocessor integrated with a large DRAM and a single-chip symmetric multi-core microprocessor with an on-chip shared memory. He is a professor in Keio University from 2014. His research interest covers not only LSI architecture and its design but system and software design based on LSIs. He is an IEEE Fellow.

**Hiroshi Nakamura** is a Professor in the Department of Information Physics and Computing at The University of Tokyo. He is also the director of Information Technology Center 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.