Crenel-Interval-Based Dynamic Power Management for Periodic Real-Time Systems

GUOHUI LI, YI ZHANG, and JIANJUN LI, Huazhong University of Science and Technology

In order to save the energy consumption of real-time embedded systems, the integration of Dynamic Voltage and Frequency Scaling (DVFS) and Device Power Management (DPM) techniques has been well studied. In this article, we propose a new energy management scheme for periodic real-time tasks with implicit deadlines. We mainly focus on the DPM part by presenting a novel approach to the real-time DPM problem. Specifically, we first identify intervals for each device, which we refer to as Crenel Intervals, by partitioning the Earliest Deadline First (EDF) schedule of the tasks that need to access the device into successive intervals. The principle for identifying Crenel Intervals is that for each task, there is only one deadline located in each Crenel Interval. Next, targeting at a single device model and a multiple device model, respectively, we propose the CI-EDF and CI-EDF^m algorithms to schedule task instances in each Crenel Interval, so as to form long and continuous slacks in each Crenel Interval but without jeopardizing any task deadlines. Then, the slack in the Crenel Intervals can be utilized to perform not only DPM, but also DVFS. The experimental results show that our approaches can achieve considerably more energy savings than existing techniques with comparable quality.

Categories and Subject Descriptors: C.2.2 [**Embedded Real-Time Systems**]: Energy Management

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Embedded real-time systems, dynamic power management, Crenel Interval, periodic tasks

ACM Reference Format:
Guohui Li, Yi Zhang, and Jianjun Li. 2015. Crenel-Interval-based dynamic power management for periodic real-time systems. ACM Trans. Embed. Comput. Syst. 14, 4, Article 74 (September 2015), 32 pages.
DOI: http://dx.doi.org/10.1145/2744197

1. INTRODUCTION

With the ever increasing use of mobile phones and the Internet, mobile and embedded applications become more and more popular nowadays. Since these devices are usually battery equipped, how to reduce the energy consumption, and thus prolong the battery life has become an important issue when designing such applications, especially when the devices work in a harsh environment where power recharging is very difficult or even impossible. Two promising techniques, Dynamic Voltage and Frequency Scaling (DVFS) [Weiser et al. 1996] and Device Power Management (DPM) [Lu et al. 2002], have been widely studied and a lot of research efforts have been reported. DVFS is a commonly used power-management technique, which allows the voltage and the clock

This work was substantially supported by the State Key Program of National Natural Science of China under Grant No. 61332001, National Natural Science Foundation of China under Grants No. 61173049 and No. 61300045, and China Postdoctoral Science Foundation under Grant No. 2013M531696.
Authors' addresses: G. Li, Y. Zhang, and J. Li (corresponding author), School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China; emails: guohuili@hust.edu.cn, zhangyihust@gmail.com, jianjunli@hust.edu.cn.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2015 ACM 1539-9087/2015/09-ART74 $15.00
DOI: http://dx.doi.org/10.1145/2744197

frequency to be decreased, to trade time for energy [Aydin et al. 2004; Pillai and Shin 2001; Qadi et al. 2003; Saewong and Rajkumar 2003; Le Sueur and Heiser 2010]. Another commonly used technique, called DPM, is feasible due to the provision of the Advanced Configuration and Power Interface (ACPI). With DPM, off-chip devices can be put into low-power (sleep) states when their idle time is longer than a given threshold [Cheng and Goddard 2006a; Swaminathan and Chakrabarty 2005], which is usually defined as break-even time [Devadas and Aydin 2012], and represents the minimum inactivity time required to compensate for the cost of entering and exiting the idle state.

In this article, we study how to minimize the system-level energy consumption for periodic real-time tasks with implicit deadlines. Our primary objective in this work is to design a novel and efficient online DPM scheme, in which the DPM effectiveness can be maximized. It has been widely recognized that in order to achieve the system-level energy minimization, DVFS and DPM should be considered in a subtle way. In fact, in the past few years, the integration of DVFS and DPM has been well studied and a lot of research effort has also been reported. Some examples are SYS-EDF [Cheng and Goddard 2005], DFR-RMS [Devadas and Aydin 2008b], and DFR-EDF [Devadas and Aydin 2010]. Nevertheless, the DPM components in almost all these approaches are implemented based on the stochastic and predictive techniques. The representative DPM-only scheme EEDS [Cheng and Goddard 2006b] is also a predication-based one. The principle behind these DPM schemes is to defer task execution as much as possible but without jeopardizing the schedulability of the task set to form big slacks for performing device state transition. Though effective, these DPM schemes did not fully explore the relationship between the maximum possible idle interval for a device and the tasks that need to access that device, and thus still have much room for improvement in terms of energy savings. To our best knowledge, there is no work relying on the relationship mentioned previously to design DPM schemes in the literature up to now. In this work, we make the first attempt to design new DPM schemes by characterizing such kind of relationship, and relying on it to guide the schedule of the task instances at runtime.

Contributions of this research. In this article, we develop new DPM schemes by identifying the maximum possible idle interval for devices. Since we focus on minimizing the system-level energy consumption, we assume a general energy model where the CPU and device power consumptions, as well as the device break-even time and transition overheads are considered. A critical building block for our DPM schemes is the introduction of Crenel Interval (CI), in which we merge all the small slacks to form a big one by scheduling the workloads residing in the CI to the two sides of it. We call such an Interval CI since it has a Crenel-like shape. Since a device's slack time is decided by the tasks that need to access it, in this article, we first show how to identify Crenel Interval for each device, to ensure that for each task needed to access that device, there is only one deadline in each Crenel Interval. In this way, CI indicates the maximum duration that the device can be in idle state.

Then, targeting at a single device model where all the tasks access the same and only device, we propose a scheduling algorithm, namely, CI-EDF, to schedule the task instances in each Crenel Interval to guarantee the schedulability of the task set. CI-EDF has a time complexity of $O(n^2)$ (n is the number of the tasks in the system), and thus is quite efficient as an online scheme. Since a long and continuous slack is reserved in each CI, CI-EDF exhibits good performance in energy savings.

Next, targeting at a more general multiple device model where a task may access multiple devices and a device may also be accessed by multiple tasks, we propose another scheduling algorithm, namely, CI-EDF^m, which has a time complexity of $O(nm)$ (m is the number of the devices in the system). In CI-EDF^m, we compute Device Crenel

Intervals (DCIs) for each device, based on the properties of the tasks that need to access the device. Considering that a task instance may need to access multiple devices and thus be located in multiple DCIs, we design a weighting-factor-based strategy to help determine whether the task instance should be delayed or not, with the objective of minimizing the energy consumption.

Since our objective is to minimize the system-level energy consumption, we show that DVFS techniques can also be integrated easily with our DPM schemes to further reduce the energy consumption. Moreover, considering that the task actual execution time is usually less than its Worst-Case Execution Time (WCET), we further briefly introduce a dynamic slack reclamation scheme for more energy savings.

Finally, we evaluate our schemes versus some existing algorithms with comparable quality over a wide spectrum of system/application parameters. The experimental evaluation with realistic processor and device specifications indicates that our schemes can result in significant energy saving compared to the state of the art. The experimental evaluation of various system profiles also shows that our schemes maintain a robust performance.

Organization. We organize the remainder of this article as follows. Section 2 reviews related work. Section 3 describes our processor, task and power models, along with some assumptions we make. Section 4 introduces the concept of Crenel Interval and describes how to identify Crenel Intervals. Section 5 details the CI-EDF and CI-EDF^m algorithms, and briefly introduces the integration of DVFS with the proposed DPM schemes, as well as the dynamic slack reclaiming policy. Section 6 presents and discusses our experimental results, and finally, Section 7 draws a conclusion.

2. RELATED WORK

DVFS plays an important role in saving processor energy consumption, which has become a critical problem in embedded and mobile systems with real-time constraint in recent years. With DVFS, the processor can execute at different voltage and frequency levels. Since the CPU power consumption increases in a convex fashion with the frequency, DVFS helps to significantly reduce the CPU dynamic energy consumption. The problem of minimizing the energy consumption while satisfying the timing constraints has been extensively studied in recent past for various task/system models. Specifically, researchers have tackled the energy reduction problem for both periodic and aperiodic real-time tasks [Aydin et al. 2001; Quan and Hu 2003; Saewong and Rajkumar 2003; Yao et al. 1995], as well as tasks with critical sections [Jejurikar and Gupta 2006; Zhang and Chanson 2004] or fully nonpreemptive tasks [Jejurikar and Gupta 2005b; Li et al. 2013], scheduling using a hybrid of the slowdown and shutdown strategies [Lee et al. 2003; Quan et al. 2004] energy reduction based on slack reclamation [Quan and Hu 2003; Jejurikar and Gupta 2005a; Pillai and Shin 2001], energy-aware scheduling with reliability requirements [Zhang and Chakrabarty 2004; Zhu and Aydin 2009; Zhu et al. 2004], multiprocessor energy-efficient scheduling [AlEnawy and Aydin 2005; Chen et al. 2008], temperature-aware scheduling [Fisher et al. 2009; Chen et al. 2009], and energy-aware scheduling at a broader system level [Aydin et al. 2006; Zhuo and Chakrabarti 2005].

DPM is another commonly used energy management technique, aiming at reducing device energy consumption. With DPM, reserving long sleep interval is the key factor for effective power management of devices. In Swaminathan et al. [2001] and Swaminathan and Chakrabarty [2003], Swaminathan and Chakrabarty presented a heuristic-based DPM algorithm called Low Energy Device Scheduler (LEDES). But LEDES is under the constraints that the start times of the tasks are fixed and cannot be changed at runtime. In Swaminathan and Chakrabarty [2005], Swaminathan and Chakrabarty proposed an offline algorithm, called Maximum Device Overlaps (MDO),

to generate near-optimal solutions in polynomial time. MDO involves relatively high time complexity and cannot be adapted to the case where job release and execution times may vary considerably. In Cheng and Goddard [2006a, 2006b], Cheng and Goddard presented an Energy Efficient Device Scheduling (EEDS) framework, which is based on exploiting task and device slacks to create long idle intervals, in preemptive and nonpreemptive environments. In Devadas and Aydin [2008b], Devadas and Aydin proposed the notation of Device Forbidden Regions (DFRs) by explicitly and periodically enforcing intervals of DFRs for each device at runtime, without causing any deadline missing. By explicitly and periodically enforcing such DFRs for each device at runtime, DFR-RMS can achieve significant energy savings. In Awan and Petters [2012], Awan and Petters first explored online intratask device scheduling for hard real-time systems, by proposing a Static Slack Container (SSC) algorithm based on the model where each device is associated with exactly one task. SSC can get significant energy savings, and at the same time, has a very low time complexity $O(n)$, which is lower than existing RT-DPM algorithms.

Some works have also tried to integrate the DPM and DVFS to save system-level energy consumption. For the frame-based task systems, Devadas and Aydin explored in Devadas and Aydin [2008a, 2012] the interplay of DVFS and DPM, and proposed a provably optimal algorithm to determine the optimal CPU frequency as well as device state transition decisions to minimize the system-level energy. In Kong et al. [2010], Kong et al. developed optimization algorithms based on 0-1 integer nonlinear programming for different system configurations. In Gerards and Kuper [2013], Gerards and Kuper presented a schedule method for a frame-based system that globally minimizes the energy consumption for DPM and the combination of DPM and DVFS where the interplay between DPM and DVFS is taken into account. All the previously mentioned works are targeted at frame-based systems, and are not suitable for general periodic task sets. The two typical works for periodic task systems are SYS-EDF and DFR-EDF. Based on the general periodic task model, Cheng and Goddard [2005] proposed a practical system-level energy management heuristic method called SYS-EDF. The DVFS component of SYS-EDF is based on the concept of energy efficient scaling, while the DPM component utilizes the next device usage time predications. In Devadas and Aydin [2010], Devadas and Aydin extended their DFR approach to Earliets Deadline First (EDF) scheduling (DFR-EDF). Before adding a new forbidden region, DFR-EDF considers the expected change in system energy consumption to obtain the maximum system energy savings. DFR-EDF establishes the relationship between DPM and DVFS and can get remarkable energy savings.

3. MODELS AND ASSUMPTIONS

3.1. Processor and Task Model

We consider a DVS-capable uniprocessor where the operating frequency can be scaled within the discrete range $[f_{\min}, f_{\max}]$. It is now common knowledge that changing from one frequency level to another takes a fixed amount of time (ranges from tens of microseconds to tens of milliseconds), referred to as the transition (or switch) overhead [Mochocki et al. 2007]. In this work, we use Δt to denote the time overhead of each frequency transition. The frequency that minimizes the processor energy consumption per cycle is called the *critical* speed when considering both dynamic and static energy consumption [Jejurikar et al. 2004]. Since executing below the *critical* speed consumes more time and energy, $f_{\min}$ is set to be equal to the *critical* speed. For convenience, we normalize the frequency value with respect to $f_{\max}$, that is, $f_{\max} = 1$.

The system workload consists of a set of independent periodic tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$. Each task τ_i is denoted by a tuple (C_i, D_i, T_i), where C_i denotes the WCET of τ_i at

the maximum processor frequency $f_{\max}$, and D_i and T_i denote the relative deadline and period of τ_i, respectively. The WCET of τ_i under frequency f_{max} is denoted by $C_i = x_i + y_i$, where x_i indicates the frequency-dependent on-chip workload, which scales linearly with frequency, and y_i represents the frequency-independent off-chip workload, which does not scale with frequency. Thus, at frequency f, the WCET of τ_i is $C_i(f) = \frac{x_i}{f} + y_i$. For each τ_i, we assume $D_i = T_i$. The j^{th} instance of τ_i is denoted by $\tau_{i,j}$, and $\tau_{i,j}$'s deadline is denoted by $d_{i,j}$. $\mathcal{T}$ is sorted in ascending order of period, that is, if $i < j$, then $T_i \leq T_j$. The first instances of all tasks are assumed to release at time 0. At any time, task instances eligible to execute are scheduled by the preemptive EDF scheduling policy, and the priority of instance $\tau_{i,j}$ is denoted by $Pr(\tau_{i,j})$.

3.2. Device Model

We assume there are totally m devices $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m\}$ in the system and each device has at least two states: an *active* state and a *sleep* (low-power) state. The devices cannot be turned off completely, but can transition from *active* to *sleep*. All devices needed by the real-time tasks must be in active state until no task is in execution. This assumption is reasonable given that the device state transitions typically involve nontrivial costs and it is fairly difficult to predict when a running application will re-request a specific device during execution [Cheng and Goddard 2006a; Swaminathan and Chakrabarty 2005; Devadas and Aydin 2012]. For a device, we use *device slack* to denote the time interval during which no task uses this device.

Similar to Devadas and Aydin [2012], we use P_a^i and P_s^i to denote the power consumption of $\mathcal{D}_i$ in *active* and *sleep* states, respectively. The energy overhead and time cost during the device state transition cannot be ignored; we use E_{sw}^i to denote the energy overhead of transitioning $\mathcal{D}_i$ from *active* to *sleep* and back from *sleep* to *active*. Similarly, T_{sw}^i indicates the total transition time consumed between *active* and *sleep* states. Due to the constraints imposed by device transition delays, $\mathcal{D}_i$ cannot be transitioned between *active* and *sleep* states over an interval length smaller than T_{sw}^i. Moreover, in order to guarantee that the device transition is energy efficient (as compared to keeping $\mathcal{D}_i$ continuously in *active* state), the minimum length of idle interval over which transitioning a device should be no less than $\frac{E_{sw}^i - P_s^i \cdot T_{sw}^i}{P_a^i - P_s^i}$. Thus, the *break-even time* B_i of $\mathcal{D}_i$ can be computed by $B_i = \max(T_{sw}^i, \frac{E_{sw}^i - P_s^i \cdot T_{sw}^i}{P_a^i - P_s^i})$ [Cheng and Goddard 2005, 2006a].

3.3. Energy Model

We mainly consider two parts of the total system energy consumption. The first part is the on-chip cost, which is mainly consumed by CPU, and the other part is the off-chip cost, which does not scale with the CPU frequency and is mainly consumed by memory and I/O devices. Thus, the total energy consumption E_{tot} can be divided into CPU energy and device energy, that is,

$$E_{tot} = E_{cpu} + \sum_{i=1}^{m} E_{device}^i. \tag{1}$$

In a CMOS circuit processor, there are two major sources of power consumption [Jejurikar et al. 2004]: (1) Dynamic power consumption P_{ac}, which mainly results from the charging and discharging of gates on the circuits. P_{ac} is generally modeled as a convex function with respect to the frequency: $P_{ac}(f) = C_{ef} f^3$, where C_{ef} is the effective switching capacitance. (2) Static power consumption P_{dc}, which mainly results from leakage current and does not scale with frequency. In addition to the dynamic and static power consumption, there is an inherent power cost in keeping the processor on, which is denoted by P_{on}. Considering the preceding three components, the total

processor power consumption P_{cpu} is given by

$$P_{cpu}(f) = P_{ac}(f) + P_{dc} + P_{on.} \tag{2}$$

The processor energy consumption E_{cpu} can be computed by $E_{cpu} = P_{cpu}(f) \cdot \frac{C}{f}$.

The device energy E^i_{device} of $\mathcal{D}_i$ consists of three parts. The first part is the energy consumed by device state transition, that is, the energy required to transmit a device from *active* to *sleep* state and then backwards. The second part is the energy consumed by a device in the *sleep* state. The third part is energy consumption by a device in the *active* state, and this part scales with the execution time C.

4. IDENTIFYING CRENEL INTERVALS

Periodic task scheduling may generate some small slacks, which are not long enough to make an energy-efficient state transition for devices. Even in certain cases, some of these slacks can be used to transition devices into low-power state; the time durations the devices stay in sleep state are usually very short, which makes the energy savings not significant.

In this article, we introduce a novel approach to the online real-time DPM problem. Considering that creating long device sleep intervals is the key for effective power management, we explicitly create long intervals for the devices at runtime. These intervals, called Crenel Intervals (CIs), each consist of three parts in order. In the first and third parts, all the applications are in execution and hence, all the devices are in the *active* state. In the second (middle) part, no task instance is in execution. Therefore, if the length of this part is longer than the break-even times, the corresponding devices can be put into *sleep* state to save energy. In our method, we first identify all the Crenel Intervals. Next, for each CI, we schedule the workloads residing in the CI to both sides of it to satisfy their time constraints, so as to form a long and continuous slack in the middle. In this section, we first introduce how to identify all the Crenel Intervals; scheduling of the task instances in each single Crenel Interval is left to Section 5.

4.1. Motivation Example

We first give an example to illustrate the motivation of our method.

Example 1. Consider the following three periodic tasks specified with their worst-case execution times, relative deadlines, and periods: $\tau_1 = (10, 40, 40)$, $\tau_2 = (10, 60, 60)$, $\tau_3 = (10, 80, 80)$ (all in *ms*). Device $\mathcal{D}_1$ is used by all three tasks, and the device specification comes from Cheng and Goddard [2006a]: Fujitsu 2300AT Hard disk with break-even time $B_1 = 40ms$.

Figure 1(a) depicts the EDF schedule of the task set during the first hyperperiod (240). It can be seen that from time instant $t = 0$ to 240, there are seven slacks, that is, $S_1 = S_2 = S_3 = S_6 = 10$, $S_4 = S_5 = 20$, and $S_7 = 30$. Note that in this example, no device state transition operation can be performed with DFR-EDF [Devadas and Aydin 2010]. This is because in DFR-EDF, each forbidden region is characterized by a length (or duration, denoted by Δ) that must be no shorter than the device break-even time and no longer than the maximum laxity of a task set. Clearly, in this example, the maximum laxity is 30, and the corresponding break-even time is 40, so we cannot find a proper Δ to satisfy the condition.

Figure 1(b) shows the schedule with our Crenel-Interval-based scheduling method for the same task set. In this schedule, the time interval $(0, 240)$ is divided into three CIs, which are $(0, 80)$, $(80, 160)$, and $(160, 240)$, respectively. There are three slacks, $S_{10} = 30$, $S_{11} = 40$, and $S_{12} = 40$, each of which resides in one CI. These three slacks are derived by merging the small slacks in each Crenel Interval. For example, S_{10} is obtained by merging S_1, S_2, and S_3 in Figure 1(a). It is clear to see that the latter two

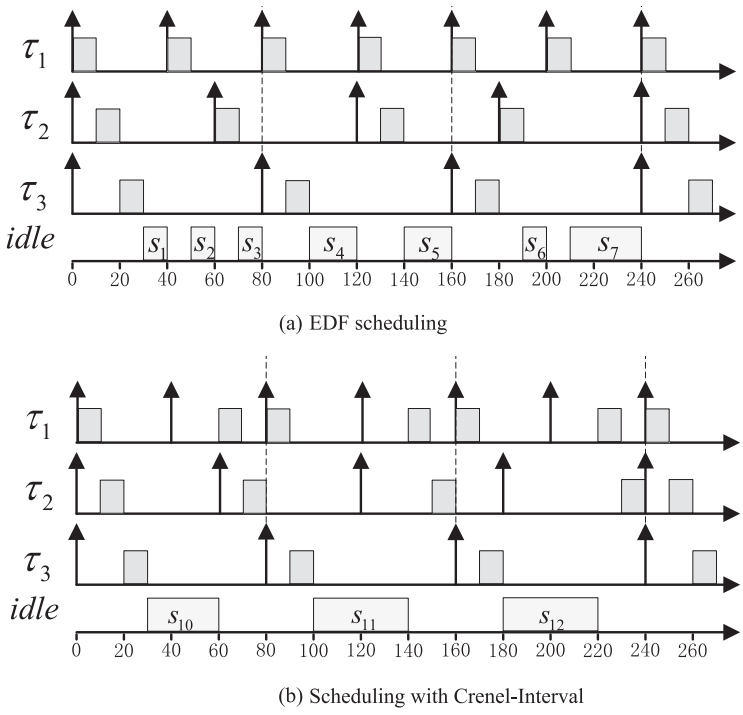


Fig. 1. Motivating example for CI-based DPM.

slacks (S_{11} and S_{12}) can be used to put the device into sleep state since they are no smaller than B_1. As can be seen from Figure 1(b), for the given task set, our Crenel-Interval-based scheduling method can help create long slacks but without jeopardizing the schedulability of the task set (the task set is schedulable by our method in the first hyperperiod).

4.2. Identifying Crenel Intervals

In the previous example, the three intervals $(0, 80)$, $(80, 160)$, and $(160, 240)$ are called Crenel Intervals, and the two end points of each Crenel Interval are called Crenel Points. For presentation convenience, we use CI_j to denote the j-th Crenel Interval, and use t_{j-1} and t_j to denote the start and end Crenel Points of CI_j, respectively. For example, $t_0 = 0$ and $t_1 = 80$ are the two Crenel Points of CI_1. From here on, we use CI_j and (t_{j-1}, t_j) interchangeably when there is no ambiguity.

While the potential benefits of Crenel Interval for enhancing the effectiveness of DPM are clear, it is imperative to make sure how to stipulate the Crenel Intervals and how to schedule the task instances in each Crenel Interval so that they would not miss deadlines. We start by partitioning the whole schedule into successive Crenel Intervals. Since each Crenel Interval is determined by two Crenel Points, it is clear that deriving the Crenel Intervals is essentially the same as computing the following Crenel Points (start with $t_0 = 0$) along the time line.

Now given a Crenel Point t_{j-1}, we show how to find the next Crenel Point t_j, so that (t_{j-1}, t_j) can form a Crenel Interval. As argued before, it is wise to have a large slack time so that the devices can be turned into sleep state, or remain in sleep state for a longer time to save more energy. Intuitively, in a given time period, the smaller the number of slacks, the bigger the size of the slacks. Motivated by this fact, when

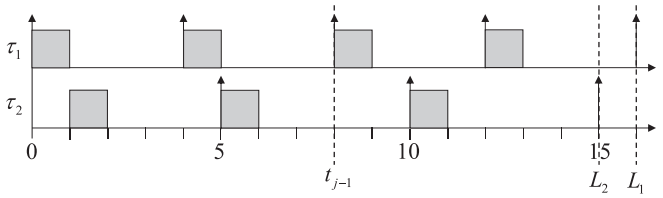


Fig. 2. Example of identifying Crenel Point.

stipulating the Crenel Interval, we try to ensure that there is only one slack in it. To achieve this goal, we need to ensure that for any task $\tau_i\,(1 \le i \le n)$, there are at most two execution instances during a given Crenel Interval. If this premise holds, we can schedule the two instances (if they exist) wisely in such a way that one instance is executed at the beginning, and the other one is executed at the end of the Crenel Interval. Then, in the middle of the Crenel Interval, a continuous slack interval can be reserved. So now the problem is based on t_{j-1}, how to set t_j judiciously to make sure that in interval (t_{j-1}, t_j), each task $\tau_i\,(1 \le i \le n)$ has at most two instances. Observation 1 gives us a hint on how to derive such a t_j.

OBSERVATION 1. *For any task $\tau_i\,(1 \le i \le n)$, if there is at most one deadline of τ_i in interval (t_{j-1}, t_j), that is, there is at most one instance $\tau_{i,k}$ with deadline $(k+1)T_i$ satisfying $t_{j-1} < (k+1)T_i < t_j$, then τ_i has at most two execution instances in CI_j.*

The preceding observation can be explained by contradiction. Suppose that during the Crenel Interval (t_{j-1}, t_j), τ_i needs to meet two deadlines, say $(k+1)T_i$ and $(k+2)T_i$, where $t_{j-1} < (k+1)T_i < (k+2)T_i < t_j$. Then in interval (t_{j-1}, t_j), three task instances, $\tau_{i,k}$, $\tau_{i,k+1}$, and $\tau_{i,k+2}$, may need to be executed, and this contradicts with the requirement that in a Crenel Interval, there are at most two execution instances for each task τ_i.

Based on Observation 1, we give an example to illustrate how to calculate the next Crenel Point based on the current one. Figure 2 depicts the EDF schedule of a task set with two tasks τ_1 and τ_2, where $C_1 = C_2 = 1$, $T_1 = 4$, and $T_2 = 5$. Suppose the first Critical Point is $t_{j-1} = 8$, and now we need to derive the next Critical Point t_j. For τ_1, $L_1 = 16$ is the largest value so that there is at most one deadline in interval (t_{j-1}, L_1). We call $L_1 = 16$ a candidate Crenel Point. For τ_2, $L_2 = 15$ is the largest value so that there is at most one task instance deadline in interval (t_{j-1}, L_2) and $L_2 = 15$ is also called a candidate Crenel Point. From these two candidate Crenel Points, we choose the minimum one, that is, $L = \min\{L_1, L_2\} = L_2$, as the next Crenel Point t_j. Obviously, L can guarantee that for both τ_1 and τ_2, there are at most two instances in (t_{j-1}, L).

We now formally show how to derive the next Crenel Point for the general case. Without loss of generality, suppose t_{j-1} is a given Crenel Point, the next Crenel Point t_j can be calculated as follows. For a task τ_i, we use α_i to denote the quantity relations between t_{j-1} and T_i, that is, $\alpha_i = \frac{t_{j-1}}{T_i}$. If α_i is an integer, then the next candidate Crenel Point, t_j^i, is $\alpha_i T_i + 2T_i$. If α_i is not an integer, then the next candidate Crenel Point, t_j^i, is $\lceil \alpha_i \rceil T_i + T_i$. In summary, we have

$$t_j^i = \begin{cases} \alpha_i T_i + 2T_i & \alpha_i \text{ is an integer} \\ \lceil \alpha_i \rceil T_i + T_i & \text{otherwise.} \end{cases} \tag{3}$$

Since when α_i is not an integer, there is $\lceil \alpha_i \rceil = \lfloor \alpha_i \rfloor + 1$, the preceding two different cases can be described in a uniform form, that is, t_j^i can be calculated as follows:

$$t_j^i = \lfloor \alpha_i \rfloor T_i + 2T_i = \left\lfloor \frac{t_{j-1}}{T_i} \right\rfloor T_i + 2T_i. \tag{4}$$

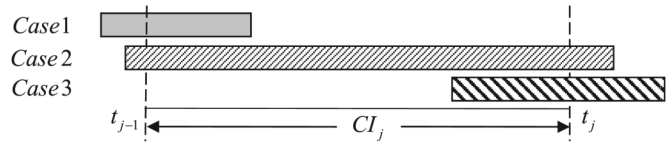


Fig. 3. Classification of task instances in a Crenel Interval.

We now prove the Crenel Point t_j^i computed by the preceding equation can guarantee that there is only one deadline for τ_i in (t_{j-1}, t_j^i).

LEMMA 1. *For a task τ_i, given a Crenel Point t_{j-1}, if t_j^i is calculated by Equation (4), then there is only one deadline in (t_{j-1}, t_j^i).*

PROOF. If α_i is an integer, then $\alpha_i T_i$ is a deadline, and $\alpha_i T_i + 2T_i$ is the second deadline after $\alpha_i T_i$, so in interval $(t_{j-1}, \alpha_i T_i + 2T_i)$, there is only one deadline, which is $\alpha_i T_i + T_i$. If α_i is not an integer, then $\lceil \alpha_i \rceil T_i$ is the first deadline from time point t_{j-1}, so in interval $(t_{j-1}, \lceil \alpha_i \rceil T_i + T_i)$, there is only one deadline $\lceil \alpha_i \rceil T_i$. □

Based on Lemma 1 and Observation 1, we can conclude that for τ_i, there are at most two instances of execution during the Crenel Interval (t_{j-1}, t_j^i). To guarantee there are at most two instances for each task in the Crenel Interval, we choose the minimum one from all the candidate Crenel Points as the next Crenel Point, that is,

$$t_j = \min \{t_j^i\}_{i=1}^n = \min \left\{ \left\lfloor \frac{t_{j-1}}{T_i} \right\rfloor T_i + 2T_i \right\}_{i=1}^n. \tag{5}$$

It is not difficult to see that the Crenel Point can be computed in $O(n)$ time. Hence, we can identify the Crenel Intervals in an online fashion.

5. SCHEDULING TASK INSTANCE IN EACH CRENEL INTERVAL

In the previous section, we have showed how to identify the Crenel Intervals. In this section, we present our method on scheduling the task instances in each Crenel Interval, so that all task deadlines can be met while DPM and DVFS can be performed to save energy. We first introduce the classification of the task instances in each Crenel Interval in Section 5.1, and then detail the CI-EDF and CI-EDF^m scheduling algorithms in Sections 5.2 and 5.3, respectively. Integration of DVFS with the proposed DPM schemes and dynamic slack reclaiming are shown in Sections 5.4 and 5.5, respectively. Finally, we compare the complexities of the proposed CI-EDF and CI-EDF^m algorithms with some existing approaches in Section 5.6.

5.1. Classification of Task Instances

For a task instance $\tau_{i,k}$ whose lifespan overlaps with a Crenel Interval CI_j, the relationship between $\tau_{i,k}$ and CI_j can be classified into three cases, as depicted in Figure 3 and detailed in the following.

—Case 1: $kT_i \le t_{j-1} < (k+1)T_i \le t_j$.
In this case, $\tau_{i,k}$ should finish its execution in CI_j. Note that if $kT_i < t_{j-1}$, then some workloads of $\tau_{i,k}$ may have been executed in previous Crenel Intervals and the left workloads of $\tau_{i,k}$ should be finished before its deadline $(k+1)T_i$, which is located in CI_j. We call $\tau_{i,k}$ a *mandatory* instance in CI_j. Since all the *mandatory* task instances are released at or before t_{j-1}, the workloads of these task instances can be executed continuously at the beginning of CI_j.

—Case 2: $kT_i \leq t_{j-1} < t_j < (k+1)T_i$.
In this case, the deadline of $\tau_{i,k}$, that is, $(k+1)T_i$, is after the Crenel Point t_j. Part of the workloads of $\tau_{i,k}$ can be optionally executed in CI_j and we call $\tau_{i,k}$ an *optional* instance in CI_j. An *optional* task instance $\tau_{i,k}$ may not finish its execution in CI_j, and the quantity of its workloads assigned to be executed in CI_j are determined by the priorities of other *optional* task instances, as well as their workloads scheduled to be executed in CI_j. Since the deadlines of all the *optional* task instances are after t_j, these task instances can be executed continuously at the end of CI_j.
—Case 3: $t_{j-1} < kT_i < t_j \leq (k+1)T_i$.
In this case, the release time of $\tau_{i,k}$ is in CI_j. If $t_j < (k+1)T$, it is obvious that $\tau_{i,k}$ is an *optional* task instance in CI_j. Otherwise, if $t_j = (k+1)T$, then $\tau_{i,k}$ should finish its workloads in CI_j. It seems that in this situation $\tau_{i,k}$ should be considered as a *mandatory* instance and scheduled to execute at the beginning of CI_j. But, if we do like this, there is a risk that when it is the turn for $\tau_{i,k}$ to execute, $\tau_{i,k}$ has not been released. Then, there may exist two slacks in CI_j. Hence, in this case, all the task instances are considered as *optional*.

Based on the preceding classification, we have the following observation.

OBSERVATION 2. *In one Crenel Interval, for each task τ_i, there is no more than one mandatory instance, and no more than one optional instance.*

We now given an example to illustrate how to classify the task instances in a Crenel Interval.

Example 2. Consider three periodic tasks $\tau = \{\tau_1, \tau_2, \tau_3\}$ with the following parameters: $C_1 = 1,\ C_2 = 3,\ C_3 = 4, T_1 = 4,\ T_2 = 11$, and $T_3 = 20$. $t_0 = 0$ is the first Crenel Point, and $t_1 = 8$, $t_2 = 16$, and $t_3 = 24$ are the Crenel Points derived by Equation (5). According to our classification, τ_1^m is the *mandatory* instance (Case 1), while τ_1^o (Case 3), τ_2^o (Case 3), and τ_3^o (Case 2) are the *optional* instances in CI_2, as shown in Figure 4. Note that although τ_1^o must finish its execution in CI_2, it is classified as an *optional* instance, since by Observation 2, there is at most one *mandatory* instance for each task in each Crenel Interval.

5.2. Scheduling Tasks in Each Crenel Interval

We first consider a simple case where there is only one device in the system, and all the tasks need to access this device. Before giving the detail of our scheduling algorithm, we introduce some notations (summarized in Table I). Since there is no more than one *mandatory* instance and no more than one *optional* instance for each task in each Crenel Interval, in the following discussion, we use τ_i^m and τ_i^o to denote the *mandatory* and *optional* instances of τ_i, respectively, and use W_i^m and W_i^o to denote the respective workloads of τ_i^m and τ_i^o, which are actually assigned to execute in the current Crenel Interval. Further, we use RW_i^j to denote the remaining workloads from previous CIs, which are to be executed (or partly) in CI_j. We now consider τ_i^m and τ_i^o in $\mathsf{CI}_j = (t_{j-1}, t_j)$.

—For τ_i^m, since the *mandatory* instance is required to finish its execution in each Crenel Interval, it is clear that if τ_i^m is released before t_{j-1}, then $W_i^m = RW_i^j$. Otherwise, if τ_i^m is released at t_{j-1}, then $W_i^m = C_i$.
—For τ_i^o, since some workloads of τ_i^o may be delayed to execute in the following Crenel Intervals, we cannot determine W_i^o at this moment. We use MW_i^j to denote the maximum workloads that can be executed in CI_j. If τ_i^o is released before t_{j-1}, then $MW_i^j = RW_i^j$. Otherwise, if τ_i^o is released in the current Crenel Interval, we have $MW_i^j = C_i$.

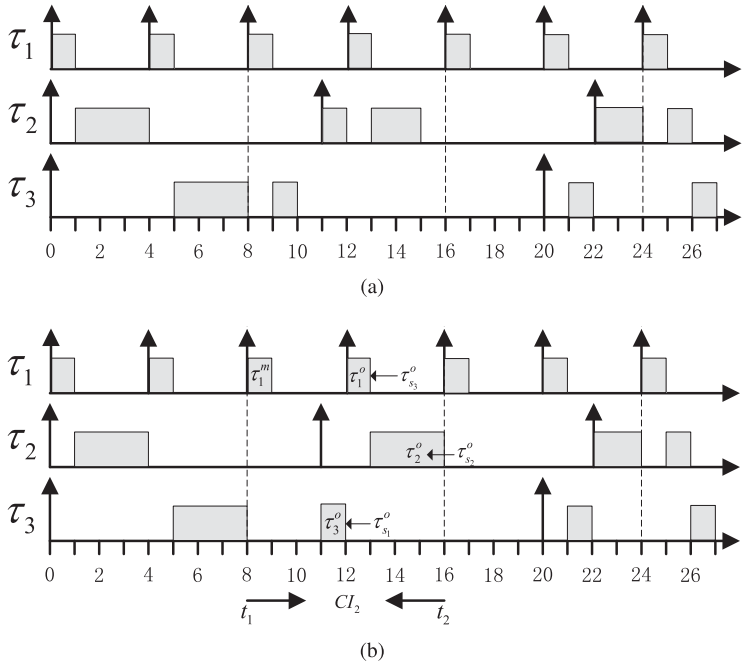


Fig. 4. Schedule produced by (a) EDF and (b) CI-based approaches.

Table I. Symbols and Definitions

Symbol	Definition
τ_i^m	The *mandatory* instance of τ_i in a Crenel Interval
τ_i^o	The *optional* instance of τ_i in a Crenel Interval
RW_i^j	Remaining workloads from previous CIs to be executed in CI_j
MW_i^j	Maximum workloads of τ_i^o that can be executed in CI_j
W_i^m	Workloads of τ_i^m need to be executed in a Crenel Interval
W_i^o	Workloads of τ_i^o need to be executed in a Crenel Interval
O_j	The set of *optional* instances in CI_j
$P_j^s(\tau_{i,k})$	Overlapping start point
$P_j^e(\tau_{i,k})$	Overlapping end point
$L_j(\tau_{i,k})$	Overlapping time length

Since we mainly focus on *optional* instances, for presentation convenience, we use O_j to denote the set of *optional* instances in CI_j. Now our major concern is how to calculate W_i^o for τ_i^o; we start by giving the following definition.

Definition 1. For a task instance $\tau_{i,k}$ whose execution overlaps with the Crenel Interval CI_j, its overlapping start point $P_j^s(\tau_{i,k})$, overlapping end point $P_j^e(\tau_{i,k})$, and overlapping length $L_j(\tau_{i,k}^o)$ can be defined as follows:

$$\begin{aligned} P_j^s(\tau_{i,k}) &= \max(t_{j-1}, kT_i), \\ P_j^e(\tau_{i,k}) &= \min(t_j, (k+1)T_i), \\ L_j(\tau_{i,k}) &= P_j^e(\tau_{i,k}) - P_j^s(\tau_{i,k}). \end{aligned} \tag{6}$$

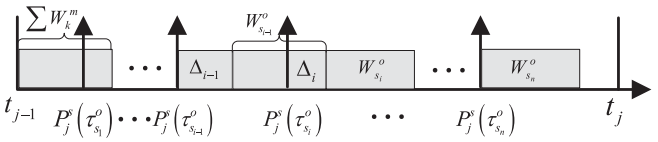


Fig. 5. Execution of *optional* instances.

Note that according to our classification of task instances introduced in Section 5.1, the overlapping start point of the *mandatory* instance τ_i^m is t_{j-1}, and the overlapping end point of the *optional* instance τ_i^o is t_j.

In our method, we use the EDF rule to identify priorities for all the task instances. From our classification of task instances, we know that in a Crenel Interval, the deadlines of the *mandatory* instances are always earlier than that of the *optional* instances. Hence, in each Crenel Interval, no *mandatory* instances will be preempted by *optional* instances, which means the execution of the *mandatory* instances are continuous and the *optional* instances only become eligible to execute after $t_{j-1} + \sum_{i=1}^{n} W_i^m$.

Now it is clear that the schedules of the *mandatory* and *optional* instances are independent in each Crenel Interval. Moreover, when computing W_i^o for τ_i^o in a Crenel Interval, we only need to consider the optional task instances whose priorities are higher than that of τ_i^o. For the optional task instance τ_l^o whose priority is lower than that of τ_i^o, we can regard $W_l^o = 0$. Since τ_i^o only becomes eligible to execute after $t_{j-1} + \sum_{i=1}^{n} W_i^m$, we have $P_j^s(\tau_{i,k}^o) = \max(t_{j-1} + \sum_{i=1}^{n} W_i^m, kT_i)$. As mentioned previously, $P_j^e(\tau_{i,k}^o) = t_j$. Then, $L_j(\tau_{i,k}^o)$ can be computed by

$$\begin{aligned} L_j\left(\tau_{i,k}^o\right) &= t_j - P_j^s\left(\tau_{i,k}^o\right) \\ &= \min\left(t_j - t_{j-1} - \sum_{i=1}^{n} W_i^m, t_j - \left\lfloor \frac{t_{j-1}}{T_i} \right\rfloor T_i\right). \end{aligned} \tag{7}$$

Note that in CI_j, the *optional* instances' overlapping start points are disordered with respect to their subscript. For presentation convenience, we use a new set $\{P_j^s(\tau_{s_i}^o)\}_{i=1}^n$ to denote the overlapping start points, so that they are in an ordered fashion with respect to the new sub-subscript, that is, if $a < b$, then $P_j^s(\tau_{s_a}^o) < P_j^s(\tau_{s_b}^o)$.

Now we consider how to compute $W_{s_i}^o$ for $\tau_{s_i}^o$. Since we use EDF to identify priorities for the optional task instances, when accounting for the maximum interferences from tasks with higher priorities than $\tau_{s_i}^o$, we need to consider the following two cases.

—The higher priority task instances whose overlapping start points are before $P_j^s(\tau_{s_i}^o)$. These task instances may have some workloads to be executed after $P_j^s(\tau_{s_i}^o)$. For simplicity, we use Δ_i to represent these workloads. In Figure 5, $\tau_{s_{i-1}}^o$ (with priority higher than $\tau_{s_i}^o$) is released at $P_j^s(\tau_{s_{i-1}}^o)$, while $\tau_{s_i}^o$ is released at $P_j^s(\tau_{s_i}^o)$, which is after $P_j^s(\tau_{s_{i-1}}^o)$. So $\tau_{s_i}^o$ cannot start its execution until $\tau_{s_{i-1}}^o$ finishes. In this figure, Δ_i is part of $W_{s_{i-1}}^o$ that is executed after $P_j^s(\tau_{s_i}^o)$.

—The higher priority task instances whose overlapping start points are after $P_j^s(\tau_{s_i}^o)$, just as τ_{s_n} shown in Figure 5. Obviously, these task instances' overlapping time lengths are included in $L_j(\tau_{s_i}^o)$ and they always consume the time budget before t_j. Hence, the sum workloads of this part is $\sum_{k=n}^{i} W_{s_k}^o$ in CI_j.

Then, by summing up the maximum interferences, we can get the maximum time budget which can be used by $\tau^o_{s_i}$ as follows:

$$W^o_{s_i} = \min \begin{cases} MW^j_{s_i}, \\ t_j - P^s_j(\tau^o_{s_i}) - \Delta_i - \sum_{k=n}^{i} W^o_{s_k}. \end{cases} \tag{8}$$

At first glance, it seems that we cannot compute $W^o_{s_i}$, since Δ_i is unknown at this moment. But actually, by the definition of Δ_i, we can use Δ_{i-1} to represent Δ_i. As shown in Figure 5, in interval $(P^s_j(\tau^o_{s_{i-1}}), P^s_j(\tau^o_{s_i}))$, there are two parts of workloads that need to be executed: (1) Δ_{i-1}; (2) the assigned workloads $W^o_{s_{i-1}}$ for *optional* instance $\tau^o_{s_{i-1}}$. Then, Δ_i can be described as follows.

$$\Delta_i = \max\left(0, W^o_{s_{i-1}} + \Delta_{i-1} - \left(P^s_j(\tau^o_{s_i}) - P^s_j(\tau^o_{s_{i-1}})\right)\right). \tag{9}$$

Consequently, $W^o_{s_i}$ can be represented as

$$W^o_{s_i} = \min \begin{cases} MW^j_{s_i}, \\ t_j - P^s_j(\tau^o_{s_i}) - \sum_{k=n}^{i} W^o_{s_k}, \\ t_j - P^s_j(\tau^o_{s_{i-1}}) - \sum_{k=n}^{i-1} W^o_{s_k} - \Delta_{i-1}. \end{cases} \tag{10}$$

If we repeat the replacement until Δ_1, then $W^o_{s_i}$ can be represented as follows.

$$W^o_{s_i} = \min \begin{cases} MW^j_{s_i}, \\ t_j - P^s_j(\tau^o_{s_i}) - \sum_{k=n}^{i} W^o_{s_k}, \\ t_j - P^s_j(\tau^o_{s_{i-1}}) - \sum_{k=n}^{i-1} W^o_{s_k}, \\ \vdots \\ t_j - P^s_j(\tau^o_{s_1}) - \sum_{k=n}^{1} W^o_{s_k} - \Delta_1. \end{cases} \tag{11}$$

When $i = 1$, since $\tau^o_{s_1}$ is the first released *optional* task instance and no other *optional* instances will interrupt its execution, we have $\Delta_1 = 0$. By $\Delta_1 = 0$ and $L_j(\tau^o_{s_i}) = t_j - P^s_j(\tau^o_{s_i})$, Formula (11) can be represented as follows for simplicity.

$$W^o_{s_i} = \min\left(MW^j_{s_i}, \left\{L_j(\tau^o_{s_q}) - \sum_{k=n}^{q} W^o_{s_k}\right\}^{1}_{q=i}\right). \tag{12}$$

By now, we have finished introducing our method on computing the actual workloads to be executed in a Crenel Interval for an *optional* instance. To help understand, we give an example to show how to compute $W^o_{s_i}$ in the following.

Illustrative Example. As shown in Figure 4(b), CI_2 is a Crenel Interval, in which $\tau^o_{s_1}$, $\tau^o_{s_2}$, and $\tau^o_{s_3}$ are three *optional* instances. Note here for this example, $P^s_j(\tau^o_{s_3}) = 12$, $P^s_j(\tau^o_{s_2}) = 11$, $P^s_j(\tau^o_{s_1}) = 9$, $L_j(\tau^o_{s_3}) = 4$, $L_j(\tau^o_{s_2}) = 5$, and $L_j(\tau^o_{s_1}) = 7$, where $s_1 = 3$, $s_2 = 2$, and $s_3 = 1$. The priority order of these *optional* instances is $p(\tau^o_{s_3}) > p(\tau^o_{s_1}) > p(\tau^o_{s_2})$. We calculate W^o_i $(1 \le i \le 3)$ according to their priorities, from high to low. Table II presents the details

Table II. Calculating W_i^o

Task instance	Value of W_i^o
τ_1^o	$W_1^o = \min(1, 4, 5, 7) = 1,\ W_2^o = 0,\ W_3^o = 0$
τ_3^o	$W_1^o = 1,\ W_2^o = 0,\ W_3^o = \min(1, 6) = 1$
τ_2^o	$W_1^o = 1,\ W_2^o = \min(3, 5, 4) = 3,\ W_3^o = 1$

of calculating each W_i^o. We first compute W_1^o. Since at first there is $W_1^o = W_2^o = W_3^o = 0$, by expression (12), we can get that $W_1^o = W_{s_3}^o = \min(MW_1^2, L_j(\tau_{s_3}^o), L_j(\tau_{s_2}^o), L_j(\tau_{s_1}^o))$. Since $MW_1^2 = 1$, we have $W_1^o = \min(1, 4, 5, 7) = 1$. Next, we proceed to calculate W_3^o; note here $W_1^o = 1$, $W_2^o = 0$, and $MW_3^2 = 1$. Then by expression (12), we have $W_3^o = W_{s_1}^o = \min(MW_3^2, L_j(\tau_3^o) - \sum_{k=n}^{1} W_k^o) = \min(1, 7 - 1) = 1$. Finally, W_2^o can be computed in a similar way.

Crenel-Interval-Based Scheduling with EDF (CI-EDF). Now we introduce our scheduling algorithm CI-EDF. With CI-EDF, all the *mandatory* and *optional* instances are scheduled by the EDF scheme. Specifically, in a Crenel Interval (t_{j-1}, t_j), all the *mandatory* instances are executed immediately at t_{j-1}. However, for *optional* instances, they only become eligible to execute at time instance $\gamma = t_j - \sum_{k=1}^{n} W_k^o$. By scheduling the task instances in such a way, there is a slack in the middle of each CI. Depending on whether the length of the slack is larger than the break-even time, the corresponding devices can be put into low-power state to save energy.

Algorithm 1 gives the pseudocode of computing γ in CI_j, with RW_i^j $(1 \le i \le n)$ as the input. First, in lines 2–13, we classify the task instances according to the method described in Section 5.1, to determine their workloads need to be executed (for *mandatory* instances), or the maximum workloads can be executed (for *optional* instances). Note here we consider Case 1 and Case 3 together in lines 3–9, since in both cases, there is one and only one deadline in CI_j. Case 2 is addressed in lines 10 and 11. Next, we compute the overlapping length for each *optional* instance (lines 14–16). Finally, we calculate the workloads need to be executed for all the *optional* instances (lines 17–24). Afterward, γ can be obtained (line 25).

Now we need to prove the correctness of CI-EDF on guaranteeing all task deadlines; we first introduce two lemmas.

LEMMA 2. *By the CI-EDF scheme, there is no idle time in interval* (γ, t_j).

PROOF. We prove it by contradiction. Suppose in interval (γ, t_j), there is an idle interval $(t, t + \varepsilon)$. It is clear no optional instances will release during this idle interval. Now we divide the *optional* instances into two groups, A and B. The instances in group A are released before t, and the instances in group B are released at or after $t + \varepsilon$. For each optional instance τ_a^o in group A, we use A_a^o to denote the maximum available execution time of τ_a^o. Since all the task instances in group A have finished their workloads, we have $A_a^o = MW_a^j \ge W_a^o$. For each instance τ_b^o in group B, we use A_b^o to denote the maximum available execution time of τ_b^o. It is clear there is $A_b^o = W_b^o$. In summary, we have $\sum_{i=1}^{n} A_i^o \ge \sum_{i=1}^{n} W_i^o = t_j - \gamma$, which means there are at least $t_j - \gamma$ *optional* workload requirements in (t_{j-1}, t_j).

On the other hand, as the interval $(t, t + \varepsilon)$ is idle, we know there are no workload requirements at time t, which implies that there are at most $t - \gamma$ *optional* workload requirements before t. Moreover, in interval $(t + \varepsilon, t_j)$, there are at most $t_j - t - \varepsilon$ *optional* instance workload requirements that can be executed. So in (t_{j-1}, t_j), there are at most $t_j - \gamma - \varepsilon$ *optional* workloads that need to be executed. But as mentioned previously,

ALGORITHM 1: Calculate γ

input: $RW_i^j (1 \le i \le n)$;
output: γ.
1: Initialize MW_i^j, W_i^m, and W_i^o $(1 \le i \le n)$ to be 0;
2: **for** $(i = 1; i \le n; i++)$ **do**
3: **if** $\lfloor \frac{t_{j-1}}{T_i} \rfloor T_i + T_i \le t_j$ **then**
4: $MW_i^j = C_i$;
5: **if** $\lfloor \frac{t_{j-1}}{T_i} \rfloor T_i == t_{j-1}$ **then**
6: $W_i^m = C_i$;
7: **else**
8: $W_i^m = RW_i^j$;
9: **end if**
10: **else**
11: $MW_i^j = RW_i^j$;
12: **end if**
13: **end for**
14: **for** $(i = 1; i \le n; i++)$ **do**
15: $L_j(\tau_i^o) = \min\left(t_j - \lfloor \frac{t_{j-1}}{T_i} \rfloor T_i, t_j - t_{j-1} - \sum_{i=1}^{n} W_i^m\right)$;
16: **end for**
17: Map τ_k^o $(1 \le k \le n)$ in O_j to $\tau_{s_i}^o$ $(1 \le i \le n)$ so that $\{P_j^s(\tau_{s_i}^o)\}_{i=1}^n$ is nondecreasing with respect to i;
18: **while** ($\tau_{s_i}^o$ has the earliest deadline in O_j) **do**
19: **for** $(q = i; q \ge 1; q--)$ **do**
20: $W_{s_i}^o = \min\left(MW_{s_i}^j, L_j(\tau_{s_q}^o) - \sum_{k=n}^{q} W_{s_k}^o\right)$;
21: **end for**
22: $RW_{s_i}^{j+1} = MW_{s_i}^j - W_{s_i}^o$;
23: Remove $\tau_{s_i}^o$ from O_j;
24: **end while**
25: Return $\gamma = t_j - \sum_{i=1}^{n} W_{s_i}^o$;

there are at least $t_j - \gamma$ *optional* workload requirements in (t_{j-1}, t_j). We thus come to a contradiction and the lemma follows. □

By the previous lemma, we know all the $t_j - \gamma = \sum_{k=1}^{n} W_k^o$ *optional* workloads can be finished before t_j.

LEMMA 3. *For each Crenel Interval* CI_j, RW_i^j *under CI-EDF scheduling, denoted by* $(RW_i^j)_{CI}$, *is the same as* RW_i^j *under EDF scheduling, that is,* $(RW_i^j)_{EDF}$.

PROOF. We only need to consider *optional* instances, since for *mandatory* instances, they need to finish their execution requirements in CI_{j-1}, no matter under EDF or CI-EDF, which indicates that the *mandatory* instances would not impact the calculation of RW_i^j.

Now we use mathematical induction to prove the claim for *optional* instances. When $j = 1$, that is, in the first Crenel Interval, since there are no CIs before CI_1, it is obvious that $(RW_i^1)_{EDF} = (RW_i^1)_{CI} = 0$. Without loss of generality, suppose when $j = k$, $(RW_i^k)_{EDF} = (RW_i^k)_{CI}$ also holds; now we need to prove $(RW_i^{k+1})_{EDF} = (RW_i^{k+1})_{CI}$. From Lemma 2, we know in each CI, the workloads of the optional instance τ_i^o under both EDF and CI-EDF, denoted by $(W_i^o)_{EDF}$ and $(W_i^o)_{CI}$, respectively, are the same, that is, $(W_i^o)_{EDF} = (W_i^o)_{CI}$. Moreover, note that in CI_k, only the *optional* instances can be

delayed to execute in CI_{k+1}, and performed as RW_i^{k+1}. So now regarding τ_i^o, we have two cases to be considered: (1) If τ_i^o is released before t_{k-1}, then $RW_i^{k+1} = MW_i^k - W_i^o$, under both EDF and CI-EDF. In this case, no instances of τ_i will be released, hence $MW_i^k = RW_i^k$. (2) If τ_i^o is released in CI_k, then we have $RW_i^{k+1} = C_i - W_i^o$, under both EDF and CI-EDF scheduling. By $(RW_i^k)_{EDF} = (RW_i^k)_{CI}$ and $(W_i^o)_{EDF} = (W_i^o)_{CI}$, we can get that $(RW_i^{k+1})_{EDF} = (RW_i^{k+1})_{CI}$. The lemma is proved. □

Now we are ready to show the correctness of CI-EDF on guaranteeing all task deadlines.

THEOREM 1. *With the workloads computed by Equation (12), the task set is schedulable under CI-EDF, provided that* $\sum \frac{C_i}{T_i} \leq 1$.

PROOF. Since we consider an implicit deadline task set, it is clear if $\sum \frac{C_i}{T_i} \leq 1$, then the task set is EDF schedulable. Without loss of generality, we consider an arbitrary Crenel Interval CI_j. For τ_i^m, since $W_i^m = RW_i^j$ or $W_i^m = C_i$, from Lemma 3, we know the workloads of τ_i^m are the same under both EDF and CI-EDF scheduling. Since the task set is EDF schedulable and all the *mandatory* instances are also scheduled by the EDF scheme, we know τ_i^m in CI_j can meet its deadline under CI-EDF. For the *optional* instance τ_i^o, we only need to consider the case where τ_i^o's deadline is equal to t_j. In this case, there is $W_i^o = MW_i^j$ under both EDF and CI-EDF scheduling. From Lemma 2, we know τ_i^o can finish W_i^o workloads before t_j, which means no *optional* instances will miss their deadlines. Since the task set is schedulable in each Crenel Interval, we can conclude that the task set is schedulable under CI-EDF and the theorem thus follows. □

Generalization to Frame-Based Task Set. Now with CI-EDF, we consider a special case where all the tasks in $\mathcal{T}$ have a common period length T, that is, we consider a frame-based task set. In this case, given t_{j-1}, the Crenel Point t_j can be computed by $t_j = \lfloor \frac{t_{j-1}}{T} \rfloor T + 2T$. Since $t_0 = 0$, it is clear that $t_j = 2jT$, which means each Crenel Interval has a length of $2T$, the double of a frame length. Then, our method CI-EDF works as follows: for an instance $\tau_{i,k}\,(k \geq 0)$, if k is even, then the workload of $\tau_{i,k}$, which is released at kT, can execute immediately; otherwise, if k is odd, then the workload of $\tau_{i,k}$, which is released at kT, is only eligible to execute at $(k+1)T - \sum_{i=1}^{n} C_i$. It is interesting to see that for the frame-based task set, our method can produce the same schedule as the one proposed in Gerards and Kuper [2013], which has been proved to be globally optimal on minimizing the energy consumption when the workloads of the same tasks in successive frames are identical.

5.3. Scheduling Tasks in Each Crenel Interval with Multiple Devices

In this section, we consider a more general case where there are multiple devices in the system, and propose the Crenel-Interval-based scheduling algorithm. To distinguish, we use CI-EDF^m to denote this algorithm. In a multiple device model, a task τ_i may access multiple devices, and correspondingly a device $\mathcal{D}_k$ may be accessed by multiple tasks. We use $\mathcal{AD}(\tau_i)$ to denote the set of devices accessed by task τ_i, and use $\mathcal{T}(\mathcal{D}_k)$ to denote the set of tasks that access device $\mathcal{D}_k$.

The same as CI-EDF, in CI-EDF^m, we also rely on Crenel Intervals to guide the scheduling of task instances. Considering that a task instance $\tau_{i,j}$ of τ_i may access multiple devices during its execution, we calculate separate Crenel Intervals for each device $\mathcal{D}_k \in \mathcal{AD}(\tau_i)$, denoted by DCI^k. Note, here we only need to use the tasks in

$\mathcal{T}(\mathcal{D}_k)$ to compute DCI^k, and the calculation method is the same as the one presented in Section 4.2.

Apparently, $\tau_{i,j}$ may locate in multiple DCIs, and thus can be classified as *mandatory* instance in one DCI, but *optional* in another. In other words, there are multiple choices from different DCIs for $\tau_{i,j}$ on when to execute. But since $\tau_{i,j}$ can only execute at one time instant, it is imperative to design a strategy to help make the choice so that the maximum energy savings can be obtained. To achieve this goal, we use a weighting factor based method to provide the guidance. Specifically, the weighting factor for DCI^k is defined as follows:

$$WF^k(\tau_{i,j}) = \begin{cases} -\frac{1}{B_k}\left(P_a^k - P_s^k\right), & \tau_{i,j} \text{ is } \textit{mandatory} \text{ instance in } \mathsf{DCI}^k \\ \frac{1}{B_k}\left(P_a^k - P_s^k\right), & \tau_{i,j} \text{ is } \textit{optional} \text{ instance in } \mathsf{DCI}^k, \end{cases} \tag{13}$$

where B_k is the break-even time of $\mathcal{D}_k$, and P_a^k (P_s^k) denotes the power consumption of $\mathcal{D}_k$ in *active* (*sleep*) state. The reason behind using this weighting factor is twofold: (1) Given a fixed-length slack, a device with smaller break-even time has a higher possibility to be transitioned to low-power state; and (2) given a fixed-length slack, a device with large $(p_a^k - p_s^k)$ can save more energy when transitioned to *sleep* state. Moreover, if $\tau_{i,j}$ is an *optional* instance in DCI^k, then we set the weighting factor to be positive. On the contrary, if $\tau_{i,j}$ is a *mandatory* instance in DCI^k, then we set the weighting factor to be negative. After computing the weighting factors for all the DCIs, we sum them up to get the final weighting factor of $\tau_{i,j}$,

$$WF(\tau_{i,j}) = \sum\nolimits_{\mathcal{D}_k \in \mathcal{AD}(\tau_i)} WF^k(\tau_{i,j}). \tag{14}$$

If $WF(\tau_{i,j}) > 0$, it means delaying $\tau_{i,j}$ can obtain more energy savings. Otherwise, if $WF(\tau_{i,j}) < 0$, it indicates that executing $\tau_{i,j}$ without delay can save more energy. Note here if $WF(\tau_{i,j}) = 0$, we let $\tau_{i,j}$ execute without delay to break the tie.

Based on the previous discussion, we can classify $\tau_{i,j}$ into the following four cases:

—Case (1): $\tau_{i,j}$ is a *mandatory* instance in DCI^k and is suggested to execute without delay.
—Case (2): $\tau_{i,j}$ is a *mandatory* instance in DCI^k but is suggested to delay its execution.
—Case (3): $\tau_{i,j}$ is an *optional* instance in DCI^k but is suggested to execute without delay.
—Case (4): $\tau_{i,j}$ is an *optional* instance in DCI^k and is suggested to delay its execution.

For presentation convenience, in the following discussion, we call the instance that belongs to the preceding four cases C1, C2, C3, and C4 instance, respectively. In CI-EDF, the *device slacks* always locate in the middle of a Crenel Interval, which indicates that there is only one type of device slack. Different from CI-EDF, in CI-EDF^m, the *device slacks* used to transition $\mathcal{D}_k$ into *sleep* state can be classified into two types: One is the slack between two C1 instances, or two separated execution parts of one C1 instance. The other is the slack between the last C1 instance and the first C2 (or C3, C4) instance. As an illustration, we give an example next.

Example 3. In Figure 6, J_1, J_2, J_3, J_4 are four task instances with priority order of $Pr(J_1) > Pr(J_2) > Pr(J_3) > Pr(J_4)$. J_2, J_3, and J_4 are all released at t_0 and need to access $\mathcal{D}_k$, while J_1 is released at t_1 and does not use $\mathcal{D}_k$. J_2 and J_3 are two C1 instances that can be executed without delay. At t_0, J_2 starts its execution and then is preempted by J_1 at t_1. After J_1 finishes its execution at t_2, J_2 resumes, followed by J_3. J_4 is a C2 instance and needs to be delayed to execute at t_5. In this example, S_1 (which is actually the WCET of J_1) between the two separated execution parts of J_2 is the first type of slack, while S_2 between J_3 and J_4 is the second type of slack.

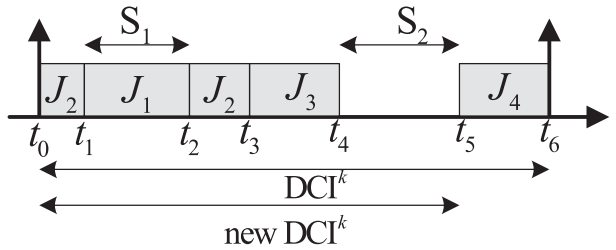


Fig. 6. Two types of device slack.

Clearly, all the C1 and C3 instances will not be delayed and thus are available to execute when they are released. Now we show how to compute the execution time instant $k_{i,j}$ for each C1 (or C3) instance $\tau_{i,j}$. Note here $k_{i,j}$ may store multiple time instants, since if $\tau_{i,j}$ is a C1 instance, it may be preempted by other instances and resume execution later, just as J_2 shown in Figure 6, for which there is $k_2 = \{t_0, t_2\}$. We first define another data structure TI to record the time interval that has not been used. TI is a two-tuple in the form of (a, b), where a and b represent the start and end points of the interval, respectively. TI-List consists of all the TIs. Initially, there is only one TI in TI-List, which is equal to the interval of DCI^k. Next, we calculate the actual execution time instant for the C1 and C3 instances one by one according to their priorities. When computing $k_{i,j}$ for $\tau_{i,j}$, we need to utilize the release time and the WCET of $\tau_{i,j}$, as well as the TI-List. Finally, we update the TI-List to remove the corresponding execution time intervals that have been used. To help understand the computation process of $k_{i,j}$, we reuse the job set in Example 3 to give an illustration.

Example 4. As shown in Figure 6, t_0 and t_6 are the start and end Crenel Points of DCI^k, respectively. Hence, the initial element in TI-List is (t_0, t_6). Since J_1 has the highest priority and will be released at t_1, and moreover, there is only one element (t_0, t_6) in TI-List, we have $k_1 = t_1$. Since the WCET of J_1 is in the length of $(t_2 - t_1)$, we update TI-List to be with two elements: (t_0, t_1) and (t_2, t_6). Then we proceed to J_2. Since J_2 is released at t_0 and its WCET is in the length of $(t_3 - t_0) - (t_2 - t_1)$, combining with the information in TI-List, we can get that $k_2 = \{t_0, t_2\}$. Afterward, we update TI-List to be with one element: (t_3, t_6). Following a similar way, we can obtain that $k_3 = t_3$.

After computing the execution time instants of all the C1 and C3 instances, we can derive the first type of device slack easily. Then, by comparing the length between the slack and the break-even time of the device, we can determine whether a DPM operation should be conducted or not.

Now the remaining challenge is how to compute the second type of device slack. As mentioned previously, the second type of device slack is the one between the last C1 instance and the first C2 (or C3, C4) instance. Since the last C1 instance (as well as the first C3 instance) can be determined by its execution time instant and WCET, now our major concern is how to compute the time instant when the first C2 (or C4) instance is available to execute. In other words, we need to decide how long the C2 (or C4) instance can be delayed but without jeopardizing the schedulability of the task set. Without loss of generality, we suppose the first such C2 (or C4) instance is $\tau_{i,j}$, and use $\lambda_{i,j}$ to denote the time instant when $\tau_{i,j}$ is available to execute.

Similar to the EEDS scheme proposed in Cheng and Goddard [2006b], we use a *runtime*-based method to compute $\lambda_{i,j}$ for $\tau_{i,j}$. The concept of *runtime* comes from known techniques [Jejurikar and Gupta 2005a], denoting the time budget allocated to each task instance. Since the system utilization is usually smaller than 1, if we assign C_i/U (U is the utilization of the task set) runtime to $\tau_{i,j}$, the system will not be overloaded

and the task set remains schedulable. To facilitate computation, we maintain a runtime list, RT-List, in the system. When a task instance is released, the associated runtime is inserted into RT-List. The runtimes have the same priorities with their associated task instances, and are sorted according to their priorities. The head of the RT-List is the runtime with the highest priority. The runtime is always consumed, not only when there are workloads being executed, but also when the processor is idle. The runtime consumption is always from the head of the RT-List. If the residual runtime of the head node is zero, it will be removed from RT-List, and the next node acts as the head node. To compute $\lambda_{i,j}$, we use the following notations, which are similar to the ones used in Cheng and Goddard [2006b].

—$R_{i,j}$: the runtime associated with $\tau_{i,j}$.
—$Pr(R_{i,j})$: the priority of the runtime $R_{i,j}$, which is the same as the priority of $\tau_{i,j}$.
—$C_{i,j}(t)$: the residual execution time of $\tau_{i,j}$ at time t.
—$AR_{i,j}(t)$: the available runtime for $\tau_{i,j}$, can be calculated by $R_{i,j} + \sum_{Pr(R_{a,b})>Pr(R_{i,j})} R_{a,b}$.

With the preceding notations, the time instant $\lambda_{i,j}$ for $\tau_{i,j}$ can be calculated by

$$\lambda_{i,j} = t + AR_{i,j}(t) - C_{i,j}(t). \tag{15}$$

After computing $\lambda_{i,j}$, we can derive the second type of device slack easily and check whether it is long enough for a DPM operation.

From Figure 6, we can observe that the two types of device slacks are both located before the time instant when the first C2 (or C3, C4) instance is available to execute. So, if we revise the end Crenel Point of DCI^k to this time instant, the possible transition of Device $\mathcal{D}_k$ would not be affected. Moreover, after the revision, it is easy to schedule in the new DCI^k, since there are only C1 instances in it. As an illustration, let us go back to Example 3. As can be seen, J_3 is the first and only C2 instance in DCI^k, so t_5 is the time instant when the first C2 instance is available to execute. After revising the end Crenel Point of DCI^k to be t_5, the new DCI^k is (t_0, t_5). Then, we only need to consider the C1 instances in the new DCI^k, and choose t_5 as the start Crenel Point to compute the next DCI. All the C2, C3, and C4 instances in DCI^k can be scheduled to execute in the next DCI.

Algorithm 2 gives the pseudocode of CI-EDF^m. The same as CI-EDF, in CI-EDF^m, we also need the information of the task instances that are released before or in the current DCI to help scheduling. In particular, the start Crenel Point of a DCI is a very important scheduling point. We define a data structure INS to record the information of each task instance that will be released in DCI. INS includes a lot of information regarding an instance, including the number for identifying the instance; the available execution time, which is initialized to be the release time of the instance; the worst-case execution time of the instance; the priority of the instance; and the weighting factor of the instance calculated by Equation (14). INS-List consists of all the INSs, which are sorted in decreasing order of their priorities. To avoid inserting the same INS in the next DCI, we maintain a global variant *Stamp* to keep the end Crenel Point of the current DCI. Then, in the next DCI, only instances that are released after *Stamp* can be inserted into INS-List. Starting by initializing the start Crenel Point of all the DCIs to be 0, Algorithm 2 schedules the task instances at two kinds of time instants, as detailed next.

(1) When t is the start Crenel Point of DCI^k, we first calculate the end Crenel Point for DCI^k by the method presented in Section 4.2. Then we initialize TI-List with the interval of DCI^k. Next, for each instance that will be released in DCI^k, we create a corresponding INS and insert it into INS-List. Meanwhile, we insert the corresponding *runtime*s of the instances into RT-List and change *Stamp* to the end

ALGORITHM 2: CI-EDF^m

1: **Preprocessing**:
2: Initialize the start Crenel Point of all the DCIs to be 0;
3: **When t is the start Crenel Point of DCI^k:**
4: Calculate the end Crenel Points of DCI^k;
5: Initialize TI-List with one element, which is the interval of DCI^k;
6: Insert INSs into INS-List, insert the corresponding *runtime*s into RT-List and change *Stamp* to the end point of DCI^k;
7: **for** $\tau_{i,j}$ in INS-List which needs to calculate $WF(\tau_{i,j})$ **do**
8: Calculate $WF(\tau_{i,j})$;
9: **if** $WF(\tau_{i,j}) > 0$ **then**
10: Calculate $\lambda_{i,j}$ by Equation (15);
11: **else**
12: Calculate $k_{i,j}$ by using TI-List;
13: **end if**
14: Update TI-List;
15: **end for**
16: Select the first available instance with the highest priority from INS-List to execute;
17: **When an instance completes or is preempted at time t:**
18: Find the next available task instance from INS-List for execution;
19: **for** each active device $\mathcal{D}_k$ **do**
20: Find the next task instance $\tau_{i,j}$ which uses $\mathcal{D}_k$ in INS-List;
21: $flag = 0$, $temp = 0$;
22: **if** $(WF(\tau_{i,j}) \leq 0)$ **then**
23: **if** ($\tau_{i,j}$ is an optional instance in DCI^k) **then**
24: $temp = k_{i,j}$, $flag = 1$;
25: **end if**
26: **if** $((k_{i,j} - t) > B_k)$ **then**
27: Transition $\mathcal{D}_k$ into sleep state;
28: **end if**
29: **else**
30: $temp = \lambda_{i,j}$, $flag = 1$;
31: **if** $(\lambda_{i,j} - t) > B_k)$ **then**
32: Transition $\mathcal{D}_k$ into sleep state;
33: **end if**
34: **end if**
35: **if** $(flag == 1)$ **then**
36: Revise the end Crenel Point of DCI^k to be *temp*;
37: **end if**
38: **end for**

point of DCI^k. When creating INS for $\tau_{i,j}$, we need to calculate $WF(\tau_{i,j})$ for it. But it is worth noting that there is no need to calculate WF for all the instances in DCI^k. Specifically, if $\tau_{a,b}$ is released before t and needs to access another device $\mathcal{D}_q$ that is already in sleep state till time $t' > t$, then we do not need to calculate $WF(\tau_{a,b})$. This is because $\tau_{a,b}$ is deemed to be delayed. The value of $WF(\tau_{i,j})$ indicates whether $\tau_{i,j}$ should be delayed or not. If $WF(\tau_{i,j}) > 0$, it means $\tau_{i,j}$ should be delayed, then we calculate its available execution time instant $\lambda_{i,j}$ by Equation (15). Otherwise, if $WF(\tau_{i,j}) \leq 0$, it means $\tau_{i,j}$ should be executed without delay, then we compute its available execution time instant $k_{i,j}$ by utilizing the information in TI-List. Finally, we update TI-List.

(2) When t is the completion or preemption time of a task instance, first, we need to choose an instance that has the highest priority among the ready instances (line 20). Then we need to decide whether an active device can be transitioned into

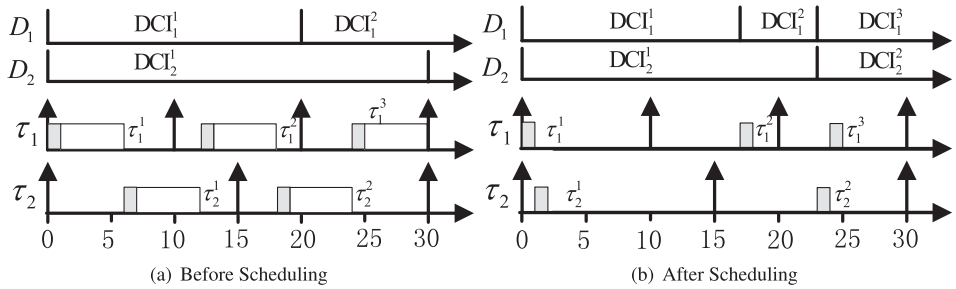


Fig. 7. CI-EDF in multiple device model.

sleep state or not. As discussed previously, there are two types of slacks. For each active device $\mathcal{D}_k$, we find the first instance $\tau_{i,j}$ that will use it. If $\tau_{i,j}$ is a C1 (or C3) instance (lines 22–28), we compare $(k_{i,j} - t)$ with B_k to determine whether a DPM operation can be performed. If $\tau_{i,j}$ is a C2 (or C4) instance (lines 29–34), we compare $(\lambda_{i,j} - t)$ with B_k to determine whether a DPM operation can be performed. Finally, we revise the end Crenel Point of DCI^k to be the time instant when the first C2 (or C3, C4) instance is available to execute (lines 35–37).

To help understand, we use a concrete example to show how CI-EDF^m works.

Example 5. Consider the following two periodic tasks specified with their WCET, relative deadlines, and periods: $\tau_1 = (1, 10, 10)$, $\tau_2 = (1, 15, 15)$ (all in *ms*). There are two devices $\mathcal{D}_1$ and $\mathcal{D}_2$, both with break-even time to be 15. $\mathcal{D}_1$ is accessed by τ_1 and τ_2, while $\mathcal{D}_2$ is accessed by τ_2. Figure 7(a) depicts the scheduling with runtime, where the white box denotes the remaining time that the time budget calculated according to the concept of runtime subtracts the WCET of the task instance, and the gray box denotes the WCET of the task instance. Figure 7(b) depicts the scheduling with CI-EDF^m. On top of these two subfigures are the DCIs of the devices.

At time $t = 0$, we initialize the start Crenel Point of DCI_1^1 and DCI_2^1 to be 0. By Equation (5), we can calculate the end Crenel Points of DCI_1^1 and DCI_2^1 to be 20 and 30, respectively. Note here we only need to use the tasks that access D_k to calculate the end Crenel Point of DCI^k. At first, TI-List is initialized to be $(0, 30)$. Next, the INSs for the instances that are released in interval $(0, 30)$ are created and inserted into INS-List. These instances include $\tau_{1,1}$, $\tau_{1,2}$, $\tau_{1,3}$, $\tau_{2,1}$, and $\tau_{2,2}$. Meanwhile, the corresponding *runtime*s are inserted into RT-List and *Stamp* is set to be 30.

It is easy to compute that $WF(\tau_{1,1}) < 0$ and $WF(\tau_{2,1}) < 0$. Hence, $\tau_{1,1}$ is a C1 instance in DCI_1^1, while $\tau_{2,1}$ is a C1 instance in both DCI_1^1 and DCI_2^1. Similarly, it can be derived that $\tau_{1,2}$ is a C4 instance in DCI_1^1, while $\tau_{2,2}$ is a C4 instance in both DCI_2^1 and DCI_2^2. For $\tau_{1,1}$ and $\tau_{2,1}$, we need to compute $k_{1,1}$ and $k_{2,1}$. Since $\tau_{1,1}$ has the highest priority, it is obvious that $k_{1,1} = 0$. Then by $C_1 = 1$, we can update the element in TI-List to be $(1, 30)$. Following a similar way, we can get that $k_{2,1} = 1$. For $\tau_{1,2}$ and $\tau_{2,2}$, by Equation (15), we can compute that $\lambda_{1,2} = (3 \times 6 - 1) = 17$ and $\lambda_{2,2} = (4 \times 6 - 1) = 23$, respectively.

After computing the necessary information of the task instances in DCI^k, we choose $\tau_{1,1}$, the instance that has the highest priority in INS-List to execute at $t = 0$. The first instance needs to access $\mathcal{D}_2$ is $\tau_{2,1}$, which will execute at $t = 1$. Then, we know $(0, 1)$ is the first type of device slack of $\mathcal{D}_2$. This slack is not long enough for transitioning $\mathcal{D}_2$. At time $t = 1$, $\tau_{1,1}$ finishes its execution and $\tau_{2,1}$ is selected to execute. The next instance needs to access $\mathcal{D}_1$ is $\tau_{1,2}$, which is delayed to execute $\lambda_{1,2} = 17$. Hence, $(1, 17)$ is the second type of device slack (longer than $B_1 = 15$), which can be used to transition $\mathcal{D}_1$ into sleep state. For DCI_1^1 , since $\tau_{1,2}$ is the first C4 instance that needs to access $\mathcal{D}_1$, we

revise the end Crenel Point of DCI_1^1 to be 17. At time $t = 2$, $\tau_{1,2}$ finishes its execution and there are no available instances in INS-List. The first instance that needs to access $\mathcal{D}_2$ is $\tau_{2,2}$, then $(2, 23)$ is the second type of device slack (longer than $B_2 = 15$), which can be used to transition $\mathcal{D}_2$ into sleep state. For DCI_1^2, $\tau_{2,2}$ is the first C4 instance that needs to access $\mathcal{D}_2$; we thus can revise the end Crenel Point of DCI_2^1 to be 23.

At time $t = 17$, since it is the start Crenel Point of DCI_1^2, we need to calculate DCI_1^2's end Crenel Point (which is 30) by Equation (5). Then we initialize the TI-List to be $(17, 30)$. Since $Stamp = 30$, we do need to create and insert any INS into INS-List. At $t = 17$, INS-List maintains the information of three instances, which are $\tau_{1,2}$, $\tau_{1,3}$, and $\tau_{2,2}$. We can calculate the weighting factors of $\tau_{1,2}$ and $\tau_{1,3}$ to be $WF(\tau_{1,2}) < 0$ and $WF(\tau_{1,3}) > 0$, hence $\tau_{1,2}$ and $\tau_{1,3}$ are the mandatory and optional instances in DCI_1^2, respectively. But note here we do not need to calculate $WF(\tau_{2,2})$, since $\tau_{2,2}$ has been delayed to $\lambda_{2,2} = 23$ in the previous DCI ($\mathcal{D}_2$ is in sleep state and will not be active until 23). For $\tau_{1,3}$ that needs to be delayed, we can compute that $\lambda_{1,3} = (5 \times 6 - 1) = 29$. Since $\tau_{1,2}$ has the highest priority and the element in TI-List is $(17, 30)$, we have $k_{1,2} = 17$, which indicates that $\tau_{1,2}$ will start its execution at $t = 17$. When $\tau_{1,2}$ completes at $t = 18$, there are no available instances in INS-List. We notice that the first instance that needs to access $\mathcal{D}_1$ is $\tau_{2,2}$, which is delayed to execute at $t = 23$. Hence, $(18, 23)$ is the second type of device slack for $\mathcal{D}_1$. This slack is not long enough for transitioning $\mathcal{D}_1$ into sleep state. Finally, we revise the end point of DCI_1^2 to be 23 to start the schedule of the next DCI in a similar way.

Now we show the correctness of CI-EDF^m on guaranteeing all task deadlines.

THEOREM 2. *The task set is schedulable under CI-EDF^m, provided that* $\sum \frac{C_i}{T_i} \leq 1$.

PROOF. Notice that in our CI-EDF^m algorithm, only the C2 and C4 instances are delayed to execute. Since we use the same *runtime*-based method as EEDS [Cheng and Goddard 2006b] to predict how long these instances can be delayed, given that $\sum \frac{C_i}{T_i} \leq 1$, the theorem follows directly from the result of Theorem 3.1 in Cheng and Goddard [2006b]. □

5.4. Integrating DVFS with DPM

In the previous two subsections, we provided the details of two DPM algorithms CI-EDF and CI-EDF^m. Now, we briefly describe how to integrate DVFS with our DPM schemes to reduce the system-level energy consumption. The same as in Devadas and Aydin [2010], we also adopt the system energy-efficient frequency, which is originally from Cheng and Goddard [2005]. In Devadas and Aydin [2010], the energy-efficient frequency threshold, denoted by ***EEF***, is derived based on the power characteristics of the processor, active devices, and devices in sleep state, at task instance dispatch times. The processor frequency is never reduced below this threshold. In this article, we use $\boldsymbol{EEF}(\mathcal{AD}(\tau_{i,j}))$ to denote the energy-efficient frequency of $\tau_{i,j}$; recall here $\mathcal{AD}(\tau_{i,j})$ is the set of devices $\tau_{i,j}$ needs to access. Specifically, energy-efficient frequency is derived by energy efficiency scale. Energy efficiency scale, which is used to compare the overall system energy efficiency to complete one unit workload with different processor frequency, is modeled by

$$\boldsymbol{ES}(f, \mathcal{AD}(\tau_{i,j})) = \frac{1}{f}\left(P_{cpu}(f) + \sum\nolimits_{D_k \in \mathcal{AD}(\tau_{i,j})} \left(P_a^k - P_s^k\right)\right). \tag{16}$$

Then, $\boldsymbol{EEF}(\mathcal{AD}(\tau_{i,j}))$ is the processor frequency f that can minimize the energy efficiency scale $\boldsymbol{ES}(f, \mathcal{AD}(\tau_{i,j}))$.

In CI-EDF, since all the tasks use the same device, we can get a unique EEF. Considering that the system utilization U is another threshold frequency, we can get the final system energy-efficient frequency to be $f = \max(f_{\min}, U, EEF)$ (recall that $f_{\min}$ is the critical speed), for all the task instances. Then, the integrated scheme works as follows: all the task instances are scheduled by CI-EDF and execute with frequency f.

In CI-EDF^m, the available *runtime* $AR_{i,j}(t)$ for $\tau_{i,j}$ is the time budget that can be used by $\tau_{i,j}$ to execute but without jeopardizing the schedulability, then we can get another threshold frequency to be $\frac{C_{i,j}(t)}{AR_{i,j}(t)}$. Consequently, the final energy-efficient frequency for $\tau_{i,j}$ can be computed by $f = \max(f_{\min}, \frac{C_{i,j}(t)}{AR_{i,j}(t)}, EEF(\mathcal{AD}(\tau_{i,j})))$. Then, the integrated schemes work as follows: all the task instances are scheduled by CI-EDF^m, and at each scheduling point t (t is the start Crenel Point of DCI^k, or the time instant when a task instance completes or is preempted), the processor frequency is updated to be f. Note that changing from one frequency level to another usually takes a fixed amount of time (denoted by Δt, ranges from tens of microseconds to tens of milliseconds), referred to as the transition (or switch) overhead [Mochocki et al. 2007]. When frequency transition overhead is large enough that they cannot be ignored, we need to recompute the final energy-efficient frequency by considering such overhead.

5.5. Dynamic Slack Reclaiming

In practice, the actual execution time of many task instances are usually smaller than their WCETs. In this case, dynamic slack arises due to early task completions. Since dynamic slack can help increase the length of device idle intervals, it is important to reclaim dynamic slack at runtime. In fact, reclaiming unused computation time to reduce the CPU speed or prolong the idle interval while preserving feasibility has been widely studied in numerous research papers in the past. Now we briefly discuss how to reclaim dynamic slacks in our Crenel-Interval-based approaches.

In CI-EDF, when a mandatory task instance completes early in a Crenel Interval, other mandatory instances with lower priorities are available to execute. Hence, this kind of dynamic slack can be merged into the slack interval, which is located in the middle of the CI. For the optional instances, since they may leave some workloads to execute in the following Crenel Intervals, if we use this kind of dynamic slack to execute more workloads, then less workloads are left to execute in the following Crenel Intervals, which in turn can prolong the length of the slack intervals in the following Crenel Intervals. It is also possible that all optional instances complete before the end Crenel Point of Crenel Interval. In such case, we revise the end Crenel Point of Crenel Interval to the completion instant of last optional instance. In CI-EDF^m, it is comparatively easy to do the reclaiming, since there are only mandatory instances in a Crenel Interval. We can use a similar method as in CI-EDF to reclaim the slacks.

5.6. Complexity Analysis

In this section, we compare the time complexity of some existing algorithms, including SYS-EDF, EEDS, DFR-EDF, and SSC [Awan and Petters 2012], with that of CI-EDF and CI-EDF^m proposed in this work.

Complexity of CI-EDF and CI-EDF^m. Since we adopt an EDF-similar schedule, it is clear that the time complexity of CI-EDF is determined by the computation of γ. In Algorithm 1, since $\sum_{k=n}^{q} W_{s_k}^o$ can be reused from last iteration, the complexity of calculating an *optional* task τ_i^o's workloads W_i^o (lines 19–21) is $O(n)$. Moreover, since there are at most n *optional* instances, calculating $\sum W_{s_i}^o$ takes at most $O(n^2)$ time. All

the other operations take at most $O(n)$ time. Overall, CI-EDF's time complexity, which is the same as computing γ, is $O(n^2)$.

From Algorithm 2, we can find that the time complexity of CI-EDF^m is mainly determined by the calculation of $WF(\tau_{i,j})$, $k_{i,j}$ and $\lambda_{i,j}$. Since a DCI cannot exceed $2 \cdot T_i$ (T_i is an arbitrary task's period), the elements in INS-List would not exceed $2 \cdot n$, where n is the task number of the task set. Due to the same reason, there are at most $2 \cdot n$ elements in TI-List. Hence, the complexities of calculating both $k_{i,j}$ and $\lambda_{i,j}$ are at most $O(n)$. Since there are totally m devices in the system, the complexity of calculating $WF(\tau_{i,j})$ is at most $O(mn)$. In summary, the time complexity of CI-EDF^m is $O(mn)$.

Note that during the execution of CI-EDF, slack reclaiming can be performed by simply revising the end Crenel Point of a Crenel Interval to the completion instant of last optional instance. So, the dynamic slack reclaiming scheme in CI-EDF has $O(1)$ time complexity. In CI-EDF^m, since there are only mandatory instances, dynamic slack reclaiming can be conducted automatically.

Complexity of Other Algorithms. Let m be the total number of devices, n be the total number of tasks in the system, and k be the number of operating frequencies that the processor can provide. For SYS-EDF, the system-wide optimal processor frequency f_{opt} is computed off-line, and the complexity of computing f_{opt} is $O(k2^m)$ [Cheng and Goddard 2005]. Moreover, SYS-EDF needs to calculate the next device request time for each device at runtime, which further needs to traverse through all the tasks, hence it has a time complexity of $O(mn)$. For EEDS, in order to get the device slack, it needs to calculate the job slack of every task instance that uses this device, so it also has a time complexity of $O(mn)$. For DFR-EDF, it needs to compute a set of forbidden regions off-line to make the task set schedulable under EDF, and this operation takes a time complexity of $O(n^2)$ [Devadas and Aydin 2010]. Moreover, in the online part of DFR-EDF, determining the time interval that a given device D_i can be put into sleep state at time t takes $O(nlogn)$ time. Then for m devices, DFR-EDF has a time complexity of $O(mnlogn)$. SSC proposed in Awan and Petters [2012] is a very time-efficient scheme. It has a time complexity of $O(n)$, because when calculating the next wake up time of a device, it only needs to traverse the tasks with higher priorities. The off-line part of SSC needs to calculate the device budget D_b for a given task set. To calculate D_b, it is sufficient to check the activation time of some jobs and the absolute deadlines of some jobs in the hyperperiod $H = LCM\{T_1, T_2, \ldots T_n\}$ [Baruah et al. 1990; Rahni et al. 2008]. Since the number of activation time and absolute deadlines is up-bounded by $\sum_{i=1}^{n} \frac{H}{T_i}$, the off-line part of SSC has a time complexity of $O(\sum_{i=1}^{n} \frac{H}{T_i})$. It is worth mentioning that SSC is based on the model that each device is associated with exactly one task, while SYS-EDF, EEDS, DFR-EDF, and CI-EDF^m all assume that a task can use multiple devices.

6. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the performance of the proposed CI-EDF and CI-EDF^m algorithms. We first describe the experimental setup in Section 6.1, and then discuss the experimental results of the single device model and the multiple device model in Sections 6.2 and 6.3, respectively.

6.1. Experimental Setup

We constructed a discrete-event simulator in C programming language to perform the evaluation. This approach is consistent with evaluation approaches adopted by several other researches for energy-aware scheduling [Devadas and Aydin 2008b; Cheng and Goddard 2006b, 2005]. The processor power is modeled according to Intel Xscale

Table III. Device Specifications

Device	P_a	P_s	P_{sw}	t_{sw}	Break-even time
Realtek Ethernet Chip	0.19 (W)	0.085 (W)	0.125 (W)	10 (ms)	20 (ms)
MaxStream wireless module	0.75 (W)	0.005 (W)	0.1 (W)	40 (ms)	80 (ms)
IBM Microdrive	1.3 (W)	0.1 (W)	0.5 (W)	12 (ms)	24 (ms)
SST Flash SST39LF020	0.125 (W)	0.001 (W)	0.05 (W)	1 (ms)	2 (ms)
SimpleTech Flash Card	0.225 (W)	0.02 (W)	0.1 (W)	2 (ms)	4 (ms)
Fujitsu 2300AT Hard disk	2.3 (W)	1.0 (W)	1.5 (W)	20 (ms)	40 (ms)

architecture specifications [Xu et al. 2004], and device specifications are adopted from Cheng and Goddard [2006a] and shown in Table III. In all experiments, we use randomly generated task sets to evaluate the performance of all algorithms. Specifically, the utilization range $[0, 1]$ is uniformly divided into 10 subintervals. For every subinterval, we generate 100 task sets; the task number in each task set is chosen randomly from the range of $[5, 20]$, to simulate their execution and record the system energy consumption. The task periods are chosen randomly from the range of [50ms, 1300ms].

In addition to our Crenel-Interval-based algorithms CI-EDF and CI-EDF^m, we implemented three state-of-the-art EDF-based system energy management algorithms as competitors:

(1) SYS-EDF [Cheng and Goddard 2005] is a system-level energy management scheme with both DVFS and DPM components. SYS-EDF performs DVFS by using the concept of energy-efficient scaling and has a relatively simple prediction-based DPM component that is applied at runtime.
(2) EEDS [Cheng and Goddard 2006b] is a DPM-only scheme for dynamic priority systems and EDF scheduling. There is no DVS component in EEDS and it is designed for systems where the device power dominates over that of the CPU.
(3) DFR-EDF [Devadas and Aydin 2010] is also a system-level energy management scheme. DFR-EDF is based on the extension of the DFR approach [Devadas and Aydin 2008b] to EDF scheduling.

We evaluated the performance of these five algorithms by the normalized energy consumptions (denoted by ES_{nor}), which is the amount of energy consumed under certain algorithms (DPM+DVFS or DPM-only) with respect to the case where no power-saving techniques are applied, that is, all the devices remain in the active state over the entire simulation. The normalized energy consumption is computed by the following equation:

$$ES_{nor} = \frac{E_{tot} \text{ with DPM-only (or DPM+DVFS)}}{E_{tot} \text{ without DPM or DVFS}}. \tag{17}$$

We examine the energy gains derived through DPM+DVFS or DPM-only policies under different break-even times. Specifically, when considering DPM+DVFS, we compare CI-EDF (or CI-EDF^m) with SYS-EDF and DFR-EDF, since EEDS does not include the DVFS component. When only considering DPM, we compare CI-EDF (or CI-EDF^m) with EEDS and DFR-EDF, since the DPM component in SYS-EDF is a relatively simple prediction-based one.

Scheduling Overhead. We did not measure scheduling overhead in a real system since all algorithms were evaluated with simulations. Instead, we used relative scheduling overhead to evaluate the scheduling overhead of CI-EDF and CI-EDF^m. Specifically, we compared the scheduling overhead of CI-EDF and CI-EDF^m with respect to EDF in our simulations. Note that some other works [Cheng and Goddard 2005, 2006b] also

use this method to measure scheduling overhead. The relative scheduling overhead, denoted by χ, is computed by

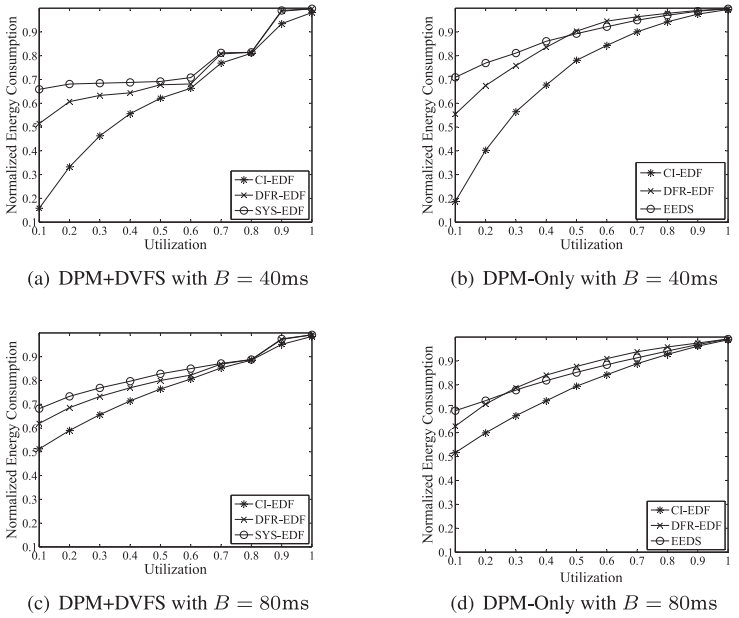
$$\chi = \frac{\text{schedule overhead with CI-EDF or CI-EDF}^m}{\text{schedule overhead with EDF}} - 1. \quad (18)$$

We use the time spent by the scheduler in a hyperperiod to measure the overhead of an algorithm. As well known, in a hyperperiod, the scheduler needs to be invoked many times. The scheduler of EDF has only one type of trigger event—choosing a task instance to execute when a task instance releases or completes. The scheduler of our CI-based methods also have only one type of trigger event, which is triggered at the start time point of each CI. For a task set we evaluated, at the beginning and end of a trigger event, we record two time instants, so as to calculate the time spent in this event. The accumulation of all these times in a hyperperiod is the scheduling overhead of the task set. We then use Equation (18) to calculate the relative scheduling overhead of this task set. For the 1,000 task sets randomly generated, we use the same method to get the relative scheduling overheads of them, and then take the average. In our experiments, the mean values of the relative scheduling overheads of CI-EDF and CI-EDF^m are 8.37% and 4.79%, respectively. Considering that the scheduling overhead of EDF is very low [Cheng and Goddard 2006b], these two overheads are affordable.

6.2. Experiment Results in Single Device Model

In the first set of experiments, we consider the case where there is only one device in the system and all the tasks need to access this device. Considering that when the break-even time is short, the device tends to be transitioned frequently and the differences between various algorithms are not significant, in our simulation, we choose two devices: *Fujitsu 2300AT Hard disk* and *MaxStream wireless module*, which have relatively long break-even times to be 40 and 80ms, respectively.

Figure 8(a) shows the relative performance of the evaluated schemes through DPM+DVFS with the device break-even time $B = 40$ms. As can be seen, the normalized energy consumption of all the tested algorithms are decreasing with the growth of the utilization. This is because both DVFS and DPM rely on slacks to perform frequency scaling or device state transition to save energy. With the growth of the system utilization, the number of the slacks is decreasing, hence the energy savings derived by the three schemes are also decreasing. Among the three schemes, CI-EDF provides clear gains over the other two throughout the entire spectrum, especially when the utilization is less than 60%. This is because, when the utilization is relatively small, after all the schemes get their optimal processor frequency, there are still many slacks that can be used for DPM. In this case, the system energy savings are mainly dependent on DPM. Since CI-EDF can merge small slacks to form big ones, it can greatly enhance the DPM effectiveness, and thus get promising energy savings. On the contrary, the DPM policy of SYS-EDF is a relatively simple look-ahead-based prediction scheme, so it has a relatively poor performance. DFR-EDF aims at explicitly and periodically creating device forbidden regions to put the devices into sleep state. Although it can achieve a better performance than SYS-EDF, on one hand, the number of the slacks may increase due to the preemption of the forbidden region. Clearly, more slacks means more device state transitions (if possible). Since the transition itself also consumes time and energy, this will inevitably reduce the system energy savings, especially when the utilization is relatively small; on the other hand, the static feasibility test of DFR is a utilization-based one. When the utilization is high, the test tends to fail. These two factors limit the performance gains of DFR-EDF. Note that in Figure 8(a), when the utilization exceeds 60%, the three curves are not smooth anymore; this is because the processor frequencies we used are discrete. Also note that at some high utilization levels, two



(a) DPM+DVFS with $B = 40$ms (b) DPM-Only with $B = 40$ms

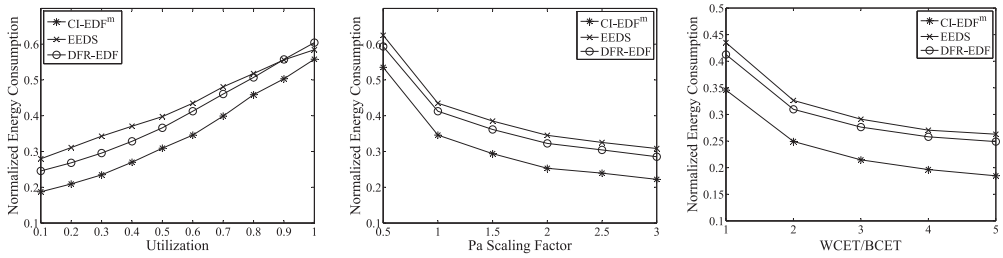
(c) DPM+DVFS with $B = 80$ms (d) DPM-Only with $B = 80$ms

Fig. 8. Impact of system utilization in single device model.

or three schemes get the same energy savings. This is because at these utilizations, the two or three schemes choose the same frequency and thus have the same energy savings by DVFS, and moreover, these slacks are not long enough for performing DPM.

When only the DPM policy is applied, the superiority of CI-EDF compared to DFR-EDF and EEDS is more significant, as shown in Figure 8(b). Without DVFS, the system executes all workloads with the maximum processor frequency, hence the energy consumption of the processor is proportional to the utilization. With the increase of the system utilization, there are less slacks that can be used to transition the devices into sleep states, so the energy consumption of the devices is also proportional to the system utilization. When $U < 0.48$, DFR-EDF performs better than EEDS, but with the increase of the utilization, EEDS can outperform DFR-EDF. This is because DFR-EDF relies on a utilization-based feasibility test to enforce DFRs to perform DPM. As the system utilization increases, less task sets can pass the feasibility test, and the performance of DFR-EDF degrades significantly.

Figures 8(c) and 8(d) show the relative performance of the the four schemes with $B = 80$ms. Compared with Figures 8(a) and 8(b), it can be seen that all the four schemes exhibit performance degradation when the device break-even time becomes larger, but CI-EDF can still outperform the other three schemes significantly. For CI-EDF, after merging small slacks to get big ones, since the break-even time becomes larger, less slacks can be used to perform DPM. Therefore, the energy consumption increases. For DFR-EDF, when the device break-even time becomes larger, the length of the forbidden region is also enlarged. In order to guarantee the schedulability of task set, the period of the forbidden region will be extended, which in turn means less forbidden regions can be used to perform DPM. For SYS-EDF, it neither creates long slacks, nor decreases the number of the slacks. When the device break-even time becomes larger, it has less slacks to transition devices into low-power state. For EEDS, it also relies on the system utilization to compute the runtime budget for each instance, and then delays



(a) Impact of System Utilization (b) Impact of Device Power (c) Impact of Workload Variability

Fig. 9. Simulation results with DPM-only in multiple device model.

the execution of the instance to create long slack. When the device break-even time becomes larger, less slacks can be used to perform device state transition.

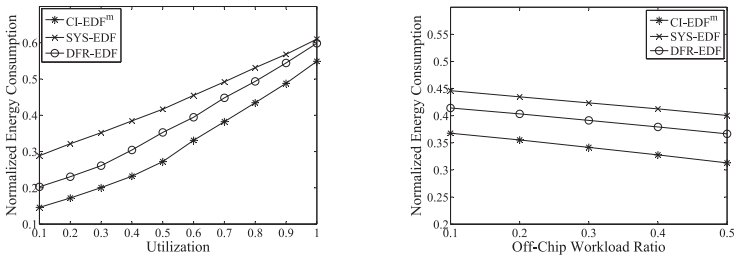
6.3. Experiment Results in Multiple Device Model

In this set of experiments, we consider a more general case where there are multiple devices in the system. Each task may access multiple devices, and each device may also be accessed by multiple tasks. Specifically, we consider the experimental settings where each task uses 0–3 devices, which are randomly selected from a list that includes *IBM Microdrive, SST Flash, Realtek Ethernet Chip*, and *SimpleTech Flash Card*. Note here the device specifications are adopted from Cheng and Goddard [2006a].

We first present and discuss the results where only the DPM policy is applied. Figure 9(a) shows the relative performance as a function of system utilization with worst-case workloads. In this figure, we can see that CI-EDF^m provides clear gains over the other two schemes throughout the entire spectrum. This is mainly because CI-EDF^m is based on Device Crenel Intervals to help schedule the task set. In this way, it can merge small slacks to form big ones, which help to transition devices into low-power state. DFR-EDF performs better than EEDS when $U \leq 0.9$. But when $U > 0.9$, EEDS is able to outperform DFR-EDF. This is because DFR-EDF relies on DFRs to perform the DPM operation. Before inserting into the system, the DFRs need to pass a utilization-based schedulability test. With increasing system utilization, less DFRs can pass the schedulability test and thus the performance of DFR-EDF degrades at high utilization values.

Figure 9(b) shows the impact of varying the power characteristics of a system component with worst-case workloads. In this experiment, the system utilization is fixed to be $U = 0.6$. We multiply the active power (P_a) of all devices by a certain scaling factor and recompute the device break-even times, while the other devices and processor specifications remain unchanged. For each scaling factor, we measure the energy consumption of the three schemes. Apparently, the higher the scaling factor, the more dominant the device power. Hence, as P_a scales up, the energy consumption of all schemes decreases. Since CI-EDF^m preserves the best DPM effectiveness, the performance gaps between CI-EDF^m and the other two schemes become more significant with increasing P_a scaling factors.

Figure 9(c) shows the relative performance of the schemes under variability in the actual workload. This variability is controlled by modifying the worst-case to best-case execution time ratio, $\frac{WCET}{BCET}$. The actual workload of each task instance is determined randomly in the range of $[BCET, WCET]$ at its release time. In this experiment, the system utilization is fixed to be $U = 0.6$. It can be seen that the energy consumption of all three schemes decreases with the increase of $\frac{WCET}{BCET}$. This is because with larger



(a) Impact of System Utilization (b) Impact of Workload Variability

Fig. 10. Simulation results with both DVFS and DPM in multiple device model.

$\frac{WCET}{BCET}$, more dynamic slacks can be used for additional energy savings. Since CI-EDF^m can utilize the dynamic slacks to help create bigger device slacks, the performance superiority of CI-EDF^m to the other two schemes is more significant with increasing $\frac{WCET}{BCET}$.

Now we proceed to discuss the results where both DPM and DVFS are applied, as shown in Figure 10. Figure 10(a) shows the relative performance as a function of system utilization with worst-case workloads. It can be seen that CI-EDF^m consistently outperforms DFR-EDF throughout the entire spectrum. This is because CI-EDF^m and DFR-EDF have similar DVFS components, but CI-EDF^m has a stronger DPM component than DFR-EDF. Also, it can be observed that DFR-EDF performs better than SYS-EDF throughout the entire spectrum. This is because DFR-EDF has a more powerful DPM component than SYS-EDF. Moreover, in the DVFS component, DFR-EDF not only uses the system energy-efficient scaling technique given in Cheng and Goddard [2005], but has its own efficient frequency, which is calculated off-line.

Figure 10(b) shows the impact of varying the off-chip workload ratio $\frac{y_i}{C_i}$ for τ_i, with worst-case workloads. In this experiment, the system utilization is fixed to be $U = 0.6$. As can be observed from this figure, the energy consumption of all three schemes decreases with increasing off-chip workload ratio. This is because, with the growth of $\frac{y_i}{C_i}$, the execution frequency of τ_i can be chosen from a bigger range. Thus, it is easier to find an efficient execution frequency. CI-EDF^m outperforms the other two schemes consistently during the whole range, because all the three schemes use a similar DVFS component, while CI-EDF^m has the most powerful DPM component.

7. CONCLUSION

Energy management is one of the key issues in the design of modern real-time mobile and embedded systems. In this article, we considered how to minimize the system-level energy consumption for period real-time tasks. Specifically, we addressed the online-DPM problem by introducing Crenel Interval (CI), in which we merge all the small slacks to get a big continuous one. In this way, we can get the best DPM effectiveness. First, targeting at the single device model, we proposed the CI-EDF algorithm, and proved the correctness of it on scheduling all task instances to meet their deadlines. Then by considering a more general multiple device model, we proposed the CI-EDF^m algorithm, which combines the CI-based scheme and the *runtime*-based delay calculation method. We also showed how to integrate DVFS with CI-EDF and CI-EDF^m to further reduce the system energy consumption. Finally, the experimental study demonstrates the effectiveness and efficiency of the proposed methods, as compared to existing approaches with comparable quality.

For future work, we intend to investigate the online DPM problem with a more general task model where task periods are not equal to their deadlines, by the techniques proposed in this article.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive and helpful comments.

REFERENCES

Tarek A. AlEnawy and Hakan Aydin. 2005. Energy-aware task allocation for rate monotonic scheduling. In *IEEE Real Time and Embedded Technology and Applications Symposium*. 213–223.

Muhammad Ali Awan and Stefan M. Petters. 2012. Online intra-task device scheduling for hard real-time systems. In *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*. 48–56.

Hakan Aydin, Vinay Devadas, and Dakai Zhu. 2006. System-level energy management for periodic real-time tasks. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. 313–322.

Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejia-Alvarez. 2001. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'01)*. 95–105.

Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. 2004. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers* 53, 5 (2004), 584–600.

Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. 1990. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems* 2, 4 (1990), 301–324.

Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Proactive speed scheduling for real-time tasks under thermal constraints. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. 141–150.

Jian-Jia Chen, Chuan-Yue Yang, Hsueh-I Lu, and Tei-Wei Kuo. 2008. Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. 13–23.

Hui Cheng and Steve Goddard. 2005. Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *Proceedings of the Workshop on Power Aware Real-time Computing*, Vol. 2. IEEE.

Hui Cheng and Steve Goddard. 2006a. Online energy-aware I/O device scheduling for hard real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 1055–1060.

Hui Cheng and Steve Goddard. 2006b. An online energy-efficient I/O device scheduling algorithm for hard real-time systems with non-preemptible resources. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 251–260.

Vinay Devadas and Hakan Aydin. 2008a. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *Proceedings of the 8th ACM International Conference on Embedded Software*. 99–108.

Vinay Devadas and Hakan Aydin. 2008b. Real-time dynamic power management through device forbidden regions. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*. 34–44.

Vinay Devadas and Hakan Aydin. 2010. DFR-EDF: A unified energy management framework for real-time systems. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*. 121–130.

Vinay Devadas and Hakan Aydin. 2012. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *IEEE Transactions on Computers* 61, 1 (2012), 31–44.

Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Thermal-aware global real-time scheduling on multicore systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. 131–140.

Marco E. T. Gerards and Jan Kuper. 2013. Optimal DPM and DVFS for frame-based real-time systems. *ACM Transactions on Architecture and Code Optimization (TACO)* 9, 4 (2013), 41.

Ravindra Jejurikar and Rajesh Gupta. 2005a. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the 42nd Annual Design Automation Conference*. ACM, 111–116.

Ravindra Jejurikar and Rajesh Gupta. 2005b. Energy aware non-preemptive scheduling for hard real-time systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS'05)*. 21–30.

Ravindra Jejurikar and Rajesh Gupta. 2006. Energy-aware task scheduling with task synchronization for embedded real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 6 (2006), 1024–1037.

Ravindra Jejurikar, Cristiano Pereira, and Rajesh Gupta. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st Annual Design Automation Conference*. ACM, 275–280.

Fanxin Kong, Yiqun Wang, Qingxu Deng, and Wang Yi. 2010. Minimizing multi-resource energy for real-time systems with discrete operation modes. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS'10)*. IEEE, 113–122.

Etienne Le Sueur and Gernot Heiser. 2010. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the International Conference on Power Aware Computing and Systems*. USENIX Association, 1–8.

Yann-Hang Lee, Krishna P. Reddy, and C. Mani Krishna. 2003. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*. IEEE, 105–112.

Jianjun Li, LihChyun Shu, Jian-Jia Chen, and Guohui Li. 2013. Energy-efficient scheduling in nonpreemptive systems with real-time constraints. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43, 2 (2013), 332–344.

Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. 2002. Power-aware operating systems for interactive systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 10, 2 (2002), 119–134.

Bren Mochocki, Xiaobo Sharon Hu, and Gang Quan. 2007. Transition-overhead-aware voltage scheduling for fixed-priority real-time systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 12, 2 (2007), 11.

Padmanabhan Pillai and Kang G. Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, Vol. 35. 89–102.

Ala Qadi, Steve Goddard, and Shane Farritor. 2003. A dynamic voltage scaling algorithm for sporadic tasks. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS'03)*. 52–62.

Gang Quan and Xiaobo Sharon Hu. 2003. Minimal energy fixed-priority scheduling for variable voltage processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 8 (2003), 1062–1071.

Gang Quan, Linwei Niu, Xiaobo Sharon Hu, and Bren Mochocki. 2004. Fixed priority scheduling for reducing overall energy on variable voltage processors. In *Proceedigs of the 25th IEEE International Real-Time Systems Symposium*. 309–318.

Ahmed Rahni, Emmanuel Grolleau, and Michael Richard. 2008. Feasibility analysis of non-concrete real-time transactions with edf assignment priority. In *16th International Conference on Real-Time and Network Systems (RTNS'08)*.

Saowanee Saewong and Ragunathan Rajkumar. 2003. Practical voltage-scaling for fixed-priority rt-systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*. 106–114.

Vishnu Swaminathan and Krishnendu Chakrabarty. 2003. Energy-conscious, deterministic I/O device scheduling in hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 7 (2003), 847–858.

Vishnu Swaminathan and Krishnendu Chakrabarty. 2005. Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)* 4, 1 (2005), 141–167.

Vishnu Swaminathan, Krishnendu Chakrabarty, and Sundaraja Sitharama Iyengar. 2001. Dynamic I/O power management for hard real-time systems. In *Proceedings of the 9th International Symposium on Hardware/Software Codesign*. ACM, 237–242.

Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. 1996. Scheduling for reduced CPU energy. In *Mobile Computing*. Springer, 449–471.

Ruibin Xu, Chenhai Xi, Rami Melhem, and Daniel Moss. 2004. Practical pace for embedded systems. In *Proceedings of the 4th ACM International Conference on Embedded Software*. 54–63.

Frances Yao, Alan Demers, and Scott Shenker. 1995. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. IEEE, 374–382.

Fan Zhang and Samuel T. Chanson. 2004. Blocking-aware processor voltage scheduling for real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)* 3, 2 (2004), 307–335.

Ying Zhang and Krishnendu Chakrabarty. 2004. Dynamic adaptation for fault tolerance and power management in embedded real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)* 3, 2 (2004), 336–360.

Dakai Zhu and Hakan Aydin. 2009. Reliability-aware energy management for periodic real-time tasks. *IEEE Transactions on Computers* 58, 10 (2009), 1382–1397.

Dakai Zhu, Rami Melhem, and Daniel Mossé. 2004. The effects of energy management on reliability in real-time embedded systems. In *IEEE/ACM International Conference on Computer Aided Design*. 35–40.

Jianli Zhuo and Chaitali Chakrabarti. 2005. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 42nd Design Automation Conference*. IEEE, 628–631.

Received April 2014; revised January 2015; accepted March 2015