Energy-Efficient Scheduling in Nonpreemptive Systems With Real-Time Constraints

Jianjun Li, LihChyun Shu, *Member, IEEE*, Jian-Jia Chen, *Member, IEEE*, and Guohui Li

***Abstract*—In the past decade, the development of mobile and embedded systems has demanded energy efficiency for improving the lifetime of embedded devices. To avoid preemption overhead or ease timing verification, nonpreemptive scheduling has been deemed useful or necessary in meeting system timing requirements for certain applications built on embedded devices. In this paper, our aim is to design nonpreemptive scheduling algorithms that ensure timing correctness and optimize energy consumption on a processor with variable speeds. We propose a representative algorithm, ISA, which can produce lower speeds for a variety of nonpreemptive task sets than other comparable methods, and hence resulting in significant energy savings. When combined with a selective frequency-inheritance policy we design to efficiently determine if processor speedup can be disabled without jeopardizing any task deadlines, ISA can achieve even larger gains, up to 30% reduction in energy consumption. Finally, we propose a dynamic slack reclamation policy built on ISA, namely ISA-DR, which can result in additional energy savings when a task consumes less than its worst-case execution time.**

***Index Terms*—Energy efficiency, fixed priority, nonpreemptive scheduling, real-time system.**

I. Introduction

POWER CONSUMPTION has become one of the most important issues when it comes to designing many mobile and embedded real-time applications. When it is necessary to trade system performance for reduced power consumption, the system can exploit *dynamic voltage scaling* (DVS) to change the supply voltage dynamically or *dynamic threshold voltage scaling* by controlling the body bias voltage to change the threshold voltage dynamically. Different supply voltages or threshold voltages result in different processor speeds. Therefore, how to choose the proper speeds, along with the supply voltages and threshold voltages, is of importance for energy reduction and meeting system timing requirements.

In this paper, we consider energy-efficient scheduling of nonpreemptive tasks on uniprocessor systems that support dynamic speed adjustment. Although a large body of past research has focused on preemptive scheduling [4], [21], [25], nonpreemptive scheduling is still attractive in different application domains where properties of device hardware and software make preemption either prohibitively expensive or just impossible [10]. Despite their inherent limitations, nonpreemptive scheduling algorithms are easier to implement, have lower runtime overhead, require less memory, and eliminate the need for synchronization and its associated overhead [10], [12]. Nonpreemption also helps preserve program locality, which in turn makes programs more amenable to worst-time execution time analysis [23]. As a result, nonpreemptive scheduling has been adopted in many avionics applications [18] as well as in embedded systems, particularly in small embedded devices with limited memory capacity [2], [16], [19], [32].

To the best of our knowledge, no specific algorithm has been developed for scheduling of fully nonpreemptive fixed priority tasks for energy minimization under real-time constraints in the past. The most relevant ones are the dual-speed (DS) algorithm due to Zhang and Chanson [34] and the uniform slowdown algorithm with frequency inheritance (USFI) proposed by Jejurikar and Gupta [13]. Both algorithms are designed for tasks that need synchronization and hence have critical sections. In this paper, we propose individual speed algorithm (ISA), a novel scheme that computes one speed for each individual task in a nonpreemptive task set. We address the challenging issue of figuring out the exact workload of higher priority tasks on each nonpreemptive task. The exact workload enables us to calculate optimal processor speeds which give optimal energy consumption while satisfying the given timing constraint. ISA achieves considerable energy savings compared to existing methods with comparable quality, particularly at high processor utilization. Another scheme we devise to cowork with ISA is *selective FI* (SFI) which efficiently determines if processor speedup can be disabled without jeopardizing any task deadlines whenever a task is blocked. We show that further improvement (20% to 30% reduction in energy consumption) can be obtained when ISA is combined with the SFI policy, at the cost of slight runtime overhead. Considering that the task actual execution time is usually less than its worst-case execution time (WCET), we further propose a dynamic slack reclamation scheme based on ISA, which is named ISA-DR, for more energy savings. Experimental study shows that up to 30% energy savings can be achieved by ISA-DR compared to ISA. In addition to their primary role in reducing processor energy consumption for real-time applications, we also demonstrate an additional use of our algorithms to trade off energy consumption against transmission delay in saving communication energy.

Manuscript received April 15, 2011; revised January 20, 2012; accepted April 7, 2012. Date of publication September 12, 2012; date of current version February 12, 2013. This work was supported in part by the National Science Foundation of China [No. 61173049], the Research Fund for the Doctoral Program of the Ministry of Education of China [No. 20090142110023], a program for New Century Excellent Talents in University supported by MoE, China [No. NCET-09-0384], and in part by grant NSC 100-2221-E-006-159. This paper was recommended by Associate Editor W. Pedrycz.

J. Li and G. Li are with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan 430074, China (e-mail: jianjunli@smail.hust.edu.cn; guohuili@mail.hust.edu.cn).

L. Shu is with National Cheng Kung University, Tainan 701, Taiwan and also with Chang Jung Christian University, Tainan 71101, Taiwan (e-mail: shulc@mail.ncku.edu.tw).

J.-J. Chen is with Karlsruhe Institute of Technology (KIT), 76133 Karlsruhe, Germany (e-mail: j.chen@kit.edu).

Digital Object Identifier 10.1109/TSMCA.2012.2199305

The remainder of this paper is organized as follows: Section II briefly discusses related work. Section III describes task and power models along with some assumptions we make. Section IV details the motivation and design of the ISA algorithm. In the same section, the SFI policy applied to ISA is also introduced. In Section V, the dynamic slack reclamation policy is presented. The experimental results are discussed in Section VI. Section VII shows an additional application of our algorithms to energy-aware communication systems, and finally, Section VIII concludes the paper with some remarks.

II. Related Work

Power consumption has become a critical problem in embedded and mobile systems with real-time constraint in recent years. Researchers have tackled energy reduction on a processor with variable speeds for both periodic and aperiodic real-time tasks [4], [21], [25], [31], as well as tasks with critical sections [13], [34], scheduling using a hybrid of the slowdown and shutdown strategies [14], [22], energy reduction based on slack reclamation [4], [11], [20], energy-aware scheduling with reliability requirements [35]–[37] (or on fault-tolerance systems [15]), energy-efficient scheduling on wireless networks [5], [7], [24], multiprocessor energy-efficient scheduling [1], [6], and energy-aware scheduling at a broader system level [3], [38]. Note that all the aforementioned studies focused on preemptive scheduling.

Hong *et al.* [9] first reported superior results yielded by nonpreemptive scheduling policies on power minimization of variable-voltage core-based systems. The authors proposed a synthesis approach that explores static scheduling algorithms, determines the cache size and configuration, and selects the processor core. The effectiveness of the approach was demonstrated on a variety of industrial-strength multimedia and communication applications. Zhang and Chanson [33] addressed energy-efficient scheduling of nonpreemptive periodic tasks on uniprocessor systems and presented the DS algorithms for dynamic priority [earliest deadline first (EDF)-based)] systems. DS computes two speeds for the task system consisting of periodic tasks with nonpreemptive critical sections. A low speed L is obtained by assuming the task set is independent, while a high speed H is computed by taking the blocking time due to nonpreemption of critical sections into consideration. The algorithm executes each task at speed L for most of the time and switches to speed H only when some task is blocked. Jejurikar and Gupta [12] proposed using the optimal feasibility test for EDF scheduling of nonpreemptive periodic tasks in [10] to obtain a H speed that is lower than the one computed in [33].

Swaminathan and Chakrabarty [28] presented a novel low-energy earliest deadline first (LEDF) scheduling algorithm for nonpreemptive periodic task sets, and an implementation of LEDF in RT-Linux is described in [29]. Nonpreemptive scheduling is considered necessary for some energy-intensive applications, e.g., in executing wireless packet transmission tasks. An example can be found in [16], in which the authors addressed the problem of power-aware scheduling of nonpreemptive aperiodic task sets, and developed a DVS algorithm by exploiting the structure of optimal sample paths. Note that all the studies mentioned above considered dynamic priority scheduling.

For fixed-priority systems, Zhang and Chanson proposed the DS algorithm [34] for RM scheduling. Jejurikar and Gupta [13] proposed the USFI, which assumes the same task and processor model as [34] and determines a uniform slowdown for each task in the task set. While DS assumes fully nonpreemptive critical sections, USFI can work together with well-known synchronization protocols such as PCP and SRP to bound task blocking times and to reduce energy consumption. The FI policy in USFI requires that when blocking occurs, if the blocked task has a higher slowdown factor, this slowdown factor is inherited by the blocking task. This policy is important in the algorithm's bounding of maximum blocking time.

III. Model and Assumptions

We consider a set of n nonpreemptively periodic tasks that are scheduled by the deadline-monotonic (DM) algorithm on a uniprocessor system. That is, the tasks with shorter deadlines have higher priority. Without loss of generality, we index the n periodic tasks from the highest priority to the lowest priority and denote them by $\tau_1, \tau_2, \ldots, \tau_n$. When the processor becomes available, the scheduler chooses the task instance with the highest priority, and this task instance is executed until it finishes. Each task τ_i is characterized by three parameters (T_i, D_i, C_i), where T_i, D_i, and C_i represent task τ_i's period, relative deadline, and worst-case computation time (WCET), respectively. For each τ_i, we assume $D_i \leq T_i$, and C_i is estimated at maximum processor speed. Each instance of a task is called a job, and the kth job of task τ_i is denoted as $\tau_{i,k}$. As in [12], the overhead of context switching is included in task computation times and the overhead of task scheduling is assumed to be zero. Associated with the task set, the system utilization is defined as $U = \sum_{i=1}^{n} C_i/T_i$. A job is said to be *blocked* if it is waiting in the ready queue for the completion of the currently executing lower priority job, and the lower priority executing job is called the *blocking* job.

For ease of presentation, we will first focus on systems with continuous frequencies/speeds and negligible frequency transition overhead, deferring discussion of systems with discrete speeds and non-negligible frequency transition overhead to Section IV-D. We assume that the operating frequency of the processor can be scaled within $[f_{min}, f_{max}]$. For convenience, we define a *slowdown factor* η (the normalized processor speed) as the ratio of the current operating frequency to the maximum frequency. Hence, we have slowdown factors varying in the interval $[\eta_{min}, 1]$, where $\eta_{min} = f_{min}/f_{max}$. We will use the terms frequency and speed interchangeably in the following discussion, if no confusion arises. For the simplicity of designing schedulers, we further assume that a frequency change can occur only when a job begins execution or when a higher priority job is blocked.

Suppose that the power consumption is P_I when the system is idle and the power consumption is $P_I + P_d(f)$ when the system is executing a task at frequency f (Note that $P_d(f)$ could also be task-dependent), we assume that running at a higher frequency in the range of $[f_{min}, f_{max}]$ will result in

higher energy consumption for execution. That is, $P_d(f)/f$ is an increasing function of f. In particular

$$P_d(f) = P_{ind} + P_{dep}(f)$$

P_{ind} is contributed by the components of memory and processor power that can be efficiently removed by putting the system to sleep (or standby), and is independent of the supply voltage and frequency, while P_{dep} is contributed by the switching power for charging and discharging the load capacitance, which can be represented as $P_{dep}(f) = C_{ef}V_{dd}^2 f$, where C_{ef} and V_{dd} denote the effective switch capacitance and the supply voltage, respectively. Moreover, as the frequency f is approximately proportional to the supply voltage V_{dd}, the power consumption $P_d(f)$ is roughly proportional to the cube of the frequency. Since $P_d(f)/f$ is a convex function where f_{ee} has the minimum $P_d(f)/f$ in the range of $[f_{min}, f_{max}]$, we have to set f_{min} as f_{ee} to ensure that only energy-efficient operation frequencies are selected.

IV. Individual-Speed Algorithm

Section IV-A introduces the motivation of ISA. We describe the design detail of ISA in Section IV-B. Section IV-C considers a policy, namely SFI, to reduce the frequency of processor speedup at task blocking times with ISA. Remarks for practical systems and a note on DS are presented in Section IV-D.

A. Motivation

Algorithm DS [34] computes two speeds for the task system consisting of periodic tasks with nonpreemptive critical sections that are executed on a variable-speed uniprocessor, while algorithm USFI [13] determines a uniform slowdown for each task in the task set. Both DS and USFI can be applied to nonpreemptive scheduling by treating each task as one critical section. However, both approaches would overestimate preemption times on nonpreemptive tasks, and hence, result in task slowdown factors higher than necessary. We will give an example later to illustrate this problem. Since DS only computes two static speeds for the entire task set and the improvement on it is trivial, a short discussion of this algorithm is postponed to Section IV-D. In the following, we first review how USFI works, and then present an example to illustrate why USFI is not energy-efficient for fully nonpreemptive task set.

USFI iteratively computes slowdown factors for all tasks, from highest to lowest priority. An index q, initialized to 1, is used to record the fact that at any instant in time, the slowdown factors for tasks τ_1 to τ_{q-1}, denoted $\eta_1, \eta_2, \ldots, \eta_{q-1}$, have been determined. During each iteration, it first finds candidate slowdown factors for tasks from τ_q to τ_n, i.e., $\eta_q, \eta_{q+1}, \ldots, \eta_n$. Suppose $\eta_m (q \le m \le n)$ is the maximum of $\eta_q, \eta_{q+1}, \ldots, \eta_n$.[1] USFI selects η_m as the slowdown factor for all tasks from τ_q to τ_m. It then updates q to be $m+1$ and continues the next iteration until all n tasks have slowdown factors determined. When determining candidate slowdown factor η_i for each task in $\{\tau_q, \tau_{q+1}, \ldots, \tau_n\}$ at each iteration, USFI uses the following time-demand-based feasibility test (a direct extension of the schedulability test in [27]):

$$\left(\sum_{1\le r<q} \frac{C_r}{\eta_r}\left\lceil \frac{S_{i,j}}{T_r}\right\rceil\right) + \frac{1}{\eta_{i,j}}\left(B_i + \sum_{q\le p\le i} C_p\left\lceil \frac{S_{i,j}}{T_p}\right\rceil\right) = S_{i,j}. \quad (1)$$

$S_{i,j}$ is a scheduling point of task τ_i, and $\eta_{i,j}$ is the candidate slowdown factor corresponding to $S_{i,j}$. $S_{i,j}$ is a member of S_i defined as: $S_i = \{(t \in S) \wedge (t < D_i)\} \cup \{D_i\}$, where $S = \{kT_j | j = 1, \ldots, i; k = 1, \ldots, \lfloor T_i/T_j \rfloor\}$. $B_i = \max_{j\in lp(i)}\{C_j\}$ is the blocking factor due to nonpreemptability, where $lp(i)$ is the set of tasks having lower priority than τ_i's. When there are more than one scheduling point for task τ_i, i.e., more than one candidate slowdown factor, to achieve the best energy savings, USFI chooses the minimum candidate slowdown factor as the slowdown factor for τ_i, i.e, $\eta_i = \min_j(\eta_{i,j})$.

Whenever blocking occurs, USFI adopts the FI policy to change the processor speed immediately by employing the slowdown factor of the blocked job in order to ensure task deadlines. In all the other cases, each job will execute at its slowdown factor to save energy. Note that the slowdown factors computed by USFI are in a nonincreasing order.

The following example explains why USFI has room for improvement when applied to fully nonpreemptive task scheduling and motivates us to push forward.

Example 1: Consider the following three periodic tasks $\tau_1 = (5, 5, 1)$, $\tau_2 = (10, 10, 2)$, $\tau_3 = (20, 20, 1)$ specified with their periods, relative deadlines, and WCETs. The slowdown factors computed by USFI are $\eta_1 = 0.6$, $\eta_2 = 0.45$, and $\eta_3 = 0.225$, respectively.

Now we look into how USFI works. Suppose we have finished the first iteration of the algorithm and determined η_1 to be 0.6, and we continue to determine η_2 at the second iteration. Consider how USFI determines η_{22} for S_{22} (one of the scheduling points of τ_2) by applying (1) to get the following equation:

$$\frac{1}{0.6}\left\lceil \frac{10}{T_1}\right\rceil + \frac{1}{\eta_{22}}\left(1 + \sum_{2\le p\le 2} C_p\left\lceil \frac{10}{T_p}\right\rceil\right) = 10. \quad (2)$$

In (2), the workload from τ_1 has been counted for two times (i.e., $\lceil 10/5 \rceil$), which means that in the worst case, τ_1 can execute twice before τ_2 starts its execution. However, actually, this is an overestimation. To see this, we assume $\tau_{1,1}$ and $\tau_{2,1}$ arrive at time $t = 0$ while $\tau_{3,1}$ arrives just a little earlier. It is not hard to see that this is a case when τ_2 has its longest response time. Initially, USFI schedules $\tau_{3,1}$ to execute. $\tau_{1,1}$ and $\tau_{2,1}$ arrive immediately while $\tau_{3,1}$ is executing. Since $\tau_{1,1}$ (as well as $\tau_{2,1}$) is blocked, the processor speed is increased to $\eta_1 = 0.6$. At $t = 5/3$, $\tau_{3,1}$ is finished, and $\tau_{1,1}$ begins execution still at speed $\eta_1 = 0.6$. At $t = 10/3$, $\tau_{1,1}$ finishes and $\tau_{2,1}$ finally gets its chance to execute at speed η_2. Fig. 1 shows the schedule during the first 10 time units (the lifetime of $\tau_{2,1}$).

As one can see from Fig. 1, although τ_1 is released twice during the lifetime of $\tau_{2,1}$, it executes only once before $\tau_{2,1}$ starts. This is because with nonpreemptive scheduling, once

[1] If there are more than one task with maximal value, then USFI selects the slowdown factor of the task with the largest index.

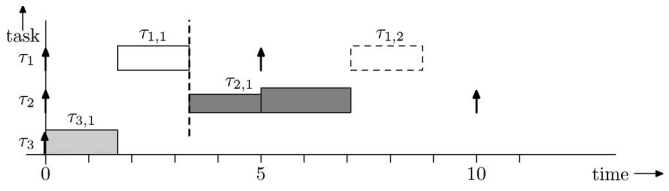

Fig. 1. τ_1 executes just once before τ_2 starts execution.

a task has seized the control of the processor, no other tasks can preempt it until it finishes. Hence, the workload from τ_1 should be counted just once when determining a candidate slowdown factor for τ_2. Since USFI is a scheduling algorithm for tasks with nonpreemptive critical sections, when adapting to fully nonpreemptive tasks, its calculation of slowdown factors must be reconsidered. Otherwise, the algorithm will be energy *inefficient*.

B. Designing ISA

Based on the above discussion, our main challenge is to come up with a more precise feasibility test formula for the calculation of slowdown factors. Specifically, our goal is to calculate new slowdown factors such that all task deadlines are guaranteed and more energy savings can be achieved. Our approach is to check, for every task τ in a given task set, whether τ can execute at a slowdown factor determined by USFI and finish some time before one of its scheduling points, even after τ experiences worst-case blocking and all task instances with priority higher than τ's have completed. If this condition holds for a scheduling point of τ, say S, other than the one that USFI uses to determine τ's slowdown, then we can compute a more precise workload on τ from higher priority tasks with respect to S in the schedulability test. From that, we obtain a new slowdown for τ.

The skeleton of the proposed ISA algorithm is given in Algorithm 1. We use $\overline{\eta}_i$ to denote the new candidate slowdown factor obtained for each task τ_i by using our approach, to distinguish from η_i that is obtained using USFI. At each iteration, the first $q-1$ ($q=1$ in the beginning) tasks already have their new slowdown factors determined. Now we need to calculate $\overline{\eta}_i$ for every task $\tau_i (q \le i \le n)$. For each τ_i, we first use (3) (obtained from (1) with η_r replaced by $\overline{\eta}_r$) and (4) to compute an intermediate slowdown factor $\tilde{\eta}_i$

$$\left(\sum_{1\le r<q} \frac{C_r}{\overline{\eta}_r}\left\lceil\frac{S_{i,j}}{T_r}\right\rceil\right)+\frac{1}{\eta_{i,j}}\left(B_i+\sum_{q\le p\le i} C_p\left\lceil\frac{S_{i,j}}{T_p}\right\rceil\right)=S_{i,j} \tag{3}$$

$$\tilde{\eta}_i = Min_{S_{i,j}\in S_i}(\eta_{i,j}). \tag{4}$$

Suppose $\tilde{\eta}_i$ is obtained from $\eta_{i,l}$ corresponding to $S_{i,l}$ that belongs to S_i, i.e., $\tilde{\eta}_i = Min_{S_{i,j}\in S_i}(\eta_{i,j}) = \eta_{i,l}$. Now, consider the following Inequality for the scheduling points in S_i:

$$\left(\sum_{1\le r<q} \frac{C_r}{\overline{\eta}_r}\left\lceil\frac{S_{i,j}}{T_r}\right\rceil\right)+\frac{1}{\tilde{\eta}_i}\left(B_i+\sum_{q\le p<i} C_p\left\lceil\frac{S_{i,j}}{T_p}\right\rceil\right)<S_{i,j}. \tag{5}$$

Algorithm 1 The ISA algorithm

```
1: Given tasks sorted in nondecreasing order of periods;
2: q = 1; //Initialization
3: while q ≤ n do
4:   for i = q; i ≤ n; i++ do
5:     Compute η̃_i by (3) and (4);
6:     S̄_i = {S_{i,k} | S_{i,k} ∈ S_i, and Inequality (5) holds for S_{i,k}};
7:     for each S_{i,k} ∈ S̄_i do
8:       Compute η̄_{i,k} by (6), and η̿_{i,k} by (7);
9:       η'_{i,k} = Max(η̄_{i,k}, η̿_{i,k});
10:    end for
11:    η̄_i = Min_{S_{i,k}∈S̄_i}(η'_{i,k});
12:  end for
13:  η̄_m = max_{i=q}^{n}(η̄_i); //Choose the maximum candidate
     slowdown factor
14:  for i = q; i ≤ m; i++ do
15:    η̄_i = η̄_m;
16:  end for
17:  q = m + 1;
18: end while
```

The left-hand side of the Inequality represents (a). the computational load of those higher priority tasks whose new slowdown factors have been determined; and (b). worst-case blocking of τ_i and the computational load of those higher priority tasks who do not have new slowdown factors yet, with the processor slowed down to $\tilde{\eta}_i$. By comparing (3) and Inequality (5), Inequality (5) certainly holds for $S_{i,l}$. However, if the Inequality also holds for another scheduling point $S_{i,k} \ne S_{i,l}$, then we know that with the processor slowed down to $\tilde{\eta}_i$, τ_i will begin execution at some time before $S_{i,k}$, even in the worst case. When τ_i begins execution, no other tasks can preempt it. This implies that the workload from tasks with priorities higher than τ_i should be calculated with respect to $S_{i,k}$, instead of $S_{i,l}$ as in USFI.

Let $\overline{S}_i = \{S_{i,k} | S_{i,k} \in S_i$, and Inequality (5) holds for $S_{i,k}\}$. We know $\overline{S}_i \ne \emptyset$, since there is at least one scheduling point in it, i.e., $S_{i,l}$. Now for each $S_{i,k} \in \overline{S}_i$, we calculate a new candidate slowdown factor $\overline{\eta}_{i,k}$ using the following equation:

$$\sum_{1\le r<q} \frac{C_r}{\overline{\eta}_r}\left\lceil\frac{S_{i,k}}{T_r}\right\rceil+\frac{1}{\overline{\eta}_{i,k}}\left(B_i+\sum_{q\le p\le i} C_p\left\lceil\frac{S_{i,k}}{T_p}\right\rceil\right)=D_i. \tag{6}$$

At first glance, one may wonder why D_i (deadline of τ_i), rather than $S_{i,k}$, is used on the right-hand side of (6)? The following lemma justifies why it is beneficial to do so.

Lemma 1: Using D_i on the right-hand side of (6) is a better strategy than using $S_{i,k}$ in terms of minimizing the slowdown factor of τ_i.

Proof: Suppose $S_{i,k}$ is used on the right-hand side of (6) to calculate $\overline{\eta}_{i,k}$. Obviously, this still meets the deadline of task τ_i. In addition, Inequality (5) will hold if we replace $\tilde{\eta}_i$ by $\overline{\eta}_{i,k}$ and $S_{i,j}$ by $S_{i,k}$ in (5). Hence, the new slowdown for τ_i can be calculated simply as $Min_{S_{i,k}\in\overline{S}_i}(\overline{\eta}_{i,k})$. Since $\overline{S}_i \subseteq S_i$, this

new slowdown factor for τ_i is certainly no smaller than $\tilde{\eta}_i$. On the other hand, by comparing (3) and (6), it is obvious that the $\overline{\eta}_i$ computed by using our approach is not larger than $\tilde{\eta}_i$. Combining these two facts, the lemma follows. ■

Note that using $\overline{\eta}_{i,k}$ in (6) as τ_i's slowdown factor can guarantee τ_i's deadline, but Inequality (5) may no longer hold if we replace $\tilde{\eta}_i$ by $\overline{\eta}_{i,k}$ and $S_{i,j}$ by $S_{i,k}$ in (5), which means τ_i may not begin execution some time before $S_{i,k}$. This is impermissible since in nonpreemptive scheduling, once the task instance has seized the control of processor, it will not release it until it finishes its execution. Thus, if τ_i does not start its execution before $S_{i,k}$, the slowdown factor calculated by (6), which uses $S_{i,k}$ to bound the span of higher priority tasks, cannot guarantee that τ_i will meet its deadline. To address this problem, consider the following ratio:

$$\frac{B_i + \sum_{q \le p < i} C_p \left\lceil \frac{S_{i,k}}{T_p} \right\rceil}{S_{i,k} - \sum_{1 \le r < q} \frac{C_r}{\overline{\eta}_r} \left\lceil \frac{S_{i,k}}{T_r} \right\rceil}. \tag{7}$$

Based on our definitions of $\overline{S}_i$ and Inequality (5), $\tilde{\eta}_i$ is larger than the value in (7). We choose a value that is a little larger than (7), but is smaller than $\tilde{\eta}_i$. We call this value $\overline{\overline{\eta}}_{i,k}$. Clearly, Inequality (5) still holds when we replace $\tilde{\eta}_i$ by $\overline{\overline{\eta}}_{i,k}$ in (5). We let $\eta'_{i,k} = Max(\overline{\eta}_{i,k}, \overline{\overline{\eta}}_{i,k})$. Now we are sure that using $\eta'_{i,k}$ as τ_i's slowdown factor cannot only meet τ_i's deadline, but also ensure τ_i's kick-off before $S_{i,k}$. In order to achieve better energy savings we let τ_i's new candidate slowdown factor $\overline{\eta}_i = Min_{S_{i,k} \in \overline{S}_i}(\eta'_{i,k})$. Finally, Lines 13–15 of Algorithm 1 further choose the maximum candidate slowdown factor from among $\overline{\eta}_q, \ldots, \overline{\eta}_n$ as the final slowdown factor for tasks $\tau_q, \ldots, \tau_m$.

In the following, we present an example to illustrate our approach.

Example 2: We compute new slowdown factors for the task set given in Example 1. The same as `USFI`, `ISA` involves three iterations of computation, and the calculation details are shown in Table I. Since the slowdown factor for τ_1 computed by `ISA` is the same as that of `USFI`, i.e., $\overline{\eta}_1 = 0.6$, we omit to show the detail of computing $\overline{\eta}_1$. Now, we come to the second iteration, with $\overline{\eta}_1 = 0.6$, we continue to find $\overline{\eta}_2$ and $\overline{\eta}_3$. By using (3) and (4) with $q = 2$, we obtain $\tilde{\eta}_2 = 0.45$ and $\tilde{\eta}_3 = 0.375$. For task τ_2, Inequality (5) holds at $S_{2,1} = 5$ and $S_{2,2} = 10$, hence $\overline{S}_2 = \{S_{2,1}(5), S_{2,2}(10)\}$. We go on to find $\overline{\eta}_{2,1} = 0.36$, $\overline{\overline{\eta}}_{2,1} = 0.3001$, $\overline{\eta}_{2,2} = 0.45$, and $\overline{\overline{\eta}}_{2,2} = 0.15001$. Hence, $\overline{\eta}_2 = Min(\eta'_{2,1}, \eta'_{2,2}) = 0.36$. Following the same steps, we get $\overline{\eta}_3 = 0.3001$. Since $\overline{\eta}_2 > \overline{\eta}_3$, the second iteration is concluded with $\overline{\eta}_2 = 0.36$. Finally, our third iteration determines $\overline{\eta}_3$ to be 0.09. In summary, the new slowdown factors computed by our method are $\overline{\eta}_1 = 0.6$, $\overline{\eta}_2 = 0.36$, and $\overline{\eta}_3 = 0.09$, respectively.

Now, we need to be sure that all task deadlines can still be met with our new slowdown factors. We first introduce two important facts. From the computation process of `ISA`, it is not difficult to derive the following fact.

Lemma 2: Given the tasks are ordered in a nondecreasing order of their periods, the new slowdown factors computed by `ISA` are in a nonincreasing order.

Proof: This property follows from the sequence in which the `ISA` algorithm assigns slowdown factors to tasks. Note that `ISA` first computes one candidate slowdown factor for each unassigned task by (6) and (7), in which the same slowdown factor is assumed for all unassigned tasks (lines 7–11 of Algorithm 1). Then, in each iteration of the algorithm, the maximum over the task candidate slowdown factors, $\bar{\eta}_m$ (line 13 of Algorithm 1), is assigned to the unassigned tasks up to index m (lines 14–16). Since the maximum over all candidate slowdown factors is assigned to some tasks, the candidate slowdown factor in future iterations can only be lower. Therefore, the candidate slowdown factor of each task and the maximum over the candidate slowdown factors is nonincreasing in sequential iterations, which means the final task slowdown factors computed by `ISA` are also in nonincreasing order. ■

TABLE I
CALCULATION OF OUR NEW SLOWDOWN FACTORS FOR THE TASK SET GIVEN IN EXAMPLE 1 IN THREE ITERATIONS

Iter.	τ_1	τ_2	τ_3
1st	$\tilde{\eta}_1 = 0.6$ $\overline{S}_1 = \{S_{1,1}\}$ $\overline{\eta}_1 = 0.6$	$\tilde{\eta}_2 = 0.5$	$\tilde{\eta}_3 = 0.45$
	$\overline{\eta}_1 = 0.6$		
2nd		$\tilde{\eta}_2 = 0.45$ $\overline{S}_2 = \{S_{2,1}, S_{2,2}\}$ $\overline{\eta}_2 = 0.36$	$\tilde{\eta}_3 = 0.375$ $\overline{S}_3 = \{S_{3,2}, S_{3,4}\}$ $\overline{\eta}_3 = 0.3001$
	$\overline{\eta}_1 = 0.6$	$\overline{\eta}_2 = 0.36$	
3rd			$\tilde{\eta}_3 = 0.45$ $\overline{S}_3 = \{S_{3,2}, S_{3,4}\}$ $\overline{\eta}_3 = 0.09$
	$\overline{\eta}_1 = 0.6$	$\overline{\eta}_2 = 0.36$	$\overline{\eta}_3 = 0.09$

With our new slowdown factors computed, the run-time scheduler will use them to control the processor speed at different points in time so as to reduce energy consumption, while still guaranteeing all task deadlines. In this paper, we consider two policies to control the processor speed: 1) `FI` [13]: When blocking occurs, if the blocked job has a higher slowdown factor, this slowdown factor is inherited by the blocking job. In all other cases, each job will execute at its corresponding slowdown factor to save energy; 2) `SFI`. We consider `FI` in this section and will discuss `SFI` in Section IV-C. By Lemma 2 and the `FI` policy, we have the following fact.

Lemma 3: During the time period when τ_i is executing, the minimum slowdown of the processor is $\overline{\eta}_i$.

With the `FI` policy, we prove that the slowdown factors computed by `ISA` can guarantee all task deadlines, as stated below.

Theorem 1: Based on the DM scheduling policy, the slowdown factors computed by `ISA` along with the `FI` policy of controlling the processor speed guarantee all task deadlines.

Proof: Our approach assigns every task a slowdown factor that is larger than or equal to the slowdown values given in (6) and (7). Based on this, we know that for every task τ_i, $i = 1, \ldots, n$, there exists a scheduling point $S_{i,k}$ such that the following two inequalities hold:

$$\frac{1}{\overline{\eta}_i} B_i + \sum_{1 \le j < i} \frac{C_j}{\overline{\eta}_j} \left\lceil \frac{S_{ik}}{T_j} \right\rceil + \frac{1}{\overline{\eta}_i} C_i \le D_i \tag{8}$$

$$\frac{1}{\overline{\eta}_i} B_i + \sum_{1 \le j < i} \frac{C_j}{\overline{\eta}_j} \left\lceil \frac{S_{ik}}{T_j} \right\rceil < S_{i,k}. \tag{9}$$

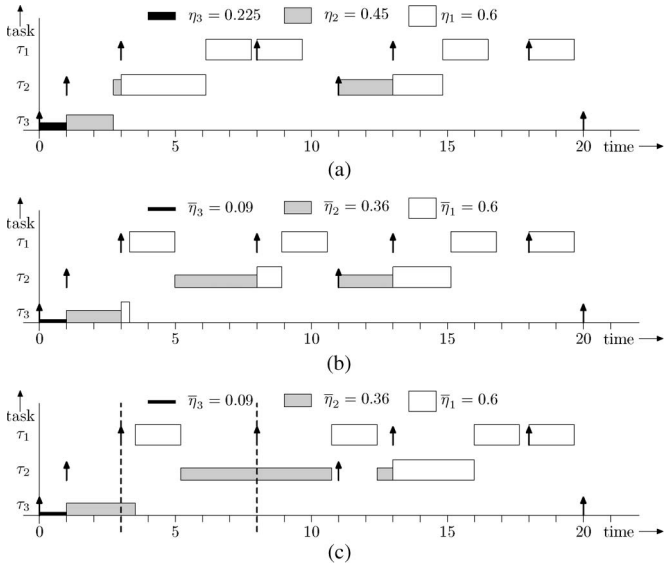

Fig. 2. (a) Task schedule by USFI for the three tasks given in Example 1. Assuming τ_1, τ_2 and τ_3 arrive at time 0, 1, and 3, respectively. (b) Task schedule by ISA with FI. ISA-FI consumes 9% less frequency-dependent energy than USFI. (c) Task schedule by ISA with SFI. Compared to USFI, 14% dynamic energy savings can be achieved by ISA-SFI. (a) Task schedule produced by USFI. (b) Task schedule produced by ISA-FI. (c) Task schedule produced by ISA-SFI.

We prove the above claim by contradiction. Suppose an instance of task τ_i misses its deadline. Let t be the time when the instance was released, and $\mathbb{A}$ be the set of jobs that arrive during $[t, t + D_i]$ with priority higher than that of τ_i, i.e., $\mathbb{A} \subseteq \{\tau_1, \ldots, \tau_{i-1}\}$. The workload from jobs in $\mathbb{A}$ during the interval is clearly bounded by $\sum_{j=1}^{i-1}(C_j/\overline{\eta}_j)\lceil D_i/T_j \rceil$. According to (9), there exists a scheduling point $S_{i,k}$ during the lifetime of τ_i before which τ_i can starts its execution. Therefore, the workload from jobs in $\mathbb{A}$ during $[t, t + D_i]$ should be characterized as $\sum_{j=1}^{i-1}(C_j/\overline{\eta}_j)\lceil S_{i,k}/T_j \rceil$ instead. The maximum execution time of τ_i at full speed is bounded by B_i. By Lemma 3, τ_i's maximum blocking is $(1/\overline{\eta}_i)B_i$. Thus, the total execution time of the jobs during $[t, t + D_i]$ is bounded by $(1/\overline{\eta}_i)B_i + \sum_{j=1}^{i-1}(C_j/\overline{\eta}_j)\lceil S_{i,k}/T_j \rceil + (1/\overline{\eta}_i)C_i$, which is larger than D_i by our assumption. However, this contradicts (8). Hence, all tasks are guaranteed to meet their deadlines. ■

Fig. 2(a) shows the schedule of the first 20 time units produced by USFI for the task set given in Example 1. We assume τ_1, τ_2 and τ_3 arrive at time 0, 1, and 3, respectively. As seen in Fig. 2(a), blocking occurs at time 1, 3, and 13, hence the processor speed is changed accordingly. With our new slowdown factors computed in ISA and the FI policy, the processor speed is changed at time 1, 3, 8, and 13, as shown in Fig. 2(b). As dynamic energy consumption (P_{dep}) is proportional to the cubic of slowdown factors, the energy consumed by USFI and ISA-FI in this example is 2.88 and 2.62, respectively, which means 9% dynamic energy savings can be obtained by using our slowdown factors, in the first 20 time units. Notice here we do not include frequency-independent power P_{ind} for comparison since by [37], there exists a minimum threshold slowdown factor, below which lower frequency will increase the energy consumption. We will address this issue later in our experiment study in Section VI.

Complexity: The time complexity of ISA is dominated by the computation of the candidate slowdown factors. Since the number of scheduling points in S_i is pseudopolynomial in the exact deadline monotonic analysis, ISA is also in the pseudopolynomial time complexity. Nevertheless, in practice, the number of scheduling points is not large, and the algorithm is very efficient. Moreover, one can utilize the approach presented by Fisher *et al.* in [8] to derive a polynomial time algorithm by taking some approximation for the schedulability tests.

C. SFI: Selective Frequency Inheritance

From our discussions in Section IV-B, one can see that our computation of the candidate slowdown factor for a task τ_i, as given by (6), assumes the same slowdown for τ_i and its blocking section, with $B_i/\overline{\eta}_i$ as the maximum blocking duration for τ_i when our approach determines $\overline{\eta}_i$. This assumption is reflected in the FI policy: When blocking occurs, if the blocked task has a higher slowdown factor, this slowdown factor is inherited by the blocking task. However, is the FI policy of changing the processor speed whenever blocking occurs a good strategy in minimizing energy consumption, while guaranteeing task deadlines? We will argue below that the answer is negative by presenting our solution.

Suppose one can estimate the maximum remaining computation time needed for each active task. Assuming an instance of τ_i arrives while a lower priority task τ_j is executing, and τ_j requires at most R_j of computation time to finish at the current processor speed. We propose the *SFI* policy as follows: If R_j is not larger than $B_i/\overline{\eta}_i$, we continue to execute τ_j at the current speed. Otherwise, the processor speed is promoted to $\overline{\eta}_i$. The SFI policy also employs the slowdown factors computed by ISA. We will show in Theorem 2 below that our ISA algorithm along with the SFI policy still guarantees all task deadlines. The correctness of the theorem follows from Theorem 1 and the following lemma.

Lemma 4: When τ_a arrives while a lower priority task τ_b is executing, if the SFI policy leaves the processor speed unchanged because $R_b \leq B_a/\overline{\eta}_a$, with R_b being the remaining execution time of τ_b at the current processor speed, then none of the tasks already blocked by τ_b will be affected in terms of meeting its deadline.

Proof: Consider any task τ_i that is already blocked by τ_b when τ_a arrives. As shown in Fig. 3, let t_1 and t_2 be the time instants when τ_i and τ_a become blocked by τ_b, respectively, and t_3 denotes the time when τ_b finishes. Consider the scheduling point $S_{i,k}$ of τ_i from which we determine $\overline{\eta}_i$. Since τ_a started to be blocked from time t_2, during the interval $[t_1, t_1 + S_{i,k}]$, τ_i will await at most the following amount of time before it becomes eligible to run: (1) τ_i has a higher priority than τ_a: $t_3 - t_1 + \sum_{j=1}^{i-1}(C_j/\overline{\eta}_j)\lceil S_{i,k}/T_j \rceil$; (2) τ_a has a higher priority than τ_i: $t_3 - t_1 + \sum_{j=1, j\neq a}^{i-1}(C_j/\overline{\eta}_j)\lceil S_{i,k}/T_j \rceil + (C_a/\overline{\eta}_a)\lceil (S_{i,k} + t_1 - t_2)/T_a \rceil$. When looking at the terms in these two expressions, one can see that despite the decision to leave the processor speed unchanged at time t_2, only the term $t_3 - t_1$ is affected. To address the effect of applying the SFI policy

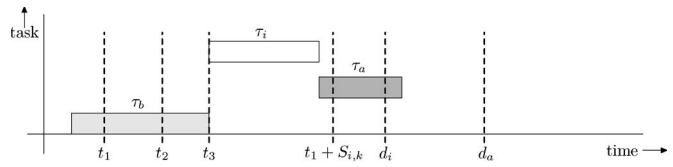

Fig. 3. Illustration example for Lemma 4.

at time t_2, we need to consider the following two situations at time t_1:

- τ_b inherits $\overline{\eta}_i$ at time t_1: During $[t_1, t_2]$, the processor speed is certainly no less than $\overline{\eta}_i$. No matter whether the SFI policy promotes the processor speed at time t_2 or not, the processor speed is still no less than $\overline{\eta}_i$ during $[t_2, t_3]$. Hence, $t_3 - t_1$ is upper bounded by $B_i/\overline{\eta}_i$.
- FI did not occur at time t_1: This means that at time t_1, $R_b \leq B_i/\overline{\eta}_i$. During $[t_1, t_2]$, the processor speed is certainly no less than the speed at time t_1. Again, no matter whether the SFI policy promotes the processor speed at time t_2 or not, the processor speed during $[t_2, t_3]$ is no less than the speed at time t_1. Hence, we know $t_3 - t_1 \leq B_i/\overline{\eta}_i$.

Based on the above facts and our arguments given in the proof of Theorem 1, τ_i can still finish its execution before $t_1 + S_{i,k}$, and hence its deadline. ■

As for τ_a, i.e., the newly blocked task when the SFI policy is applied, it is not hard to see that its maximum blocking time is bounded by $B_a/\overline{\eta}_a$. The workload on τ_a from higher priority tasks remains the same. By this fact, Lemma 4, and Theorem 1, we thus have the following theorem.

Theorem 2: Based on the DM scheduling policy, the slowdown factors computed by ISA along with the SFI policy can guarantee all task deadlines.

Fig. 2(c) shows the schedule produced by the SFI policy for the task set given in Example 1. At time 3, $\tau_{1,1}$ arrives while $\tau_{3,1}$ is running at speed $\overline{\eta}_2 = 0.36$. At this time instant, $\tau_{3,1}$ needs at most 0.53 time units to complete at its current speed, whereas $\tau_{1,1}$ can tolerate a maximum blocking of $B_1/\overline{\eta}_1 = 2/0.6$. Based on Theorem 2, $\tau_{3,1}$ can keep running at speed 0.36 without jeopardizing the deadline of $\tau_{1,1}$. Likewise, $\tau_{2,1}$ can continue execution at speed $\overline{\eta}_2$ at time 8. In this case, the energy consumed by our SFI policy is 2.48 in the first 20 time units, which is 13.8% improvement over USFI. Our experimental results in Section VI show that the SFI policy gives us a large gain in energy savings under a variety of parameter settings. Note that the overhead involved with SFI is small—the scheduler only needs to record the processor time spent on executing each active job, and whenever blocking occurs it performs a simple computation and test, as described in Lemma 4, to determine if the processor speed must be increased.

D. Remarks for Practical Systems and DS

1) Processors With Discrete Speeds: For commercial processors, only discrete speeds might be available. To address this problem, we can first derive a solution by assuming continuously available speeds as the approaches in Section IV-B, and then use some available speeds to approximate the speeds determined by ISA. For example, we first compute a speed for τ_1 by ISA. If this speed is unavailable, we then choose the next higher speed (also called *inflated* speed) which is available and assign it to τ_1. After that, we utilize the *inflated* speed to compute slowdown factors for the remaining tasks $\tau_2, \ldots, \tau_n$, and choose the next higher available speeds for them, in a similar way. Note here our method is similar to the one adopted by the PM-Clock algorithm proposed in [25], which focuses on fixed priority *preemptive* scheduling.

2) Non-Negligible Frequency Transition Overhead: It is now a common knowledge that changing from one frequency level to another takes a fixed amount of time (denoted Δt, ranges from tens of microseconds to tens of milliseconds), referred to as the transition (or switch) overhead [17]. When frequency transition overhead is large enough that they cannot be ignored, we need to recompute the slowdown factors by considering such overhead. From the computation process of ISA, it can be observed that for task τ_i, the number of frequency switches, say λ, is bounded by the number of jobs with higher priorities than τ_i, plus two additional possible switches, one for τ_i itself (when the system state is changed from idle to executing τ_i, or conversely), and the other for a lower priority blocking job. Specifically, we have $\lambda = \sum_{1 \leq r < q} \lceil S_{i,j}/T_r \rceil + 2$ for formulas (3) and (5), and $\lambda = \sum_{1 \leq r < q} \lceil S_{i,k}/T_r \rceil + 2$ for (6). Hence, if the transition workload is non-negligible, we should add $\lambda \Delta t$ into the left-half-part of (3), (5) and (6) to compute $\overline{\eta}_i$. Note here our method for counting transition workload is a conservative one since it considers the worst-case scenario, but it is also a safe one as it suffices to guarantee schedulability.

In addition to time overhead, transition also incurs energy overhead [17], hence, it is important to guarantee that the energy saved by slowdown policy must be no less than the increased energy consumption resulted from the transition overhead. However, since the actual number of frequency transition may change during run-time and depends on the actual execution cycles of each job, it is hard to characterize a condition under which the energy saved by slowdown is less than the energy consumption by frequency transition. Because the focus of this paper is handling time overhead while meeting deadlines, we leave the problem of tackling the energy overhead of transition for future work.

3) Refinement on DS: Even though we only present algorithm ISA for deciding one individual speed for each task, it is also applicable to refine algorithm DS[34]. A simple yet efficient refinement would be choosing the highest speed by ISA as the high speed in DS. This certainly will guarantee the schedulability, while energy reduction can also be achieved comparing to DS. Since the refinement is trivial and straightforward, we do not detail it in this work.

V. Dynamic Slack Reclamation

The ISA algorithm computes one speed for each task to decrease the energy consumption under nonpreemptive scheduling. However, in practice, many task instances may finish

before their estimated completion times obtained under the assumption of worst-case workload with the computed static speeds. In this case, dynamic slack arises due to early task completions. It is important to note that even tasks spend less than their WCETs to complete, the feasibility of the task set with the computed slowdown factors can still be guaranteed, due to the FI (or SFI if possible) policy that is applied when blocking occurs. Nevertheless, these slacks can be reclaimed to further reduce the processor speed of later tasks as long as the feasibility is guaranteed, which in turn can result in additional energy savings. In fact, reclaiming unused computation time to reduce the CPU speed while preserving feasibility has been widely studied in numerous research papers in recent years [4], [11], [20], though most of them focused on preemptive scheduling.

Algorithm 2 The ISA-DR algorithm

1: **When a new job**(J_i) **arrives**:
2: $C_i^r(t) = C_i$;
3: $R_i^r(t) = C_i/\eta_i$;
4: **if** P_i(Priority of J_i) $> P_c$(Priority ofJ_c) **then**
5: **if** $(R_c^r(t) \leq B_i/\overline{\eta}_c)$ **then**
6: continue execution with η_c; //SFI is employed.
7: **else**
8: setSpeed $(\overline{\eta}_i)$; //FI is employed.
9: **end if**
10: **end if**
11: **When a job** J_i **is selected to run**:
12: setSpeed $(C_i^r(t)/(R_i^F(t) + R_i^r(t)))$;
13: Execute J_i;
14: **When job** J_i **completes**:
15: **if** $R_i^r(t) > 0$ **then**
16: Insert F-List$(R_i^r(t), P_i)$;
17: **end if**

In this section, we present a reservation-based scheme which dynamically collects the residue time from early task completions. Since the algorithm is an extension to the ISA algorithm, we call it ISA with dynamic reclamation (ISA-DR). Our approach is based on the idea that when no higher priority tasks are blocked, tasks can reclaim the unconsumed run-time from higher priority (than the current running task's priority) tasks while meeting all task deadlines. First, we need to define the *run time* of a job (or task instance), which can be viewed as a budget assigned to the job. Specifically, the run time of a job with a workload C (at maximum processor speed) and slowdown factor η, is defined as C/η. Each run time has a time budget and a priority associate with it. The system maintains a Free-run-time List named F-List to collect the run time not consumed. Each item in F-List contains two values: the run-time budget and its priority, which is equal to the priority of the completed task (τ_i) instance and defined as P_i. The F-List is sorted in decreasing order of item priorities. In this way, ISA-DR effectively reclaims unused run time for redistribution, which in turn reduces the processor idle time and leads to more energy savings.

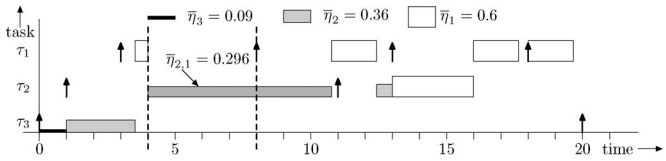

Fig. 4. Dynamic slack reclamation.

When a task instance is scheduled to run, it is eligible to use its own run time, as well as the run time in the F-List with priority larger than its priority. With the additional free run time, the task instance can be processed at a lower speed. If the run time in an item of the F-List is depleted, this item is removed. Before we formally present the ISA-DR algorithm, similar to [11] and [34], we first introduce the following notations that will be used in the algorithm:

- J_c: the current job of task τ_c executing in the system.
- J_i: the current job of task τ_i.
- $\overline{\eta}_i$: the slowdown factor computed by ISA for τ_i.
- $R_i^r(t)$: the available run time of job J_i at time t.
- $R_i^F(t)$: the run time in the F-List that can be used by J_i at time t.
- $C_i^r(t)$: the worst-case residue execution time of job J_i under the maximum speed at time t.

The skeleton of ISA-DR is given in Algorithm 2. The execution speed is computed according to the usable run time $R_i^F(t) + R_i^r(t)$ and the WCET C_i (line 12), except when the current job is blocking another job, where the FI (or SFI) policy is employed (line 4–10). On job completion, the unconsumed run time is added to the F-List with the same priority of the job J_i (line 16). To complete the ISA-DR algorithm, the following rules are applied to update the run time and the worst-case remaining execution time of a job, as well as the run time in the F-List.

1) As job J_i executes, it consumes run time at the same speed as wall clock. If $R_i^F(t) > 0$, the run time is used from the F-List; Otherwise, $R_i^r(t)$ is used. $C_i^r(t)$ is reduced by the processor execution speed per time unit.
2) When the processor is idle, if $R_i^F(t) > 0$, the run time in F-List is consumed at the same speed as wall clock.

Note that the rules only need to be applied on the arrival and completion of a task instance in the system. Since the FI (or SFI if possible) policy is always adopted when task blocking occurs, it is relatively easy to see that ISA-DR can guarantee all task deadlines.

We now give an example to illustrate how ISA-DR works. Consider again the task set given in Example 1, assume that $\tau_{1,1}$ completes earlier at time $t = 4$, leaving an unused run time of 1.2 time units, as shown in Fig. 4. This run time has higher priority than $\tau_{2,1}$ and hence can be reclaimed to further reduce the speed of $\tau_{2,1}$. When $\tau_{2,1}$ starts to execute, the available run time is 2/0.36, and the free run-time in the F-List that can be used by $\tau_{2,1}$ is 1.2. Consequently, we have $\overline{\eta}_{2,1} = (2/(2/0.36 + 1.2) = 0.296$, and $\tau_{2,1}$ will start its execution with this speed. At $t = 8$, $\tau_{1,2}$ arrives and is blocked by $\tau_{2,1}$. Due to the SFI policy, $\tau_{2,1}$ can continue its execution with speed 0.296.

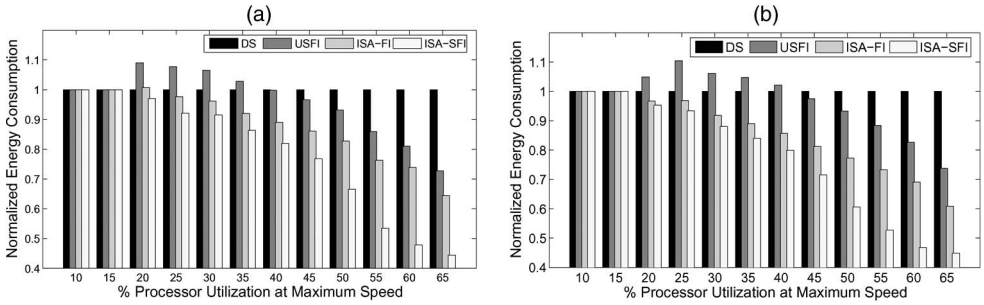

Fig. 5. Normalized energy consumption with varying processor utilization, $\eta_{ee} = 0.29 (P_{ind} = 0.05)$. (a) $\mathsf{SFr} = 0.05$. (b) $\mathsf{SFr} = 0.15$.

VI. Performance Evaluation

To evaluate the effectiveness of our techniques, we have conducted extensive experiments based on randomly generated test data. We first describe the system model in Section VI-A and z present the experimental results in Section VI-B.

A. Experimental Setup

We have developed an event-driven simulator written in C++, which imitates the processing of periodic tasks. As described in detail in Section III, the tasks run on a uniprocessor system that is capable of DVS.

We perform our simulation based on the power model by Zhu *et al.* [37], in which the power consumption consists of the frequency-dependent active power (P_{dep}) and frequency-independent active power (P_{ind}). That is, $P_d = P_{ind} + P_{dep}$. Since P_I is a constant, we will not consider this part in our experiments. By [37], there exists a minimum threshold slowdown $\eta_{ee} = \sqrt[m]{P_{ind}/C_{ef}(m-1)}$ (expressed as critical frequency) below which lower frequency will increase the energy consumption for execution. The effective switching capacitance C_{ef} and the dynamic power exponent m are system dependent constants. As in [37], we assume $C_{ef} = 1$ and $m = 3$, and we consider P_{ind} to be either 0.05 or 0.1, which means η_{ee} is either 0.29 or 0.37. In our simulation experiments below, the normalized maximum and minimum processor speed are assumed to be 1 and η_{ee}, respectively. Speed levels between these two bounds are discrete and spaced by 0.05.

To facilitate comparison, we have adopted the parameters given in [13], [34] to generate task sets. In particular, each task period belongs to one of the following three ranges: long period (2000 $\sim$ 5000), medium period (500 $\sim$ 2000), and short period (90 $\sim$ 250). The WCET for the three period ranges is (10 $\sim$ 500), (10 $\sim$ 100), and (10 $\sim$ 20), respectively. The tasks in each task set generated are uniformly distributed in the three period/WCET categories, while the ratio between task deadline and period is uniformly distributed in [0.8, 1]. Within each category, a taskaŕs period and WCET are randomly selected from the corresponding ranges. For each task set to qualify for our experiments, the static slowdown factors computed by every algorithm must be no larger than 1.

We have conducted three sets of simulations based on different system parameters, i.e., processor utilization, task granularity and slack factor. For each particular level in each simulation set, we randomly generate 1000 qualified task sets and take the average, each task set consists of 5 $\sim$ 15 tasks. All tasks in a task set are released at time 0. We then perform experiments for every evaluated approach for 100 000 time units and record the energy consumed by the approach (including both P_{dep} and P_{ind}). For each point plotted in the figure, the simulations continued until a confidence interval of 95% with half-width of less than 5% about the mean was achieved.

B. Experiment Results

1) Processor Utilization: In the first set of simulations, we vary the processor utilization U from 10% to 65% to see how different approaches fare in the energy consumption metric. The tasks in each task set generated are uniformly distributed in the three period/WCET categories, while the ratio between task deadline and period is uniformly distributed in [0.8, 1]. Within each category, a task's period and WCET are randomly selected from the corresponding ranges. All tasks are assumed to execute at their WCETs. We compare the performance of the following algorithms: `DS`, `USFI`[13] algorithms adapted to nonpreemptive task sets by treating each individual task as one single critical section, `ISA` with FI (`ISA-FI`) and `ISA` with SFI (`ISA-SFI`). To facilitate comparison, the energy consumed by `DS` is used as the baseline, and the energy consumed by each other solution is normalized against the baseline. We define the improved slowdown factor ratio, denoted SF_r, as $\eta_{USFI}/\eta_{ISA} - 1$, where η_{USFI} and η_{ISA} represent the maximum slowdown factor computed by `USFI` and `ISA`, respectively. For most of the task sets generated, SF_r lies in the range [0.0, 0.2]. We thus classify the task sets into two groups, with one having $SF_r = 0.05 \pm 0.02$ as shown in Figs. 5(a) and 6(a), and the other having $SF_r = 0.15 \pm 0.02$ as shown in Figs. 5(b) and 6(b). We present observations from our experiments below:

1) When the system utilization is below 15%, the slowdown factors computed by every algorithm we investigate are all less than the threshold frequency η_{ee}, hence they are set to be the threshold frequency, as shown in Figs. 5 and 6.
2) As plotted in Figs. 5 and 6, `ISA-FI` consistently outperforms `USFI` in that it computes a smaller slowdown factor for each task. Recall that the slowdown factors computed by both `USFI` and `ISA-FI` are in a nonincreasing order, since the highest priority task in a task set in these two approaches has the largest slowdown factor and executes and invokes speed promotion most

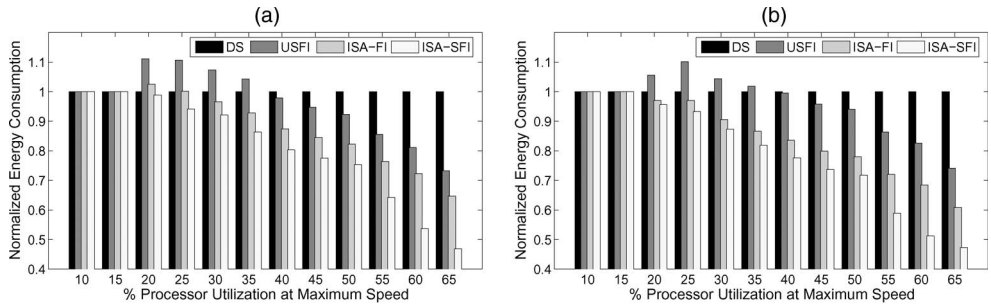

Fig. 6. Normalized energy consumption with varying processor utilization, $\eta_{ee} = 0.37$ ($P_{ind} = 0.1$). (a) $\mathsf{SFr} = 0.05$. (b) $\mathsf{SFr} = 0.15$.

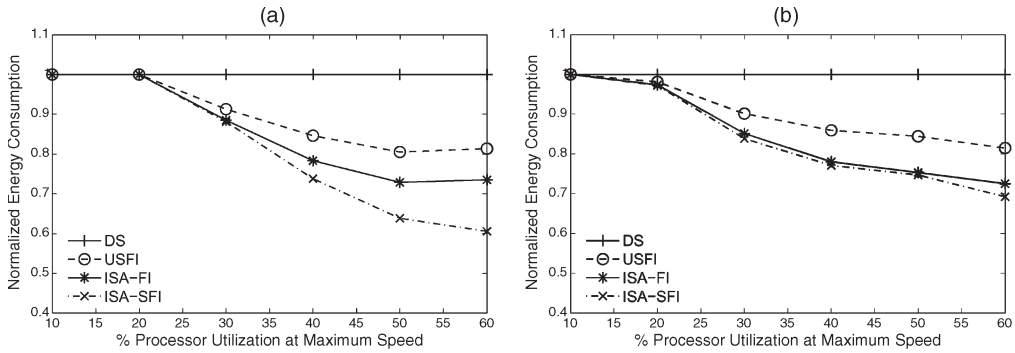

Fig. 7. Normalized energy consumption with different task granularity, $\eta_{ee} = 0.29$ ($P_{ind} = 0.05$). (a). Looser task set. (b). Stringent task set.

frequently, it is natural to conjecture that energy savings would be more significant as the difference between the maximum slowdown factors computed by USFI and ISA-FI gets larger. This trend is indeed reflected in Figs. 5 and 6. Both figures show that the performance gap between USFI and ISA-FI is wider at a larger value of SF_r. On average, USFI consumes 8% to 15% more energy than ISA-FI. The energy reduction achieved by ISA-FI, relative to USFI, mainly comes from its lower slowdown factors, since both ISA-FI and USFI use the same speed-control policy.

3) ISA-SFI performs better than all other approaches in nearly all cases. Like ISA-FI, as the processor utilization becomes larger, the energy reduction achieved by ISA-SFI becomes more. When the processor utilization reaches 60%, ISA-SFI consumes, on average, 32% and 27% less energy than USFI when $\eta_{ee} = 0.29$ and 0.37, respectively. This is because at high processor utilization, tasks tend to experience more blocking, and the benefit of SFI becomes obvious. Unlike ISA-FI, the change of SF_r does not affect the magnitude of energy reduction by ISA-SFI, as shown in both Figs. 5 and 6. This phenomenon also explains ISA-SFI's primary energy savings come from the SFI policy, rather than its smaller slowdown factors.
4) The effect of increasing the proportion of frequency-independent power in our energy consumption formula is clearly seen from Figs. 5 and 6. When the frequency-independent power increases in the formula, our approach becomes less energy efficient. However, we can still achieve 5% to 10% more energy savings, compared to USFI, when η_{ee} changes from 0.29 to 0.37.

2) Task Granularity: In the second set of experiment, we generate two types of task set, namely stringent task set and looser task set, to examine how different approaches fare in the energy consumption under different task granularity. In the stringent task set, one task is generated from the long period, and one is selected from the medium period, while the others are randomly generated from the short period. Conversely, in the looser task set, one task is selected from the short period, and one is from the medium period, while the others are randomly from the long period. In this simulation, all tasks are also assumed to execute up to their WCETs and η_{ee} is fixed to be 0.29. The system utilization is varied from 10% to 60%. It can be observed from Fig. 7(a) that for looser task set, ISA-SFI consistently outperforms the other three schemes on energy consumption performance, except when the utilization is below 15%, all the four schemes have the same energy consumption, due to that the slowdown factors computed by every algorithm are all less than the threshold frequency η_{ee} and they are set to be η_{ee}. Moreover, it can be seen that the gap between ISA-SFI and the other three approaches is increasing with the growth of the utilization. This is because at high utilization, tasks tend to experience more blocking, and the benefit of SFI becomes obvious. For the stringent task set, ISA-SFI also consistently outperforms the other three schemes, as shown in Fig. 7(b). However, the difference between ISA-SFI and ISA-FI is not that large as in the coarse-gained task set. The reason is that for stringent task set, the slowdown factors computed are more close to each other than that for the looser task set. In fact, in our experiment we observe that most of the task slowdown factors computed are the same for the stringent task set, hence the energy savings from the SFI policy is not that significant as for the looser task set.

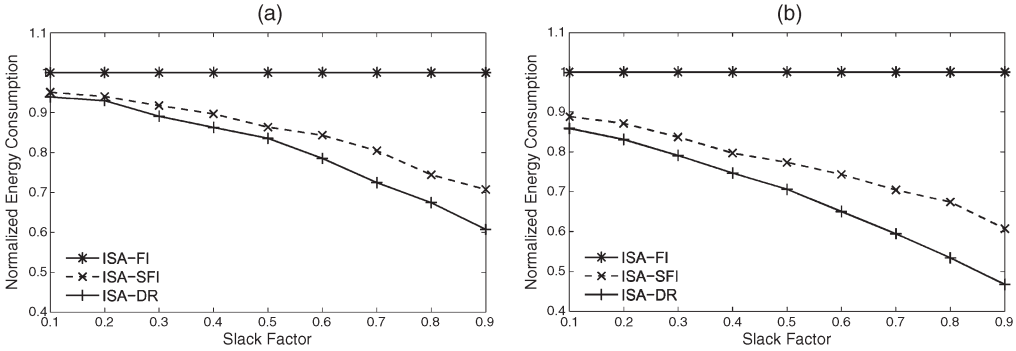

Fig. 8. Normalized energy consumption with varying task execution time, $\eta_{ee} = 0.29$ ($P_{ind} = 0.05$). (a). Utilization $= 0.3$. (b). Utilization $= 0.5$.

3) Slack Factor: In this set of simulations, we examine the additional energy gains derived through the dynamic slack reclamation policy. For simplicity, we only compare the performance of the following algorithms: ISA-FI, ISA-SFI, and ISA-DR. We define a notation namely slack factor (sf) to specify the difference between the actual case execution time (ACET) and the task's WCET. Specifically, the ACET of the tasks are uniformly distributed between $(1 - sf) \times$ WCET and the task's WCET. η_{ee} is fixed to be 0.29. To facilitate comparison, the energy consumed by ISA-FI is used as the baseline, and the energy consumed by each other solution is normalized against the baseline. Fig. 8 shows the simulation results for slack factor varies from 0.1 to 0.9 in steps of 0.1. It can be seen that ISA-DR can achieve considerable energy savings (up to 20%–30% on average) compared to ISA-FI, due to its ability to reclaim unused run time for early completions. Also, ISA-SFI gains considerable energy savings when the tasks exhibit varying execution times, due to the SFI policy which enables to execute at a low speed when blocking occurs. Comparing the energy savings at $U = 0.3$ and $U = 0.5$, we see that ISA-DR achieves better energy performance at higher utilization level. This is because with higher utilization, the slowdown factors calculated are larger than those with lower utilization. This can result in more free run time when tasks exhibit variable execution times, which in turn can be reclaimed to reduce the execution speed of later tasks and thus leads to more energy savings.

In summary, ISA-FI achieves 12%–20% energy gains over USFI. SFI is a very effective policy, and when applied in ISA, achieves 20%–30% performance improvement over USFI and ISA-FI. Moreover, SFI performs better with coarse-grained task set, where most tasks are selected from the range of long period. When tasks do not always execute at their WCETs, ISA-DR is able to save additional energy (up to 20%–30% on average), compared with the ISA algorithm owing to its dynamic slack reclaiming mechanism.

VII. Additional Application of Our Approaches

The primary use of our algorithms is certainly to compute optimal speeds for nonpreemptive tasks running on variable-speed uniprocessors. In this section, we demonstrate an additional application of our approach by applying the technique to dynamic modulation scaling (DMS) [26] and show its usefulness in saving communication energy. DMS and DVS are rooted in similar scaling techniques; both exploit the presence of a convex energy-delay curve. While DVS trades processor speeds for reduced energy consumption, modulation level is one of the radio control knobs that can be scaled to minimize the run-time energy of communication subsystems at the cost of increased message transmission time. We give a brief overview of DMS below; detailed descriptions can be found in [26].

TABLE II
Simulation Settings (Taken From [26])

Parameters	Meaning	Value
R_s	Symbol rate(number of symbols transmitted per second)	1 MHz
C_S	Transmit power	12 nJ
C_E	Electronics power	15 nJ
$b_{\min}$	Minimum modulation level	2
$b_{\max}$	Maximum modulation level	8

In digital wireless communication systems, bits are coded into channel symbols to transmit information. DMS tunes the *modulation level* b, often expressed in number of bits per channel symbol, to trade off energy consumption against transmission delay. The descriptions of the various symbols used in the sequel are given in Table II. The transmission delay in seconds per bit is $T_{bit} = 1/(b \cdot R_s)$.

Assuming 2^b-QAM [30] as the modulation scaling scheme, the energy associated with the transmission of 1 bit is given by

$$E_{bit} = \left[C_S \cdot (2^b - 1) + C_E\right] \cdot \frac{1}{b}. \tag{10}$$

Note that the energy consumption computed in (10) is a convex function of the modulation level b. Schurgers *et al.* [26] pointed out one major difference between DMS and DVS: A change in modulation level requires handshaking between the sender and the receiver. The sender cannot decide on a new modulation setting midway through the packet transmission. This implies that packet scheduling must be nonpreemptive, which makes our results relevant.

Our idea is to use ISA with FI (ISA-FI) and ISA with SFI (ISA-SFI) to compute modulation levels for energy-efficient transmission of real-time data streams in DMS, just as they are used to compute static task speeds in DVS. Thus, both ISA-FI and ISA-SFI would compute one individual modulation level for each stream. One thing to note is that modulation level b in DMS has discrete values. We can use the closest integer that is larger than each value computed by our algorithms to be the

TABLE III
NUMBER OF STREAMS WITH GIVEN PERIOD

	20 ms	25 ms	40 ms	50 ms
$U = 0.16$	1	1	2	1
$U = 0.275$	2	2	3	1
$U = 0.38$	3	3	2	3
$U = 0.45$	4	3	2	4
$U = 0.58$	4	4	5	4

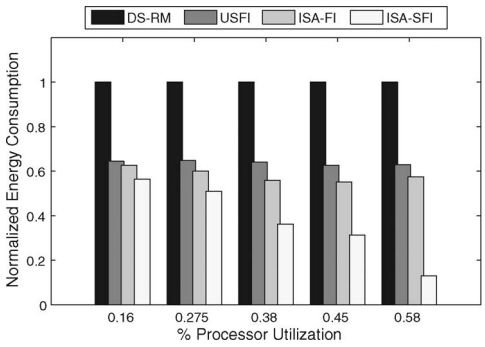

Fig. 9. Relative energy for real-time packet scheduling.

modulation level. In our experiments, the modulation level lies in the range $[b_{min}, b_{max}]$, as shown in Table II.

To evaluate the performance of our techniques, we adapt the various scheduling scenarios given in [26] to suit RM-based nonpreemptive scheduling algorithms by reducing the number of streams in each scenario to decrease the system workloads, but keeping the task periods and message sizes intact.[2] Each row in Table III represents a scenario with a particular utilization that has certain number of packet streams with a given period. For example, the third scenario has three streams with a period of 20 ms, three with a period of 25 ms, two with a period of 40 ms, and three with a period of 50 ms. The deadline of each stream is equal to its corresponding period. We assume all packets are of maximum size. According to Schurgers *et al.* [26], the worst-case transmission time for each stream packet is 1 ms. The total utilization U for the third scenario in Table III is thus $3/20 + 3/25 + 2/40 + 3/50 = 0.38$.

Schurgers *et al.* [26] sketched a real-time packet scheduling algorithm based on the nonpreemptive EDF algorithm of Jeffay *et al.* [10]. Since the algorithm computes one single static scaling factor, called α_{static}, assuming all packets are of maximum size, it amounts to using the H speed calculated by the EDF version of the `DS` algorithm [34]. The algorithm can certainly be made RM-based, which we did in our experiments in order to compare with our algorithms. We call this algorithm `DS-RM` as it uses the RM version of the `DS` algorithm to compute a single speed (H speed) for a set of data streams.

Fig. 9 plots the energy consumption of `USFI` and our algorithms, normalized against `DS-RM` at different utilization rates. It can be seen from the figure that `USFI` and `ISA-FI` outperform `DS-RM` at all utilization rates. The reason is that both `USFI` and `ISA-FI` compute one modulation level for each stream in a scenario, and these modulation levels are lower than those computed by `DS-RM`. `ISA-FI` outperforms `USFI` on energy savings due to that it can compute lower modulation levels than `USFI`. Note that the energy consumption of `ISA-FI` does not change much with different utilization rates. This is because given the scenarios in Table III, most of the calculated modulation levels happen to be very close. Among the four algorithms we investigate, `ISA-SFI` gives the best performance. This is mainly due to the SFI policy.

[2]The original scenarios given in [26] are meant to be scheduled by an EDF-based nonpreemptive algorithm [10], and all the scenarios have rather high utilization rates.

VIII. CONCLUSION

Energy management is one of the key issues in the design of modern real-time mobile and embedded systems. In this paper, we consider energy-efficient real-time scheduling where task preemption is impossible or prohibitively expensive and propose a novel ISA. Our experimental results show that the proposed scheme achieves considerable energy gains compared to existing representative algorithms. The SFI policy, which avoids unnecessary speed promotion when task blocking occurs, is found to be very effective in reducing energy consumption while incurring small overhead. Considering that the task's actual execution time is usually less than its WCET, a dynamic slack reclamation scheme is proposed to reclaim unused runtime and dispatch it to later jobs to further decrease their execution speed, which in turn can result in more energy savings.

For future work, we plan to extend our method to a wider system scope by taking the energy consumption of other system components into consideration. As multiprocessor systems are increasingly used in real-time environments, we also plan to extend our current solutions to multiprocessor systems in the future.

ACKNOWLEDGMENT

The author J. Li would like to thank Mr. W. Wei from HUST for his help in the simulation experiments. The authors would also like to thank anonymous reviewers for their comments on improving the quality of this paper.

REFERENCES

[1] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2005, pp. 213–223.

[2] L. Almeida and J. Fonseca, "Analysis of a simple model for non-preemptive blocking-free scheduling," in *Proc. Euromicro Conf. Real-Time Syst.*, 2001, pp. 1–8.

[3] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Proc. Real-Time Syst. Symp.*, 2006, pp. 313–322.

[4] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. IEEE Real-Time Syst. Symp.*, 2001, pp. 95–105.

[5] C. Chang, J. Sheu, Y. Chen, and S. Chang, "An obstacle-free and power-efficient deployment algorithm for wireless sensor networks," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 4, pp. 795–806, Jul. 2009.

[6] J.-J. Chen, C.-L. Yang, H.-R. Hsu, and T.-W. Kuo, "Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2008, pp. 13–23.

[7] H. Du, X. Hu, and X. Jia, "Energy efficient routing and scheduling for real-time data aggregation in WSNS," *Comput. Commun.*, vol. 29, no. 17, pp. 3527–3535, Nov. 2006.

[8] N. Fisher and S. Baruah, "A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines," in *Proc. Euromicro Conf. Real-Time Syst.*, 2005, pp. 117–126.
[9] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1702–1714, Dec. 1999.
[10] K. Jeffay, D. Stanat, and C. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proc. IEEE Real-Time Syst. Symp.*, 1991, pp. 129–139.
[11] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proc. Des. Autom. Conf.*, 2005, pp. 111–116.
[12] R. Jejurikar and R. Gupta, "Energy aware non-preemptive scheduling for hard real-time systems," in *Proc. Euromicro Conf. Real-Time Syst.*, 2005, pp. 21–30.
[13] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1024–1037, Jun. 2006.
[14] Y. Lee, K. Reddy, and C. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Proc. Euromicro Conf. Real-Time Syst.*, 2003, pp. 105–112.
[15] W. Luo, X. Qin, X. Tan, K. Qin, and A. Manzanares, "Exploiting redundancies to enhance schedulability in fault-tolerant and real-time distributed systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 3, pp. 626–639, May 2009.
[16] J. Mao, C. Cassandras, and Q. Zhao, "Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints," *IEEE Trans. Mobile Comput.*, vol. 6, no. 6, pp. 678–688, Jun. 2007.
[17] B. Mochocki, X. Hu, and G. Quan, "Transition-overhead-aware voltage scheduling for fixed-priority real-time systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 2, pp. 11–40, Apr. 2007.
[18] A. M. Mok and W.-C. Poon, "Non-preemptive robustness under reduced system load," in *Proc. IEEE Real-Time Syst. Symp.*, 2005, pp. 200–209.
[19] M. Park, "Non-preemptive fixed priority scheduling of hard real-time periodic tasks," in *Proc. ICCS*, vol. 4490, *Lecture Notes in Computer Science*, pp. 881–888.
[20] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. SOSP*, 2001, pp. 89–102.
[21] G. Quan and X. Hu, "Minimum energy fixed-priority scheduling for variable voltage processors," in *Proc. DATE*, 2002, pp. 782–787.
[22] G. Quan, L. Niu, X. Hu, and B. Mochocki, "Fixed priority scheduling for reducing overall energy on variable voltage processors," in *Proc. IEEE Real-Time Syst. Symp.*, 2004, pp. 309–318.
[23] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemption points," in *Proc. IEEE Real-Time Syst. Symp.*, 2006, pp. 212–224.
[24] X. Ruan, S. Yin, A. Manzanares, M. Alghamdi, and X. Qin, "A message-scheduling scheme for energy conservation in multimedia wireless systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 2, pp. 272–283, Mar. 2011.
[25] S. Saewong and R. Rajkumar, "Practical voltage-scaling for fixed-priority rt-systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2003, pp. 106–114.
[26] C. Schurgers, V. Raghunathan, and M. Srivastava, "Power management for energy-aware communication systems," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 2, no. 3, pp. 431–447, Aug. 2003.
[27] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.
[28] V. Swaminathan and K. Chakrabarty, "Real-time task scheduling for energy-aware embedded systems," *J. Franklin Inst.*, vol. 338, no. 6, pp. 729–750, Sep. 2001.
[29] V. Swaminathan, C. Schweizer, K. Chakrabarty, and A. Patel, "Experiences in implementing an energy-driven task scheduler in RT-linux," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2002, pp. 229–238.
[30] W. Webb and R. Steele, "Variable rate qam for mobile radio," *IEEE Trans. Commun.*, vol. 43, no. 7, pp. 2223–2230, Jul. 1995.
[31] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. Annu. Symp. Found. Comput. Sci.*, 1995, pp. 374–382.
[32] G. Yao, G. Buttazzo, and M. Bertogna, "Bounding the maximum length of non-preemptive regions under fixed priority scheduling," in *Proc. RTCSA*, 2009, pp. 351–360.
[33] F. Zhang and S. Chanson, "Processor voltage scheduling for real-time tasks with non-preemptible sections," in *Proc. 22nd IEEE Real-Time Syst. Symp.*, 2002, pp. 235–245.
[34] F. Zhang and S. Chanson, "Blocking-aware processor voltage scheduling for real-time tasks," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 2, pp. 307–335, May 2004.
[35] Y. Zhang and K. Chakrabarty, "Dynamic adaptation for fault tolerance and power management in embedded real-time systems," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 2, pp. 336–360, May 2004.
[36] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2007, pp. 225–235.
[37] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *Proc. ICCAD*, 2004, pp. 35–40.
[38] J. Zhuo and C. Chakrabarti, "System-level energy-efficient dynamic task scheduling," in *Proc. Des. Autom. Conf.*, 2005, pp. 628–631.

Jianjun Li is currently working toward the Ph.D. degree at the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China.

His research interests include real-time systems, real-time databases, and energy-aware real-time scheduling.

LihChyun Shu (M'94) received the Ph.D. degree in computer sciences from Purdue University, West Lafayette, IN, in 1994.

He is a Professor in the College of Management, National Cheng Kung University, Tainan, Taiwan. From August 2011, he also serves as the Dean of College of Information and Engineering of Chang Jung Christian University, Tainan, Taiwan. His research interests include real-time systems, data stream and process mining, and location-based query processing.

Dr. Shu is a member of the IEEE Computer Society.

Jian-Jia Chen (M'05) received the B.S. degree from the Department of Chemistry at National Taiwan University, in 2001, and the Ph.D. degree from the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, in 2006. After finishing the compulsory civil service in Dec. 2007, between Jan. 2008 and April 2010, he was a Postdoc Researcher at Computer Engineering and Networks Laboratory (TIK) in Swiss Federal Institute of Technology (ETH) Zurich, Switzerland.

He joined the Department of Informatics at Karlsruhe Institute of Technology, Karlsruhe, Germany as a Junior Professor for the Institute for Process Control and Robotics in 2010. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing.

Dr. Chen received Best Paper Awards from ACM Symposium on Applied Computing in 2009 and IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2005.

Guohui Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1999.

He was promoted to a Full Professor in 2004 and currently acts as the Vice Dean of the School of Computer Science and Technology in HUST. From 2009, he is also affiliated with the School of Mathematic and Computer Science, Wuhan Polytechnic University, Wuhan, China. His research interests mainly include real-time systems, mobile computing, and advanced data management.