Energy-Aware Task Scheduling With Task Synchronization for Embedded Real-Time Systems

Ravindra Jejurikar, *Student Member, IEEE*, and Rajesh Gupta, *Fellow, IEEE*

***Abstract*—Slowdown factors determine the extent of slowdown that a computing system can experience based on functional and performance requirements. Dynamic voltage scaling (DVS) of a processor based on slowdown factors can lead to considerable energy savings. This paper addresses the problem of DVS in the presence of task synchronization. Tasks synchronize to enforce mutually exclusive access to the shared resources and can be blocked by lower priority tasks. Task slowdown factors that guarantee meeting all task deadlines are computed. Both static and dynamic priority scheduling viz. rate monotonic (RM) scheduling and earliest deadline first (EDF) scheduling, respectively, are studied.**

***Index Terms*—Frequency inheritance, low power, processor scheduling, real-time systems, task synchronization.**

I. Introduction

POWER is one of the important metrics for optimization in the design and operation of embedded systems. The two primary ways to reduce processor power consumption are shutdown and slowdown. Slowdown using frequency and voltage scaling is known to be more effective in reducing the power consumption due to the quadratic dependence of power on voltage. Note that scaling the frequency and voltage of a processor leads to energy gains at the cost of increased execution time for a job. In real-time systems, we want to minimize energy consumption while adhering to task deadlines. Power reduction and meeting deadlines are often contradictory goals, and we have to judiciously manage power and time to achieve our goal of minimizing energy.

In this paper, we focus on the system level power management via the computation of task slowdown factors. A slowdown factor is the normalized operating frequency that determines the processor speed at runtime. The computation of slowdown factors can be classified into static slowdown, computed off-line based on task characteristics, and dynamic slowdown, computed using on-line task execution information. This work focuses on the computation of static slowdown factors under synchronization constraints. We consider a real-time system where tasks arrive periodically and have deadlines. The tasks are scheduled on a single processor system based on a preemptive scheduling policy. Tasks synchronize to access shared resources in a mutually exclusive manner. We address both static and dynamic priority scheduling policies and study the rate monotonic (RM) scheduling and the earliest deadline first (EDF) [1], [2] scheduling, respectively.

Most of the earlier works on energy-aware scheduling consider independent task sets. Among the earliest works, Yao *et al.* [3] presented an optimal off-line algorithm to schedule a job set with specified arrival times and deadlines on a continuous voltage processor. The analysis and correctness of the algorithm is based on an underlying EDF scheduler, which is an optimal scheduling policy [1]. Kwon and Kim [4] extend the algorithm proposed by Yao *et al.* to compute optimal slowdown factors for the case of discrete voltage levels. The same problem, under fixed priority scheduling, is addressed by Quan and Hu [5], [6] and shown to be non-polynomial-hard (NP-hard) [7]. Low power scheduling in the context of real-time systems has also been addressed. Shin *et al.* [8] have computed uniform slowdown factors for an independent periodic task set. In this technique, RM analysis is performed to compute a constant static slowdown factor for the processor. Gruian [9] observed that performing more iterations can result in better slowdown factors for the individual task types. Under the EDF scheduling policy, a constant slowdown to maximize the processor utilization is known to be a feasible solution [10]. The computation of optimal slowdown factors for tasks with different power consumption characteristics is addressed by Aydin *et al.* [11]. Dynamic slowdown techniques, which reclaim the slack arising from early completion of tasks (compared to the worst case) is addressed in [10], [12], and [13]. Energy-aware scheduling of periodic and aperiodic tasks based on sporadic servers is presented in [14] and [15]. The problem of maximizing the system value for a given energy budget, as opposed to minimizing the total energy, is addressed in [16] and [17].

Scheduling of task graphs on multiple processors has also been considered. Luo *et al.* [18]–[20] have addressed the scheduling of periodic and aperiodic task graphs in a distributed system. Nonpreemptive scheduling of a task graph on a multiprocessor system is considered by Gruian and Kuchcinski [21]. A framework for task scheduling and voltage assignment of dependent tasks on a multiprocessor system based on integer programming is presented in [22] and [23]. Energy-aware scheduling in distributed systems using static and dynamic slowdown is addressed by Zhu *et al.* [24]–[26].

Though DVS is well studied, very few works address slowdown in the presence of task synchronization. Most real-life applications have shared resources in the system and mutually

Manuscript received December 3, 2002; revised August 7, 2003, March 11, 2004, October 10, 2004, and December 1, 2004. This work was supported in part by the National Science Foundation under Award CCR-0098335 and in part by the Semiconductor Research Corporation under Contract 2001-HJ-899. This paper was recommended by Associate Editor M. F. Jacome.

R. Jejurikar is with the Department of Information and Computer Science, University of California, Irvine, CA 92697 USA (e-mail: jezz@ics.uci.edu).

R. Gupta is with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093 USA.

Digital Object Identifier 10.1109/TCAD.2005.855964

exclusive access to shared resources can lead to priority inversion. If a lower priority job has access to a resource, a higher priority job requesting the resource is blocked and can miss its deadline. Scheduling in the presence of task synchronization is NP-hard [27]–[29], and sufficient feasibility tests [30], [31] have been studied. Based on these feasibility test, the computation of a constant slowdown is addressed in [32] and [33]. In a similar work, Zhang and Chanson have addressed the slowdown of task with nonpreemptive sections and presented the dual-speed (DS) algorithm [34]. We show that the DS algorithm only permits nonpreemptive sections within tasks and is not applicable to scheduling with task synchronization. In this work, we propose an algorithm to compute slowdown factors for individual tasks under synchronization constraints. Note that any known resource access protocol can be used with the proposed slowdown algorithm.

The rest of the paper is organized as follows: Section II formulates the problem with motivating examples. In Section III, we present a generic algorithm to compute slowdown factors. Computation of slowdown factors for EDF and RM scheduling is discussed in Sections III-B and III-C, respectively. Experimental results are given in Section IV, and Section V concludes the paper with future directions.

II. Preliminaries

This section describes the system model and formulates the problem. We motivate the problem by showing that earlier proposed algorithms cannot be used under task synchronization constraints.

A. System Model

A task set of n periodic real-time tasks is represented as $\Gamma = \{\tau_1, \ldots, \tau_n\}$. A three-tuple $\tau_i = \{T_i, D_i, C_i\}$ represents each task, where T_i is the period of the task, D_i is the relative deadline with $D_i \leq T_i$, and C_i is the worst case execution time (WCET) of the task at the maximum processor speed, given that it is the only task running in the system. Each invocation of the task is called a job and the kth invocation of task τ_i is denoted as $\tau_{i,k}$. A task set is said to be feasible if all jobs meet their deadline. The processor utilization for the task set $U = \sum_{i=1}^{n} C_i/T_i \leq 1$ is a necessary condition for the feasibility of any schedule [1]. The system has a set of shared resources that are accessed by the tasks in a mutually exclusive manner. Common synchronization primitives to enforce exclusive access include semaphores, locks, and monitors [35]. We assume that semaphores are used for task synchronization. All tasks are assumed to be preemptive with the constraint that the access to the shared resources must be serialized. When a task has been granted access to a shared resource, it is said to be executing in its critical section [35]. We assume that critical sections of a task are properly nested [31], wherein if two critical sections intersect, then one lies completely within the other. The kth critical section of task τ_i is represented as $z_{i,k}$. We say that a task is blocked if the task has to wait for a lower priority task to release a shared resource and the task holding the resource is called the blocking task. Note that the amount of time that a task is blocked is referred to as the task blocking time. Each task specifies the access to the shared resources and the WCET of each critical section. With the specified task information and a given resource access protocol, the maximum blocking time for a task can be computed. Let B_i be the maximum blocking time for task τ_i under a specified resource access protocol.

The tasks are scheduled on a single processor that supports variable frequency and voltage levels. The processor slowdown results in energy reduction and the slowdown factor can be viewed as the normalized frequency. At a given instance, it is the ratio of the scheduled frequency to the maximum frequency of the processor. In this paper, we consider assigning slowdown factors to tasks as opposed to a constant slowdown to better exploit the slack in the system. We assume the same slowdown factor for all instances of a task and is referred to as a uniform task slowdown. The speed of the processor can be varied over a discrete range with the execution time of a job inversely proportional to the processor speed. Our aim is to determine the processor speed for each task such that all tasks meet their deadlines and the energy consumption is minimized. The time and energy required to change the processor speed are very small compared to that of a task. We assume that the voltage change overhead, similar to the context switch overhead, is incorporated in the task execution time.

B. Variable-Speed Processors

A wide range of processors such as the Intel StrongARM processors [36], Intel XScale [37], and Transmeta Crusoe [38] support variable voltage and frequency levels. Voltage and frequency levels are tightly coupled and a change in the processor speed includes a change in the operating frequency as well as a proportionate change in the voltage level. The important point to note is that, when we perform a slowdown, we change both the frequency and voltage of the processor. We assume that the processor supports discrete voltage and frequency levels with the minimum and maximum frequency represented by $f_{\min}$ and $f_{\max}$, respectively. We normalize the speed to the maximum speed to have discrete slowdown factors in the range $[\eta_{\min}, 1]$, where $\eta_{\min} = f_{\min}/f_{\max}$.

C. Motivating Example

Consider a simple real-time system with two periodic tasks having the parameters

$$\tau_1 = \{5, 5, 2\} \quad \tau_2 = \{40, 40, 4\}. \tag{1}$$

The two tasks have a shared resource, and access to the shared resource is granted through a semaphore S. The critical section for task τ_1 is $z_{1,1} = [1.5, 2]$ (within the task execution interval) and that for τ_2 is $z_{2,1} = [0, 3.0]$. The critical sections can block tasks, and the maximum blocking time for the tasks is $B_1 = 3$ and $B_2 = 0$ at full speed. Fig. 1(a) shows the tasks at their arrival time with their workload at maximum speed. We consider the case where the lower priority task τ_2 arrives at time $t = 0$ and the higher priority task τ_1 arrives at time $t = 1$. The task set, scheduled on a single processor, is feasible at the maximum speed under both EDF and RM scheduling

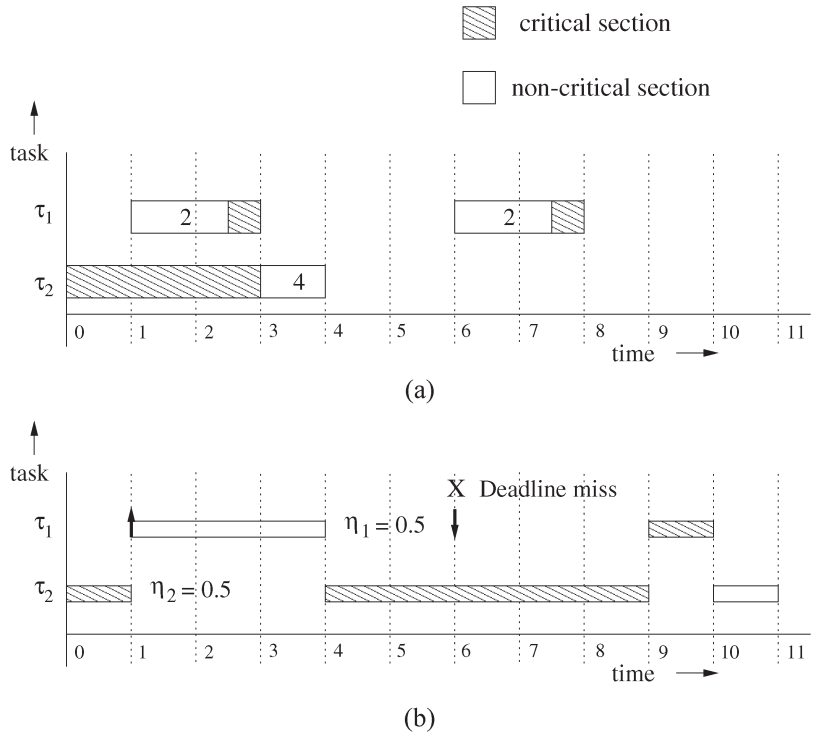

Fig. 1. Motivation for static slowdown with task synchronization. (a) Task arrival times and deadlines (period = deadline) with critical sections. (b) EDF schedule: Processor utilization as the static slowdown factor $\eta = (2/5) + (4/40) = 0.5$, and job τ_1 misses the deadline. (RM scheduling also results in the same slowdown factors and a similar task schedule.)

policies. We show that the blocking time cannot be ignored while computing slowdown factors.

An independent task set scheduled by the EDF policy is feasible at a slowdown equal to the processor utilization, $U = (2/5) + (4/40) = 0.5$. Though an independent task set would allow a uniform slowdown of $\eta = 0.5$, the same slowdown cannot be used with task synchronization. As seen in Fig. 1(b), the blocking time increases with slowdown, and task τ_2 blocks τ_1 for 5.0 time units, which results in task τ_1 missing the deadline. RM scheduling also results in the same task slowdown factors for the example task set. To keep the task set feasible under RM scheduling, 20 units of computation is needed in 40 time units, allowing for a uniform slowdown of $\eta = (20/40) = 0.5$, for an independent task set. This results in a similar schedule as shown in Fig. 1(b), where task τ_1 is blocked by task τ_2 for 5.0 time units and misses its deadline. Thus, we need to consider the blocking time in the computation of task slowdown factors.

D. Dual-Speed (DS) Algorithm

Zhang and Chanson [34] have addressed task slowdown in the presence of nonpreemptive critical sections, whereby a critical section blocks all higher priority tasks. The authors have proposed the DS algorithm, which computes two speeds for the task system. A low speed L is computed based on an analysis for an independent task set, and a high speed H is computed taking into account the worst case blocking time for each task. The system begins execution at speed L and switches to speed H when tasks are blocked. The details of the transition between speeds H and L are described in [34]. Note that the correctness of the DS algorithm requires that the critical sections be nonpreemptive (to ensure the appropriate speed transitions). We show that the DS algorithm can lead to tasks missing the deadline, if used with task synchronization (protocols). Also note that nonpreemption (within critical sections) is not desirable since this can increase the task blocking time [31] and result in a higher speed than that required with task synchronization.

Consider that the task set described in (1) is scheduled by the DS algorithm under the EDF scheduling policy. Under the DS algorithm, the processor utilization (at maximum speed) is used as speed L and speed H is derived from the feasibility analysis with blocking, as described in [34]. Speeds L and H for the given task set, based on EDF scheduling, are $L = (2/5) + (4/40) = 0.5$ and $H = (2/5) + (3/5) = 1.0$. The task arrival times are shown in Fig. 2(a). The lower priority task (τ_2) arrives earlier (time $t = 0$) than the higher priority task τ_1 (arrival time $t = 1$). At time $t = 0$, the system begins task execution at the lower speed L, and task τ_2 enters the critical section. Task τ_1 arrives at time $t = 1$, which is blocked due to the nonpreemptive nature of the critical section. When a task is blocked, the system enters the high speed $(H = 1.0)$ and the blocking critical section and the entire blocked task execute at the high speed (H) to meet all task deadlines. The feasible schedule under the DS algorithm is shown in Fig. 2(b).

However, the DS algorithm can result in a deadline miss with task synchronization. Under task synchronization, a critical section can be preempted as long as the critical sections (using the same resource) are serialized. We show a task schedule based on the priority ceiling protocol (PCP), where a task can preempt a critical section and is not blocked until it tries to enter a critical section. Fig. 2(c) shows the task schedule under PCP.

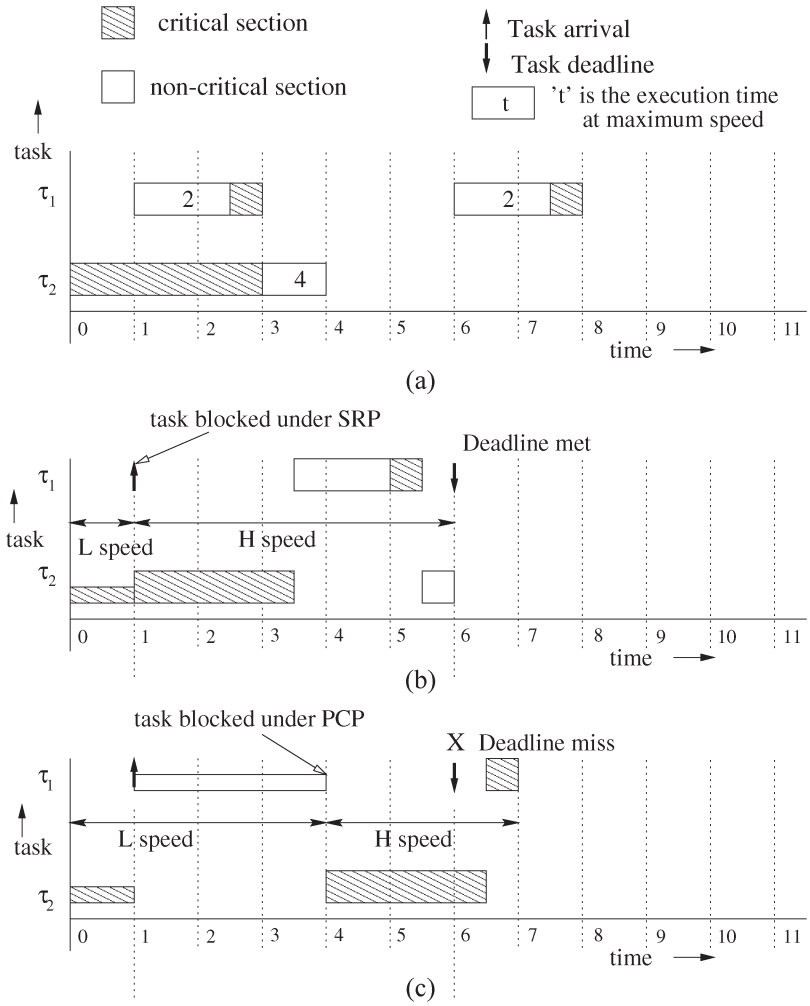

Fig. 2. Motivation for static slowdown with task synchronization. (a) Task arrival times and deadlines (period = deadline) with critical sections. (b) EDF schedule: DS schedule with nonpreemptive critical sections, and all tasks meet the deadline. (c) EDF schedule: DS schedule under the PCP, and task τ_1 misses the deadline.

Task τ_2 arrives at time $t = 0$ and enters the critical section. The higher priority task τ_1 arrives at $t = 1$ and preempts task τ_2 to execute its noncritical section. Since no task is blocked, the system continues execution at speed L. At time $t = 4$, task τ_2 is blocked as it tries to enter the critical section that is in use by task τ_2, and the system switches to speed H. The blocking critical section of task τ_2 and the remaining execution of task τ_1 is executed at speed H. However, it can be seen that task τ_2 completes only at time $t = 7$ and misses its deadline of $t = 6$. Thus, we see that the correctness of the DS algorithm is dependent on the critical sections being nonpreemptive. In this work, we present an energy-efficient slowdown algorithm that is applicable to scheduling with any task synchronization protocol.

III. Computation of Task Slowdown Factors

In this work, we compute task level slowdown factors in the presence of task synchronization with a uniform slowdown for each task. We first present a generic slowdown algorithm that is used to compute slowdown factors under different scheduling policies.

A. Generic Slowdown Algorithm

The slowdown factors for the tasks are based on the feasibility analysis and differ with the scheduling policy. For the scheduling policies considered in this paper, the algorithms are similar, and we present a generic slowdown algorithm to compute static slowdown factors. The order in which tasks are assigned slowdown factors is identical and is captured by the generic slowdown algorithm. The algorithm is similar to the work by Gruian [9], which considers slowdown of independent tasks scheduled by the RM scheduling policy. Our work is an extension of [9], and we address task slowdown with synchronization under both fixed and dynamic priority scheduling.

Algorithm 1. Compute static slowdown factors $(\tau_1, \ldots, \tau_n)$

1: {Given tasks are in nonincreasing order of their relative deadline}
2: $q = 1$; {initialization}
3: WHILE $(q \leq n)$ DO
4: FOR $(i = q;\ i \leq n;\ i++)$ DO
5: Compute the candidate slowdown factor η_i based on feasibility analysis;
6: END FOR

7: $\eta_m = \max_{i=q}^{n}(\eta_i)$ {Compute the maximum candidate slowdown factor}
8: FOR $(i = q;\ i \leq m;\ i++)$ DO
9: $\eta_i = \eta_m$;
10: END FOR
11: $q = m + 1$;
12: END WHILE

The generic slowdown algorithm to compute task slowdown factors is described in Algorithm 1. The algorithm assumes that tasks are in nondecreasing order of their relative deadline, thus, $j > i$ implies $D_j \geq D_i$. The slowdown factors are computed iteratively from tasks with the smallest to the largest index (relative deadline). An index q points to the first task that has not been assigned a slowdown factor. Initially, $q = 1$ and the slowdown factor for all tasks is unassigned. Based on the scheduling analysis, a slowdown factor is computed for each unassigned task and is referred to as the candidate slowdown factor for the task. The candidate slowdown factor for each unassigned task is computed in line 5. There is a task with index m for which the candidate slowdown factor is the largest among all tasks. This maximum is computed in line 7. Note that this is not necessarily the last task with index n. Unassigned tasks up to index m are assigned a slowdown factor of η_m as shown in lines 8–10. This completes one iteration of the algorithm, and we set $q = m + 1$ for the next iteration. The algorithm terminates when all tasks have been assigned a slowdown factor. Note that the computation of candidate slowdown factors is based on the feasibility analysis. The rest of the algorithm is independent of the scheduling policy. The computation of the candidate slowdown factor for each scheduling policy is presented next.

B. Slowdown Factors and EDF Scheduling

Resource access protocols have been designed to bound the blocking time, and sufficient feasibility tests have been proposed. Resource access protocols such as dynamic priority ceiling protocol (DPCP) [39], stack resource protocol, and minimal stack resource protocol [30] have been designed to handle tasks with dynamic priorities, which encompasses EDF scheduling. Our slowdown algorithm is independent of the resource access protocol and any given protocol can be used to manage the access to the resources. The knowledge of the maximum blocking time is important in testing the feasibility of the task set and computing slowdown factors. Let B_i denote the maximum blocking time (under no slowdown) for task τ_i under a given resource access protocol. First, we state the feasibility test which is then used to compute task slowdown factors.

1) Feasibility Test: Let $\Gamma = \{\tau_1, \ldots, \tau_n\}$ be the tasks in the system ordered in nondecreasing order of their relative deadline and B_i be the maximum blocking time for task τ_i. The task set is feasible under the EDF scheduling policy if it satisfies the condition

$$\forall i,\ i = 1, \ldots, n, \quad \frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{C_k}{D_k} \leq 1. \tag{2}$$

2) Slowdown Factor Computation: We first describe the computation of candidate slowdown factor for a task that determines the task slowdown factors. Note that the task slowdown factors are computed by the generic slowdown algorithm given by Algorithm 1. The candidate slowdown factor for task τ_i, used in line 5 of the algorithm, is computed using

$$\left(\sum_{1 \leq r < q} \frac{1}{\eta_r} \frac{C_r}{D_r}\right) + \frac{1}{\eta_i}\left(\frac{B_i}{D_i} + \sum_{q \leq p \leq i} \frac{C_p}{D_p}\right) = 1. \tag{3}$$

This is an extension of the feasibility test considering task slowdown factors. Tasks with index less than q have been assigned a slowdown factor, and their contribution under slowdown is considered. A uniform slowdown of η_i is computed for the unassigned tasks and their blocking critical sections while ensuring the feasibility of task τ_i.

C. Slowdown Factors and RM Scheduling

RM scheduling is an optimal fixed priority scheduling scheme [40] and is well studied. RM scheduling assigns task priorities based on the task period, with the priority inversely proportional to task period. If the deadlines of the tasks are not equal to their period, deadline monotonic (DM) scheduling is an optimal fixed priority scheduling policy. Resource access protocols such as priority inheritance protocol and PCP [31] have been proposed for fixed priority systems. First, we describe the feasibility test under RM (and DM) scheduling policy, followed by the computation of task slowdown factors.

1) Feasibility Test: Lehoczky *et al.* [41] have shown that the feasibility analysis is needed only at discrete points, called the scheduling points. Given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with tasks in nonincreasing order of their priority, the set of scheduling points for task τ_i is defined by

$$S_i = \left\{ kT_j | j = 1, \ldots, i;\ k = 1, \ldots, \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\} \tag{4}$$

when the period is the same as the deadline, i.e., $T_i = D_i$ for each task τ_i. If D_i is less than T_i for some task τ_i, we consider only the scheduling points up to the task deadline D_i. This set of scheduling point S_i' is defined as

$$S_i' = \{(t \in S_i) \wedge (t < D_i)\} \cup \{D_i\}. \tag{5}$$

The feasibility test in the presence of blocking time is given by Sha *et al.* [31]. Each task τ_i can be scheduled without violating its deadline, if there exists one or more scheduling point $S_{ij} \in S_i$ that satisfies

$$B_i + \sum_{k=1}^{i} C_k \left\lceil \frac{S_{ij}}{T_k} \right\rceil \leq S_{ij} \tag{6}$$

where B_i is the maximum blocking time (at full speed) for task τ_i under a given resource access protocol.

2) Slowdown Factor Computation: The task slowdown factors are computed using the generic slowdown algorithm described in Algorithm 1. In line 5 of the algorithm, a candidate

slowdown factor is computed for each task that is not yet assigned a slowdown factor. The computation of the task candidate slowdown factor is based on the slowdown factor η_{ij}, corresponding to each scheduling point $S_{ij} \in S_i$. The computation of η_{ij} is given as

$$\left(\sum_{1\le r<q} \frac{C_r}{\eta_r}\left\lceil\frac{S_{ij}}{T_r}\right\rceil\right) + \frac{1}{\eta_{ij}}\left(B_i + \sum_{q\le p\le i} C_p\left\lceil\frac{S_{ij}}{T_p}\right\rceil\right) = S_{ij}. \tag{7}$$

Note that the tasks τ_r, $1 \le r < q$ have an assigned slowdown factor η_r, and their workload under slowdown is considered. We compute a slowdown factor η_{ij} for the unassigned tasks, which results in S_{ij} being a feasible scheduling point. For task τ_i, the best scheduling point from the energy point of view is the one that results in the smallest η_{ij}. Thus, taking the minimum over the slowdown factors corresponding to each scheduling point of task τ_i gives the slowdown factor η_i, written as

$$\eta_i = \min_j(\eta_{ij}). \tag{8}$$

We show that the slowdown factors computed by the generic slowdown algorithm, under both scheduling policies, are in nonincreasing order. The claim along with the proof is given below.

Lemma 1: Given the tasks are ordered in a nondecreasing order of their relative deadline, the task slowdown factors computed by the generic slowdown algorithm are in a nonincreasing order.

Proof: The claim follows from the task ordering and the sequence in which task are assigned slowdown factors. Note that the generic slowdown algorithm computes a candidate slowdown factor for each unassigned task based on the feasibility test of the given scheduling policy. We would like to emphasize that the computation of a candidate slowdown factor for a task [(3) and (7)] assumes the same slowdown factor for all unassigned tasks. In each iteration of the algorithm, the maximum over the task candidate slowdown factors η_m is assigned to the unassigned tasks up to index m. Since the maximum over all candidate slowdown factors is assigned to some tasks, this can only lower the candidate slowdown factor in future iterations. Hence, the candidate slowdown factor of each task and the maximum over the candidate slowdown factors are nonincreasing in consecutive iterations. Since the tasks are assigned a slowdown in increasing order of their index and the value of η_m is nonincreasing in consecutive iterations, the task slowdown factors are in nonincreasing order. ■

D. Frequency Inheritance

We show that the property of frequency inheritance, where a task inherits the frequency of a blocked task, is important in ensuring task deadlines. Note that the computation of the candidate slowdown factor for a task τ_i, as given by (3) and (7), assumes the same slowdown for task τ_i and its blocking section B_i. However, it is known that tasks are blocked by the critical section of tasks with larger relative deadlines [30] and a blocking task can have a smaller slowdown factor than the blocked task (by Lemma 1). A lower slowdown factor for a blocking task can increase the blocking time for the task and result in a deadline miss. Therefore, it is necessary that the slowdown of a blocking task be adjusted at runtime to avoid deadline misses.

We illustrate in Fig. 3 that tasks can miss their deadline if they do not follow frequency inheritance. We consider tasks scheduled by the EDF scheduling policy and a similar case arises under RM scheduling as well. Fig. 3(a) shows the task set represented by (1). For the example, the algorithm based on the EDF scheduling policy (3) computes task slowdown factors of $\eta_1 = 1.0$ and $\eta_2 = 0.167$. Fig. 3(b) shows the EDF schedule at the computed slowdown factors where τ_1 misses its deadline. The computation of the slowdown factor for task τ_1 assumes the same slowdown of η_1 for the blocking sections and can tolerate a blocking time of $3/1.0 = 3.0$. However, executing task τ_2 at a speed of $\eta_2 = 0.167$ can extend the blocking time to as much as $3/0.167 = 18$. As seen in the figure, task τ_1 is blocked for 17 time units, which is greater than the period (deadline) of task τ_1 and results in τ_1 missing the deadline. With frequency inheritance, task τ_2 inherits a slowdown of $\eta = \eta_1 = 1.0$, when blocking task τ_1. This bounds the blocking time, and task τ_1 meets its deadline as shown in Fig. 3(c). We later prove that the property of frequency inheritance guarantees all task deadlines for the proposed algorithms.

We now state the processor slowdown rules that incorporate frequency inheritance. The processor slowdown η depends on the executing task and the tasks that are blocked in the system. The slowdown rules are stated as follows.

1) Rule 0: $\eta = 0$ (shutdown the processor), when idle.
2) Rule 1: $\eta = \eta_i$, when τ_i is executing and not blocking any other task.
3) Rule 2: $\eta = \max(\eta_i, \max_j(\eta_j))$, when τ_i is blocking tasks τ_j. This rule enforces frequency inheritance. If the blocked task has a higher slowdown factor, this slowdown is inherited by the blocking task.

Resource access protocols like priority inheritance protocol implement priority inheritance [31] and frequency inheritance can be easily augmented to these protocols. We prove next that all tasks meet the deadline under the frequency inheritance rules for processor slowdown.

Theorem 1: Based on the EDF scheduling policy, the slowdown factors computed by the static slowdown algorithm along with the frequency inheritance rules for processor slowdown guarantee all task deadlines.

Proof: The workload (computation time) for a task τ_k at the assigned slowdown of η_k is C_k/η_k. Since the tasks are assigned a slowdown factor greater than or equal to the candidate slowdown factor computed using (3), each task τ_i, $i = 1, \ldots, n$, satisfies the condition

$$\frac{1}{\eta_i}\frac{B_i}{D_i} + \sum_{k=1}^{i}\frac{1}{\eta_k}\frac{C_k}{D_k} \le 1. \tag{9}$$

The proof is similar to the proof by Baker [30] with the consideration of task slowdown factors and is proven by contradiction. Suppose the claim is false and we let t be the first time that a job misses its deadline. Let t' be the latest time before t such

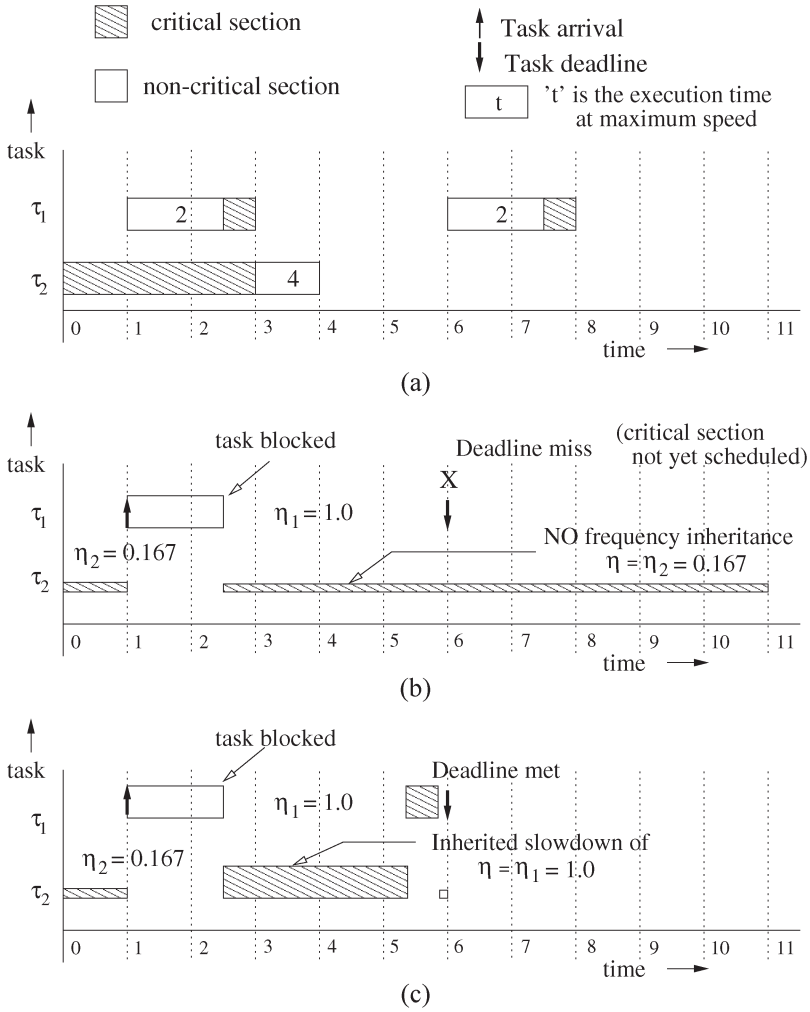

Fig. 3. Motivation for frequency inheritance. (a) Task arrival time and deadlines with critical sections. (b) Slowdown computed by the generic algorithm based on EDF scheduling, $\eta_1 = 1.0$ and $\eta_2 = 0.167$. A slowdown of $\eta = 0.167$ for task τ_2 increases the blocking time for task τ_1, and job τ_1 misses the deadline. (c) Frequency inheritance bounds the blocking time, and all tasks meet the deadline.

that there are no pending jobs with arrival times before t' and deadlines less than or equal to t. Since no requests can arrive before system start time (time $= 0$), t' is well defined. Let the length of the interval be $X = t - t'$ and $\mathcal{A}$ be the set of jobs that arrive in $[t', t]$ and have deadlines in $[t', t]$. There exists an index i such that $\mathcal{A} \subseteq \{\tau_1, \ldots, \tau_i\}$. By choice of t', there are pending requests of jobs in $\mathcal{A}$ at all times during the interval $[t', t]$ and the system is never idle in this interval. By the EDF priority assignment, the only jobs that are allowed to start in $[t', t]$ are in $\mathcal{A}$. However, jobs with a deadline greater than t, which are holding resources (at time t') required by the jobs in $\mathcal{A}$, can execute in $[t', t]$. Such jobs are denoted by the set $\mathcal{B}$. Each job $J_b \in \mathcal{B}$ executes with a deadline greater than t, and it is true that $X < D_b$. In particular, the maximum execution time at full speed of blocking jobs in $[t', t]$ is bounded by B_i [31]. By the frequency inheritance rules, the blocking critical section inherits the maximum slowdown over the jobs blocked in the system. By Lemma 1, the minimum slowdown of the jobs in $\mathcal{A}$ is η_i. Since the jobs in $\mathcal{B}$ block some job in $\mathcal{A}$, they execute at a slowdown of at least η_i. Thus, the blocking time is bounded by $(1/\eta_i)B_i$. The total execution time of the jobs in $\mathcal{A}$ and $\mathcal{B}$ is given by $(1/\eta_i)B_i + \sum_{k=1}^{i}(1/\eta_k)(\lfloor (X - D_k)/T_k \rfloor + 1)C_k$. Since a task misses its deadline at time t, the execution time for these jobs exceeds X, which is the length of the interval. Therefore

$$\frac{1}{\eta_i}B_i + \sum_{k=1}^{i}\frac{1}{\eta_k}\left(\left\lfloor\frac{X - D_k}{T_k}\right\rfloor + 1\right)C_k > X.$$

Since $X/T_k \geq \lfloor X/T_k \rfloor$, we have

$$\frac{1}{\eta_i}\frac{B_i}{X} + \sum_{k=1}^{i}\frac{1}{\eta_k}\left(\frac{X - D_k + T_k}{T_k X}\right)C_k > 1$$

$$= \frac{1}{\eta_i}\frac{B_i}{X} + \sum_{k=1}^{i}\frac{1}{\eta_k}\left(1 + \frac{T_k - D_k}{X}\right)\frac{C_k}{T_k} > 1$$

$D_k \leq X \ \forall k,\ k = 1, \ldots, i$, and we have

$$\frac{1}{\eta_i}\frac{B_i}{D_i} + \sum_{k=1}^{i}\frac{1}{\eta_k}\left(1 + \frac{T_k - D_k}{D_k}\right)\frac{C_k}{T_k} = \frac{1}{\eta_i}\frac{B_i}{D_i} + \sum_{k=1}^{i}\frac{1}{\eta_k}\frac{C_k}{D_k} > 1$$

which contradicts (9). Hence, all tasks are guaranteed to meet the deadline. ■

Theorem 2: Based on the RM scheduling policy, the slowdown factors computed by the slowdown algorithm along with the frequency inheritance rules for processor slowdown guarantee all task deadlines.

Proof: We prove the claim by contradiction, and the proof for RM scheduling is similar to that of EDF scheduling. All tasks are assigned a slowdown factor greater than or equal to the candidate slowdown factor computed using (7), and there exists a scheduling point S_{ij} for each task τ_i, $i = 1, \ldots, n$, which satisfies the condition

$$\frac{1}{\eta_i} B_i + \sum_{1 \le k \le i} \frac{C_k}{\eta_k} \left\lceil \frac{S_{ij}}{T_k} \right\rceil \le S_{ij}. \tag{10}$$

Suppose the above claim is false and an instance of task τ_i misses its deadline. Let t' be the latest time before t such that there are no pending jobs with priority less than $\mathcal{P}(\tau_i)$. Corresponding to each scheduling point S_{ij}, let $\mathcal{A}$ denote the set of jobs that arrive in $[t', t' + S_{ij}]$ with priority greater than or equal to that of τ_i, then $\mathcal{A} \subseteq \{\tau_1, \ldots, \tau_i\}$ Let $\mathcal{B}$ denote the jobs that execute in $[t', t' + S_{ij}]$ with a priority less than task τ_i. The maximum execution time of jobs in $\mathcal{B}$ at full speed is bounded by B_i [31]. By Lemma 1 and the frequency inheritance rules, the minimum slowdown of the jobs in $\mathcal{B}$ is η_i, and the blocking time is bounded by $(1/\eta_i)B_i$. Thus, the total execution time of the jobs in an interval of length S_{ij} is bounded by $(1/\eta_i)B_i + \sum_{k=1}^{i} (C_k/\eta_k)\lceil S_{ij}/T_i \rceil$. Since task τ_i misses its deadline, for each scheduling point S_{ij}, the execution time of the jobs in the interval $t' + S_{ij}$ exceeds the length of the interval. Therefore

$$\frac{1}{\eta_i} B_i + \sum_{k=1}^{i} \frac{C_k}{\eta_k} \left\lceil \frac{S_{ij}}{T_k} \right\rceil > S_{ij}$$

which contradicts (10). Hence, all tasks are guaranteed to meet the deadline. ■

E. Computation Time

The complexity of the algorithm is dictated by the computation of the candidate slowdown factors, given by (3) and (7). Each iteration of the generic slowdown algorithm computes candidate slowdown factors for the tasks that are not yet assigned slowdown factors. Under EDF scheduling, each iteration of the algorithm can be performed in linear time. While most examples require only one iteration, the generic slowdown algorithm can have n iterations in the worst case, resulting in a worst case time complexity of $O(n^2)$. For the RM scheduling policy, the computation of slowdown factors has a pseudopolynomial time complexity. This is due to the fact that the total number of scheduling points arising in the exact RM analysis are pseudopolynomial in number. However, in practice, the number of scheduling points is not large and the algorithms are efficient. Note that the slowdown algorithms are in the same time complexity class as the feasibility test for the respective scheduling policy.

F. Examples

We compute the slowdown factors under both scheduling policies for the example in Section II. The task set is $\tau_1 = \{5, 5, 2\}$ and $\tau_2 = \{40, 40, 4\}$, and their worst case blocking time is $B_1 = 3$ and $B_2 = 0$ at maximum speed.

1) EDF Scheduling: Under the EDF scheduling policy, we use (3) to compute the task slowdown factors. Initially, the slowdown factor of all tasks is unassigned, and the computation of candidate slowdown factor for each task is given as

$$\frac{1}{\eta_1}\left(\frac{3}{5} + \frac{2}{5}\right) = 1 \text{ gives } \eta_1 = \frac{5}{5} = 1.0$$

$$\text{and } \frac{1}{\eta_2}\left(\frac{0}{40} + \frac{2}{5} + \frac{4}{40}\right) = 1 \text{ gives } \eta_2 = 0.5.$$

The maximum over the candidate slowdown factors is $\eta_1 = 1.0$, and we assign task τ_1 a slowdown of $\eta_1 = 1.0$. This completes the first iteration. In the next iteration, we compute the candidate slowdown for task τ_2 considering the assigned slowdown of task τ_1

$$\frac{1}{1.0}\left(\frac{2}{5}\right) + \frac{1}{\eta_2}\left(\frac{0}{8} + \frac{4}{40}\right) = 1 \text{ gives } \eta_2 = 0.167.$$

Task τ_2 being the only unassigned task has a slowdown factor set to $\eta_2 = 0.167$. Thus, the slowdown factors for the tasks are $\eta_1 = 1.0$ and $\eta_2 = 0.167$.

2) RM Scheduling: Under the RM scheduling policy, we compute slowdown factors corresponding to each scheduling point of a task as given by (7). Task τ_1 has only one scheduling point $S_{11} = 5$ and task τ_2 has eight scheduling points $S_2 = \{S_{2k} = 5k | k = 1, \ldots, 8\}$. The candidate slowdown factor for task τ_1 is determined by $S_{11} = 5$, and its computation is given as

$$\frac{1}{\eta_{11}}\left(3 + 2 \cdot \left\lceil \frac{5}{5} \right\rceil\right) = 5 \text{ gives } \eta_1 = \eta_{11} = 1.0.$$

The candidate slowdown factor for task τ_2 is the minimum of the slowdown factors corresponding to the eight scheduling points. We compute the slowdown factor for each scheduling point, and the computation for two points $S_{21} = 5$ and $S_{28} = 40$ is given as

$$\frac{1}{\eta_{21}}\left(0 + 2 \cdot \left\lceil \frac{5}{5} \right\rceil + 4 \cdot \left\lceil \frac{5}{40} \right\rceil\right) = 5 \text{ gives } \eta_{21} = 1.2$$

$$\frac{1}{\eta_{28}}\left(0 + 2 \cdot \left\lceil \frac{40}{5} \right\rceil + 4 \cdot \left\lceil \frac{40}{40} \right\rceil\right) = 40 \text{ gives } \eta_{28} = 0.5.$$

The slowdown factor for each scheduling point decreases as the scheduling point increases (for this particular example). The minimum occurs at the scheduling point $S_{28} = 40$, which results in a slowdown factor of $\eta_2 = \eta_{28} = 0.5$. The maximum

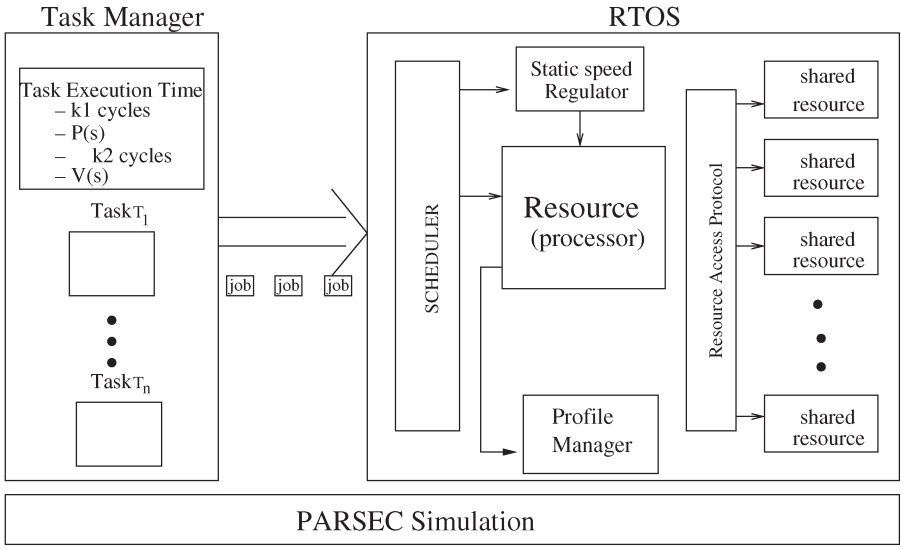

Fig. 4. Generic simulator using PARSEC.

over the candidate slowdown factors for both tasks is $\eta_1 = 1.0$, and we assign task τ_1 a slowdown of $\eta_1 = 1.0$. With the slowdown assigned to task τ_1, we compute the candidate slowdown for task τ_2. For brevity, we only show the computation of slowdown factor corresponding to the two extreme scheduling points S_{21} and S_{28}

$$\frac{2}{1.0} \cdot \left\lceil \frac{5}{5} \right\rceil + \frac{1}{\eta_{21}} \left(0 + 4 \cdot \left\lceil \frac{5}{40} \right\rceil \right) = 5 \text{ gives } \eta_{21} = 1.33$$

$$\frac{2}{1.0} \cdot \left\lceil \frac{40}{5} \right\rceil + \frac{1}{\eta_{28}} \left(0 + 4 \cdot \left\lceil \frac{40}{40} \right\rceil \right) = 40 \text{ gives } \eta_{28} = 0.167.$$

The minimum slowdown factor for the scheduling points of task τ_2 occurs at S_{28}. The candidate slowdown factor for task τ_2 is $\eta_2 = \eta_{28} = 0.167$ and τ_2 is assigned a slowdown of $\eta_2 = 0.167$. Thus, the task slowdown factors computed under RM scheduling are $\eta_1 = 1.0$ and $\eta_2 = 0.167$.

IV. Experimental Results

We have used PARSEC [42], a C-based discrete event simulation language, to implement a simulator with the scheduling policies and the slowdown algorithms. The simulator block diagram is shown in Fig. 4. It consists of two main entities, namely, the task manager and the real time operating system (RTOS). The task manager generates jobs for each task periodically and sends it to the RTOS entity. The RTOS is the crucial entity, and all the scheduling algorithms are part of this entity. It schedules the jobs on the processor and controls the processor speed through the speed regulator. The jobs access the shared resource through the resource access protocol. The profile manager profiles the energy consumed by each task and calculates the total energy consumption of the system.

The dynamic power consumption P for complementary metal oxide semiconductor (CMOS) circuits [43] depends on the operating voltage and frequency of the processor and is given by

$$P = C_{\text{eff}} V_{\text{dd}}^2 f \tag{11}$$

where C_{eff} is the effective switching capacitance, V_{dd} is the supply voltage, and f is the operating frequency. Note that the transistor gate delay (and, hence, frequency) depends on the voltage, and a decrease in voltage is accompanied by a decrease in the processor frequency. The relationship between gate delay t_{inv}, which is inversely proportional to the operating frequency (f), and voltage is given by

$$t_{\text{inv}} = \frac{k V_{\text{dd}}}{(V_{\text{dd}} - V_{\text{th}})^{\alpha}} \tag{12}$$

where V_{th} is the threshold voltage and α is a technology-dependent parameter. Considering the voltage levels in current embedded processors, the operating voltage range for the processor is between 0.6 and 1.8 V. We have normalized the operating speed and support discrete slowdown factors in steps of 0.05 in the normalized range.

We compare the processor energy consumption of two techniques based on the above power model.

1) Uniform slowdown with frequency inheritance (USFI): The algorithm computes a uniform slowdown factor for each task, as presented in Section III-B. Frequency inheritance is part of the resource access protocol, where a blocking task inherits the maximum slowdown factor of the blocked tasks, and, hence, the name USFI.
2) DS algorithm: The DS algorithm is proposed by Zhang and Chanson [34]. A brief explanation of the algorithm is given in the introductory part, and the details are present in [34]. For correctness, the critical sections are nonpreemptive under the DS algorithm.

We also transform the task set to an independent task set and use slowdown algorithms for independent tasks to compute task

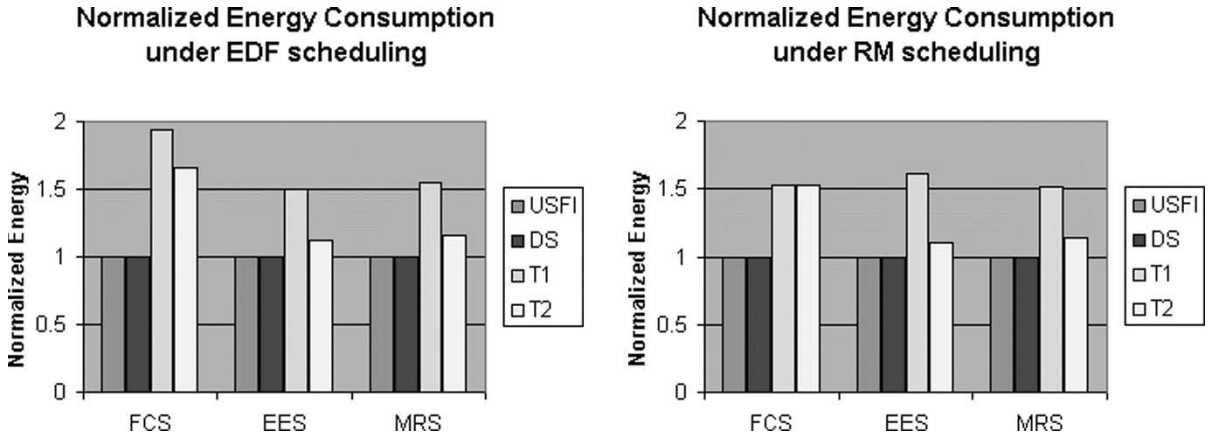

Fig. 5. Energy consumption normalized to USFI under the EDF and RM scheduling policies.

slowdown factor [9], [10]. With tasks having different slowdown factors, frequency inheritance is important in bounding the maximum blocking time and is part of the resource access protocol. The two transformations are explained as follows:

1) Transformation I (T1): For each task τ_i, the execution time C_i is increased by its blocking time B_i to result in a transformed task set $\Gamma' = \{\tau'_1, \ldots, \tau'_n\}$, where each task $\tau'_i = \{T_i, D_i, (C_i + B_i)\}$. The slowdown factors for the transformed task set with frequency inheritance guarantees all task deadlines.
2) Transformation II (T2): We add a new task called the blocking task $\tau_b = \{T_b, T_b, C_b\}$ in the system, where $C_b = \max_i(B_i)$ and $T_b = \min_i(T_i)$. Task τ_b has the smallest deadline, and its computation time is accounted for in the feasibility analysis of all tasks, thereby accounting for the blocking time. Frequency inheritance is required to bound the blocking time and is part of the resource access protocol. All task deadlines are ensured under this transformation.

To manage the resource access, we use the DPCP [39] under EDF scheduling policy and the PCP [31] under fixed priority scheduling. The maximum blocking time for each task is computed based on the protocol (nonpreemptive critical sections under the DS algorithm). We have performed experiments on real-life task sets as well as randomly generated tasks. The slowdown factors are computed using the different algorithms, and the task set is simulated for a time period equal to the hyperperiod of the task set. Each task is assumed to execute up to its WCET.

A. Real-Life Applications

We have used three applications (task sets) given in the Prototyping Environment for Embedded Real Time Systems [44] (PERTS) software. The applications are flight control system (FCS), end to end scheduling (EES), and multiple resource scheduling (MRS). A task set with multiple resources is converted to an equivalent task set by scaling the execution period. Each application has shared resources that are accessed in a mutually exclusive manner. Fig. 5 shows the energy consumption of each algorithm normalized to the energy consumption of the USFI algorithm, under both EDF and RM scheduling policies. It is seen in the graph that the energy consumption follows a similar trend for both EDF and RM scheduling.

For the examples considered, it is seen that the USFI and DS have the same energy consumption. The blocking time of tasks is not significant and even if the slowdown factors are computed assuming the tasks are independent, the feasibility of the task set is not compromised. In such a case, both algorithms compute the same slowdown factors and have the same energy consumption. Transformations T1 and T2 have higher energy consumption compared to USFI. Transformation T1 increases the task execution time by the task blocking time, thereby increasing the utilization of the transformed task set. A higher utilization results in a higher slowdown and, hence, higher energy consumption. T1 has the maximum energy consumption and consumes, on average, 50% more energy than USFI. The transformation T2 has a better performance than T1 and the energy consumption of T2 is relatively closer to that of USFI. Transformation T2 adds a blocking task with a workload of the maximum blocking time over all tasks. The blocking task has the minimum period, and it contributes to a considerable increase in the utilization, thereby leading to higher slowdown factors and more energy consumption. Note that if a transformation results in infeasible slowdown factors, we do not perform slowdown and the tasks are executed at the maximum processor speed ($\eta_i = 1$ for all tasks).

B. Randomly Generated Task Sets

We performed experiments on randomly generated task sets to compare the performance of the DS and USFI algorithms. Similar to the work in [34], we used a mixed workload with task periods belonging to one of the three period ranges [2000, 5000], [500, 2000], and [90, 200]. The WCET for the three ranges is [10, 500], [10, 100], and [10, 20], respectively. Task sets comprise of 10–15 tasks, uniformly distributed in these categories, with the period and WCET of a task randomly selected within the corresponding ranges. The number of semaphores (within 0–2) and the position of the critical sections within each task execution were selected randomly. The length of the critical sections were chosen to be CSperc $\times$ WCET, where

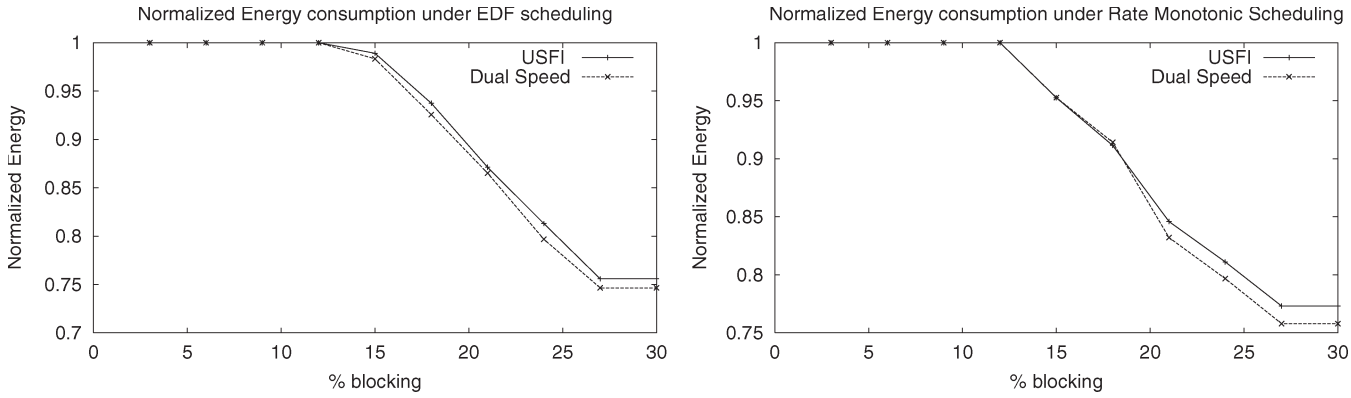

Fig. 6. Energy consumption for random tasks normalized to the HS algorithm.

CSperc is the size of the critical section as a percentage of the WCET. We vary CSperc up to 30% of the WCET in steps of 3%. Task sets are generated with a utilization between 50% and 75% at maximum speed.

Fig. 6 shows the energy consumption of DS and USFI algorithms under both EDF and RM scheduling policies. The energy consumption is normalized to the high speed (HS) [34] algorithm, where the tasks execute at the high speed H of the DS algorithm. From the figure, it is seen that the DS and USFI algorithms follow a similar curve with an increase in blocking. With the blocking time less than 12%, the task slowdown factors are not influenced by blocking time, and the energy consumption is the same as that of the HS algorithm. As the blocking increases further, both DS and USFI compute better slowdown factors and save more energy (up to 30%) compared to the HS. However, it is seen that the DS algorithm consumes slightly less energy than the proposed USFI algorithm. Overall, the DS algorithm performs better than USFI with 1%–4% more energy savings than the USFI algorithm under EDF scheduling. Under the RM scheduling policy, the gains are negligible up to 18% blocking, with 2%–4% energy gains with further increase in blocking.

To explain the differences in the energy savings between the DS and USFI algorithm, we elaborate the differences in the two algorithms. The DS algorithm is based on the computation of two operating speeds, a speed H considering the worst case blocking times for tasks and a speed L assuming tasks to be independent. Conceptually, the two speeds represent two modes of operation referred to as H_{mode} and L_{mode} corresponding to the speeds H and L, respectively. The mode transitions are described by the DS algorithm, and the execution time in H_{mode} and L_{mode} is determined dynamically based on task blocking. When $H > L$, executing tasks at speed L reduces the energy consumption. Comparing the USFI algorithm to the DS algorithm, the system always executes in H_{mode} under the USFI algorithm, where all task slowdown factors are computed considering the worst case task blocking times. As opposed to DS algorithm where all tasks have a constant slowdown (speed H) in H_{mode}, tasks can have different slowdown factors under the USFI algorithm. The slowdown factors computed by the USFI algorithm are less than or equal to the speed H under the DS algorithm. Some tasks have a slowdown factor of $\eta_i = H$, while the remaining tasks have slowdown factors smaller than H, which reduces the energy consumption of the system.

The factors that lead to the difference in energy gains can be explained by comparing the energy consumption during the two modes of execution (H_{mode} and L_{mode}) in the DS algorithm to the analogous counterparts of the USFI algorithm. Note that the USFI algorithm does not have two modes of execution, but all tasks are always executed at the slowdown factors computed by Algorithm 1.

1) Execution in L_{mode}: When the system is executing in L_{mode} under the DS algorithm, all tasks are executed at speed L. The same tasks if executed under the USFI algorithm would execute at speed $\eta_i \leq H$, where some tasks have a slowdown factor, $\eta_i = H$ and other tasks have a slowdown factor $\eta_i < H$ (potentially η_i can be less than L as well). Even when some tasks have a slowdown factor less than L, it is important to note that a uniform slowdown of L for all tasks is more energy efficient than some tasks having higher slowdown and some having a lower slowdown factor [45]. Under the EDF scheduling policy, a uniform slowdown factor of L for the task set is known to be the optimal slowdown (assuming tasks are independent). Thus, the DS algorithm results in significant energy savings when executing the system at speed L, compared to the counterpart task executions under the USFI algorithm.
2) Execution in H_{mode}: When the system is executing in H_{mode} under the DS algorithm, all tasks are executed at speed H. Whereas under the USFI algorithm, the same tasks would be executed at a slowdown factor of $\eta_i \leq H$ (even after frequency inheritance rules). Since some tasks are executed at lower speeds than H, this results in reduced energy consumption compared to DS algorithm. Thus, the USFI algorithm consumes less energy than the DS algorithm for the duration when the system is in H_{mode} under the DS algorithm.

Thus, L_{mode} is beneficial for the DS algorithm, whereas the USFI algorithm performs better when compared to H_{mode} of the DS algorithm. Thus, the effectiveness of the two algorithms depends on the duration of the execution time in the two modes. Fig. 7(a) compares the execution time of the DS algorithm in the two modes (for the case of EDF scheduling). The y-axis

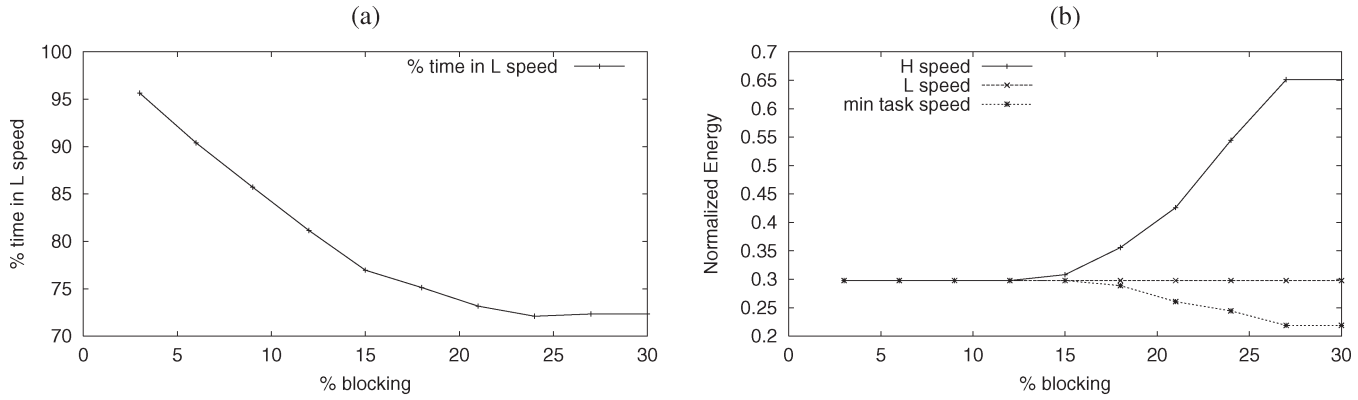

Fig. 7. (a) Percentage time spent by the DS algorithm in L_{mode} and H_{mode}. (b) Energy consumption at speeds H, L, and $\eta_{\min}$.

shows the percentage execution time in L_{mode} as blocking time is varied. It is seen in the figure that the duration of L_{mode} is significantly greater (greater than 78% of the total time) than that of H_{mode}. Since the DS algorithm results in higher energy savings than USFI when executing in L_{mode}, the energy savings of the DS algorithm are higher than that of USFI.

In Fig. 7(a), it is seen that an increase in the blocking percentage decreases the time duration of L_{mode} (time duration in H_{mode} increases). This is due to the fact that an increase in the blocking time (critical sections) increases the chances of a task being blocked, which results in more transitions to H_{mode}. Thus, it is expected that the difference in the energy savings between the two algorithms should decrease as blocking increases. However, the difference in the energy consumption of DS and USFI algorithms is seen to increase as blocking increases (Fig. 6). Note that the difference in the energy consumption in the two modes also plays an important role. Fig. 7(b) compares the normalized energy consumption (normalized to full speed) at speeds H and L. Speed L is independent of the blocking and depends only on the task set utilization. Higher blocking time results in an increase in the speed H, and, hence, we see an increase in the energy consumption at speed H. We also analyze the task slowdown factors computed by the USFI algorithm. Increase in the speed H with higher blocking results in assigning a higher slowdown factor for some tasks $(\eta_i = H)$, and this gives a chance to lower the slowdown factors for other tasks. While tasks can potentially have different slowdown factors, it was seen in our experiments that the task are partitioned into two sets based on the slowdown factors, namely: 1) tasks with a speed $\eta_{\max} = H$; and 2) tasks with a speed $\eta_{\min}$. The energy consumption at a slowdown factor of $\eta_{\min}$ is also shown in Fig. 7(b) (referred as min task speed in the figure). Note that the increase in the energy consumption at speed H, compared to that at speed L, is much greater than the decrease in the energy consumption at speed $\eta_{\min}$. The difference in the energy consumption between speeds H and L increases the energy efficiency of task execution in L_{mode} (of DS algorithm) compared to that under the USFI algorithm. With a higher energy efficiency of L_{mode} with increased blocking, even though the duration of L_{mode} decreases, the total energy savings of the DS algorithm increase. Thus, the difference in energy consumption between DS and USFI increases with greater blocking.

Though the DS scheme performs slightly better than USFI, it is noteworthy that the correctness (meeting all deadlines) of the DS algorithm is based on nonpreemptive critical sections (an example is presented in Section II-D). On the other hand, the USFI algorithm works with task synchronization. The USFI algorithm does not have two modes of operation similar to the DS algorithm, and the task slowdown factors are computed, taking into account the worst case blocking time encountered by each task. Thus, the USFI algorithm guarantees task feasibility under task synchronization. Furthermore, frequency inheritance can be easily augmented to known protocols. Priority inheritance is an integral part of most protocols, and frequency inheritance is a generalization of this idea. Thus, USFI is a simple extension to known protocols and can work with any resource access protocol.

V. Conclusion and Future Work

In this paper, we present algorithms to compute static slowdown factor for a periodic task set. We take into consideration the effect of blocking that arises due to task synchronization. We study static and dynamic priority scheduling policies and have considered the RM and EDF scheduling, respectively. The algorithms have a similar time complexity as their counterpart slowdown algorithms for independent tasks. Computing slowdown factors under EDF scheduling has polynomial time complexity, and computing the slowdown factors under RM scheduling has pseudopolynomial time complexity. Experimental results show that the techniques are energy efficient, and the algorithms can be easily implemented in an RTOS. This will have a great impact on the energy utilization of portable and battery-operated devices.

We plan to further exploit the static and dynamic slack in the system to make the system more energy efficient. As future work, we plan to compute slowdown factors considering the effects of slowdown on the energy consumption of all components in the system.

References

[1] J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice-Hall, 2000.
[2] G. C. Buttazzo, *Hard Real-Time Computing Systems*. Boston, MA: Kluwer, 1995.

[3] F. Yao, A. J. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Symp. Foundations Computer Science*, Milwaukee, WI, 1995, pp. 374–382.
[4] W. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," in *Proc. Design Automation Conf.*, Anaheim, CA, 2003, pp. 125–130.
[5] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc. Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 828–833.
[6] ——, "Minimum energy fixed-priority scheduling for variable voltage processors," in *Proc. Design Automation and Test Europe*, Paris, France, Mar. 2002, pp. 782–787.
[7] H. Yun and J. Kim, "On energy-optimal voltage scheduling for fixed-priority hard real-time systems," *Trans. Embed. Comput. Syst.*, vol. 2, no. 3, pp. 393–430, Aug. 2003.
[8] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proc. Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 2000, pp. 365–368.
[9] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," in *Proc. Int. Symp. Low Power Electronics and Design*, Huntington Beach, CA, Aug. 2001, pp. 46–51.
[10] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. IEEE Real-Time Systems Symp.*, London, U.K., Dec. 2001, pp. 95–105.
[11] ——, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," in *Proc. EuroMicro Conf. Real-Time Systems*, Delft, The Netherlands, Jun. 2001, pp. 225–232.
[12] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th Symp. Operating Systems Principles*, Banff, Canada, 2001, pp. 89–102.
[13] W. Kim, J. Kim, and S. L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *Proc. Design Automation and Test Europe*, Paris, France, Mar. 2002, pp. 788–794.
[14] P. Mejia-Alvarez, E. Levner, and D. Mosse, "Adaptive scheduling server for power-aware real-time tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 2, no. 4, pp. 226–241, Nov. 2003.
[15] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," in *Proc. Asia South Pacific Design Automation Conf.*, Yokohama, Japan, 2004, pp. 651–656.
[16] C. Rusu, R. Melhem, and D. Mosse, "Maximizing rewards for real-time applications with energy constraints," *ACM Trans. Embed. Comput. Syst.*, vol. 2, no. 4, pp. 537–559, Nov. 2003.
[17] T. A. AlEnawy and H. Aydin, "Energy-constrained performance optimizations for real-time operating systems," in *Workshop Compilers and Operating System Low Power*, New Orleans, LA, Sep. 2003, pp. 11–20.
[18] J. Luo and N. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," in *Proc. Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 2000, pp. 357–364.
[19] J. Luo, N. Jha, and L. S. Peh, "Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems," in *Proc. Design Automation and Test Europe*, Munich, Germany, Mar. 2003, pp. 11150–11151.
[20] L. Yan, J. Luo, and N. K. Jha, "Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," in *Proc. Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 2003, pp. 30–38.
[21] F. Gruian and K. Kuchcinski, "LEneS: Task scheduling for low-energy systems using variable supply voltage processors," in *Proc. Asia South Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2001, pp. 449–455.
[22] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. Design Automation Conf.*, New Orleans, LA, 2002, pp. 183–188.
[23] L. Leung, C. Tsui, and W. Ki, "Minimizing energy consumption of multiple-processor-core systems with simultaneous task allocation scheduling and voltage assignment," in *Proc. Asia South Pacific Design Automation Conf.*, Yokohama, Japan, 2004, pp. 645–650.
[24] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," in *Proc. IEEE Real-Time Systems Symp.*, London, U.K., Dec. 2001, pp. 84–94.
[25] D. Zhu, N. AbouGhazaleh, D. Mosse, and R. Melhem, "Power aware scheduling for and/or graphs in multi-processor real-time systems," in *Proc. Int. Conf. Parallel Processing*, Vancouver, Canada, 2002, pp. 593–601.
[26] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *Proc. Int. Parallel and Distributed Processing Symp.*, Nice, France, 2003, p. 21.
[27] A. K. Mok, "Fundamental design problems of distributed systems for hard real-time environment," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, 1983.
[28] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of Np-Completeness*. San Francisco, CA: Freeman, 1979.
[29] J. A. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo, "Implications of classical scheduling results for real-time systems," *IEEE Trans. Comput.*, vol. 28, no. 6, pp. 16–25, Jun. 1994.
[30] T. P. Baker, "Stack-based scheduling of real-time processes," *J. Real-Time Syst.*, vol. 3, no. 1, pp. 67–99, Mar. 1991.
[31] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.
[32] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," in *Proc. Int. Conf. Compilers, Architecture and Synthesis Embedded Systems*, Grenoble, France, Oct. 2002, pp. 164–169.
[33] ——, "Energy aware edf scheduling with task synchronization for embedded real time operating systems," in *Workshop Compilers and Operating System Low Power*, Charlottesville, VA, Sep. 2002, pp. 7.1–7.6.
[34] F. Zhang and S. T. Chanson, "Processor voltage scheduling for real-time tasks with non-preemptible sections," in *Proc. IEEE Real-Time Systems Symp.*, Austin, TX, Dec. 2002, pp. 235–245.
[35] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. New York: Wiley, 2001.
[36] Intel StrongARM Processor, *Intel Inc.*, [Online]. Available: http://www.intel.com/design/strong/specupdt/278259.htm
[37] Intel XScale Processor, *Intel Inc.*, [Online]. Available: http://developer.intel.com/design/intelxscale/xscale_datasheet4.htm
[38] Transmeta Crusoe Processor, *Transmeta Inc.*, [Online]. Available: http://www.transmeta.com/crusoe/specs/html
[39] M. Chen and K. Lin, "Dynamic priority ceilings: A concurrency control protocol for real-time systems," *Real Time Syst. J.*, vol. 2, no. 1, pp. 325–346, Nov. 1990.
[40] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[41] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour," in *Proc. IEEE Real-Time Systems Symp.*, Santa Monica, CA, Dec. 1989, pp. 166–171.
[42] Parallel Computing Laboratory, *PARSEC: A C-Based Simulation Language*, Los Angeles: Univ. California, [Online]. Available: http://pcl.cs.ucla.edu/projects/parsec
[43] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Reading, MA: Addison-Wesley, 1993.
[44] Real Time Systems Laboratory, *Prototyping Environment for Real-Time Systems (PERTS)*, Urbana-Champaign: University of Illinois at Urbana Champaign (UIUC), [Online]. Available: http://pertsserver.cs.uiuc.edu/software/
[45] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processor," in *Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1998, pp. 197–202.

Ravindra Jejurikar (S'99) received the bachelor's degree in computer science (systems software) from the Government College of Engineering, Pune, India, in 1997, and the M.Tech. degree in computer science and engineering from the Indian Institute of Technology, Bombay, India, in 1999. He is currently working toward the Ph.D. degree in computer sytems design at the Department of Information and Computer Science, University of California, Irvine.

His research interests include system level power management with an emphasis on low-power scheduling techniques for real-time systems.

Rajesh Gupta (S'83–M'85–F'04) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1984, the M.S. degree in EECS from the University of California, Berkeley, in 1986, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1993.

He is a Professor and a holder of the Qualcomm endowed chair in embedded microsystems in the Department of Computer Science and Engineering, University of California, San Diego. His current research interests are in embedded systems, very large scale integration (VLSI) design, and adaptive system architectures. Earlier, he was in the faculty of the Computer Science departments of the University of California, Irvine, and the University of Illinois, Urbana-Champaign. Prior to that, he worked as a Circuit Designer at Intel Corporation, Santa Clara, CA, on a number of processor design teams. He is the author/coauthor of over 150 articles on various aspects of embedded systems and design automation and three patents on phase-locked loop (PLL) design, data-path synthesis, and system-on-chip modeling.

Dr. Gupta is a recipient of the Chancellors Fellow at the University of California, Irvine, UCI Chancellors Award for excellence in undergraduate research, National Science Foundation CAREER Award, two Departmental Achievement Awards, and a Components Research Team Award at Intel. He is the Editor-in-Chief of the IEEE Design and Test of Computers and serves on the editorial boards of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON MOBILE COMPUTING. He is a Distinguished Lecturer for the ACM/SIGDA and the IEEE CAS Society.