Optimized Slowdown in Real-Time Task Systems

Ravindra Jejurikar
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697
jezz@cecs.uci.edu

Rajesh Gupta
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093
gupta@cs.ucsd.edu

Abstract

Slowdown factors determine the extent of slowdown 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. We address the problem of computing slowdown factors for dynamically scheduled tasks with specified deadlines. We present an algorithm to compute a near optimal constant slowdown factor based on the bisection method. As a further generalization, for the case of tasks with varying power characteristics, we present the computation of near optimal slowdown factors as a solution to convex optimization problem using the ellipsoid method. The algorithms are practically fast and have the same time complexity as the algorithms to compute the feasibility of a task set. Our simulation results show on an average 20% *energy gains over known slowdown techniques using static slowdown factors and* 40% *gains with dynamic slowdown.*

1. Introduction

Power is an important metric for optimization in the design and operation of embedded systems. A processor is central to an embedded system and contributes to a significant portion of the total power consumption of the system. Modern processors have higher operating speeds and processing capacity to meet the increasing computation demands of application. With the increasing speeds, the power consumption of the processor also increases. Though processors are designed to handle large workloads, the peak processing capacity may not be needed for all applications. This observation has led to two primary ways of reducing the power consumption in embedded computing systems: processor *shutdown* and processor *slowdown*. Slowdown using frequency and voltage scaling has been shown to be effective in reducing the processor energy consumption [28, 29, 2].

Recent processors [12, 30] support slowdown, where we can vary the operating frequency and voltage at run-time. The power consumption, P, depends on the operating voltage and frequency of the processor and is given by:

$$P = C_{eff} \cdot V_{dd}^2 \cdot f \tag{1}$$

where C_{eff} is the effective switching capacitance, V_{dd} is the supply voltage and f is the operating frequency. Due to the quadratic relationship between power and voltage, a decrease in the supply voltage decreases the power consumption. However, the transistor gate delay increases with a decrease in voltage, forcing a decrease in the operating frequency. The relationship between the transistor gate delay, t_{inv}, and supply voltage is given by:

$$t_{inv} = \frac{k \cdot V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{2}$$

where V_{dd} is the operating voltage and V_{th} is the threshold voltage, α has a value in the range 1 to 2 and k is a technology constant [31, 25]. Note that a decrease in the supply voltage has a quadratic decrease in the power consumption but only a linear reduction in the operating frequency, thus resulting in lower energy consumption per unit work. The important point to note is that energy savings are achieved at the cost of increased execution time. Energy reduction and meeting deadlines are often contradictory goals and we have to judiciously manage the tradeoff between time and power to achieve our goal of minimizing energy.

Among the earliest works on this problem, Yao *et. al.* [32] presented an off-line algorithm to compute the optimal speed schedule for a set of N jobs. The optimality is based on the EDF scheduling policy and a continuous voltage range. Kwon *et. al.* [17] have extended this work by relaxing the assumption of a continuous voltage range. Offline scheduling using fixed priority tasks has been addressed in [23] [24] and shown to be NP-hard [33]. As opposed to minimizing the energy consumption of a system, Rusu *et. al.* have addressed the problem of maximizing the system value (utility) for a given energy budget [27, 26]. Scheduling of task graphs on multiple processors has also been

addressed. Luo and Jha [20] have considered scheduling of periodic and aperiodic task graphs in a distributed system. Non-preemptive scheduling of a task graph on a multi processor system is considered by Gruian and Kuchcinski [10]. Zhang *et. al.* [35] have given a framework for task scheduling and voltage assignment for dependent tasks on a multi-processor system. They have formulated the voltage scheduling problem as an integer programming problem.

Dynamic voltage scaling techniques for real-time periodic task systems has been the focus of many works, where known feasibility test have been extended to compute static slowdown factors [29, 9]. A generalization of the energy minimization problem, which incorporates individual tasks with different power consumption characteristics, is addressed by Aydin, Melhem and Mossé [1]. Note that the static slowdown factors are computed based on worst case execution time of each task. Dynamic reclamation techniques in [22, 2, 16] result in additional energy savings by reclaiming run-time slack that arises due to variation in task execution time. Recent work, including our own, has addressed extension of slowdown algorithms to handle task synchronization [34, 14] and aperiodic tasks [21]. Furthermore, the need for leakage energy minimization, which is increasingly important in current and future generation CMOS circuits [5], has lead to procrastination scheduling techniques proposed in [18, 15].

DVS for periodic tasks is well researched, however most works are based on the assumption that the relative task deadline is equal to the task period. Based on this assumption, the Earliest Deadline First (EDF) policy is known to be optimal [19, 4], and the system utilization can be used as a slowdown factor [2]. When the deadlines differ from the period, a similar approach implies that the system density [19] or similar feasibility results [6] can be used as a constant slowdown. However, as we show later in this paper, this slowdown is far from optimal and we bridge this gap in our work. We extend previous work by computing slowdown for periodic tasks with (1) task deadlines less than the period and (2) varying power characteristics for the tasks. We propose the *bisection method* and the *ellipsoid method* to compute optimized static slowdown factors. We gain on an average 20% energy savings over the known techniques with static slowdown and 40% savings with dynamic slowdown.

The rest of the paper is organized as follows: Section 2 formulates the problem with motivating examples. This is followed by algorithms to compute energy efficient slowdown factors. We present the bisection method in Sections 3 and the ellipsoid method is explained in Section 4. The experimental results are given in Section 5 and Section 6 concludes the paper with future directions.

2. Preliminaries

In this section, we introduce the necessary notation and formulate the problem. We first describe the system model followed by an example to motivate the problem.

2.1. System Model

A task set of n periodic real time tasks is represented as $\Gamma = \{\tau_1, \ldots, \tau_n\}$. A task τ_i is a 3-tuple $\{T_i, D_i, C_i\}$, 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 WCET for the task at maximum speed. The phase, ϕ_i, of a periodic task τ_i is the release time of the *first instance* of the task. A set of tasks said to be *in phase* if the first instances of each task is released at the same time. A system, where all tasks are in phase with $\phi_i = 0$, is referred to as a *synchronous* task system [3]. The *hyper-period* of the task set, H, is defined as the least common multiple (lcm) of the task periods. The tasks are scheduled on a single processor system based on a preemptive scheduling policy and all tasks are assumed to be independent. A task system is said to be *feasible* if all tasks meet the 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 [19]. The *density* of the system, $\Delta = \sum_{i=1}^{n} C_i/min(T_i, D_i) \leq 1$, is a sufficient feasibility condition under EDF scheduling [19].

Each invocation of the task is called a *job* and the k^{th} invocation of task τ_i is denoted as $\tau_{i,k}$. Each job J_k is represented by a 3-tuple $\{a_k, d_k, e_k\}$ where a_k is its arrival time, $d_k = a_k + D_i$ its absolute deadline and $e_k \leq C_i$ is its execution time at maximum speed. The time interval $[a_k, d_k]$ is referred to as the *job interval* and e_k is the weight of the interval. The *intensity* of an interval $I = [z, z']$, denoted by $g(I)$ is is defined as in [32] : $g(I) = \frac{\sum_k e_k}{z'-z}$, where the sum is over all job intervals J_k with $[a_k, d_k] \subseteq [z, z']$ i.e. all jobs with their intervals lying completely within $[z, z']$. The interval I^* that maximizes $g(I)$ is called the *critical interval* for a given job set J. In this paper, we only compute the intensity of intervals of the form $[0, t]$, which can be efficiently computed. For a synchronous system ($\forall_i : \phi_i = 0$), the intensity of an interval $[0, t]$, with all tasks executed at maximum speed, is given by : $\frac{1}{t}\sum_{i=1}^{n}(\lfloor \frac{t-D_i}{T_i} \rfloor + 1) \cdot C_i$.

2.2. Variable Speed Processors

A wide range of processors like the Intel StrongARM processors [11], Intel XScale [12], Transmeta Crusoe [30] support variable voltage and frequency levels, which can be varied at run-time. A task *slowdown factor* can be viewed as its normalized operating frequency. At a given instance, it is the ratio of the current frequency to the maximum frequency of the processor. Note that the voltage and frequency levels are tightly coupled, and a (frequency, voltage) pair is

associated with each slowdown factor. The important point to note is when we perform a slowdown, we change the frequency along with a proportionate change in voltage. We assume that the frequency can be varied over a discrete range, with f_{min} and f_{max} being the minimum and maximum frequency respectively. We normalize the speed to the maximum speed to have discrete points in the interval $[\eta_{min}, 1]$, where $\eta_{min} = f_{min}/f_{max}$.

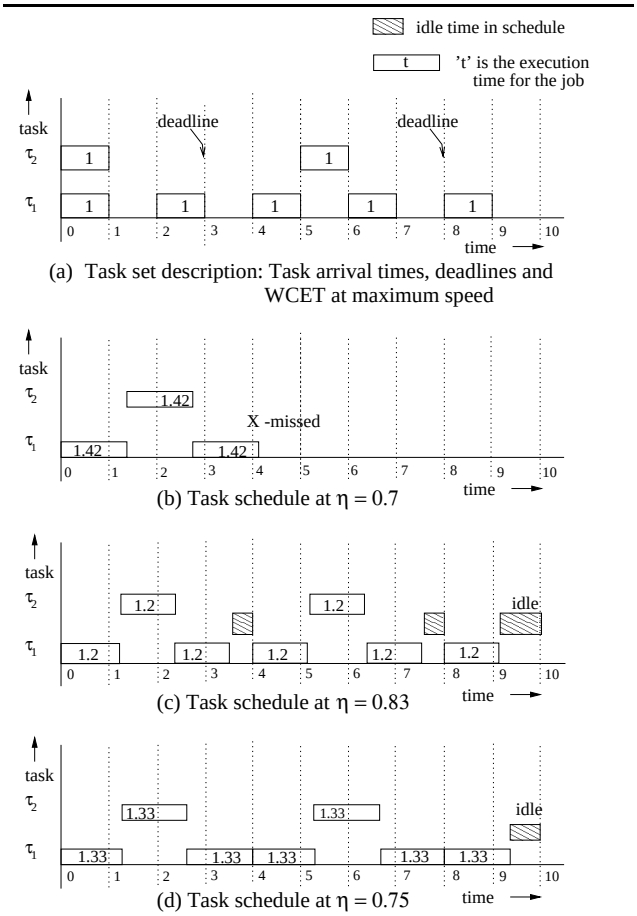
We assume that all invocations (jobs) of a particular task are assigned an equal time budget for execution and this is referred to as a *uniform slowdown*. The assigned time budget can be used for intra-task voltage scaling, however in this work we assume that the budget is utilized by performing a uniform slowdown during the entire task execution. Note that the time budget dictates the extent of slowdown and can be expressed by the task slowdown factor. If all tasks are assigned the same static slowdown factor, it is called a *constant slowdown*. With slowdown, the system utilization increases and is represented by $U_\eta = \sum_{i=1}^{n} \frac{1}{\eta_i} \frac{C_i}{T_i}$, where η_i is the slowdown factor for task τ_i. We assume that the overhead incurred in changing the processor speed is incorporated in the task execution time. Considering static and dynamic slowdown, a speed change occurs only at a context switch. This overhead, similar to the context switch overhead, is constant and can be incorporated in the worst case execution time of a task. We note that the same assumption is made in previous works [1][2].

2.3. Motivating example

Consider a simple real time system with 2 periodic tasks,

$$\tau_1 = \{2, 2, 1\}, \tau_2 = \{5, 3, 1\}$$

Figure 1(a) shows the jobs for each task at their arrival time and their workload at full speed. We have explicitly shown the deadlines when the deadline differs from the period. The task set is feasible under EDF scheduling at full speed. A slowdown equal to the processor utilization $U = (1/2 + 1/5) = 0.7$ is optimal when the relative deadlines are equal to the task period. However, as seen in Figure 1(b), job $\tau_{1,2}$ misses its deadline at a constant slowdown of $\eta = U = 0.7$. Three units of work has to be done in first 4 time units. At a slowdown of 0.7, it requires $3(1/0.7) = 4.285$ time units and a task misses its deadline. Thus, to ensure all task deadlines, the utilization cannot be used as a constant slowdown factor. A constant slowdown equal to the density, $\Delta = (1/2 + 1/3) = 0.83$, keeps the system feasible [19]. (Note that for this example, the feasibility test described in [6] also results in the same slowdown of 0.83.) The schedule at a slowdown of $\eta = \Delta = 0.83$ is shown in Figure 1(c). Note that, this is not the optimal slowdown and the schedule has many idle intervals which can be exploited for further energy savings. A slowdown of $\eta = 0.75$ suffices as shown in Figure 1(c). Note that three units of workload has to be finished within the interval $[0, 4]$ and the intensity of the interval is $3/4 = 0.75$. Thus $\eta = 0.75$ is a lower bound on the constant slowdown and $\eta = 0.75$ is the optimal constant slowdown.



(a) Task set description: Task arrival times, deadlines and WCET at maximum speed

(b) Task schedule at $\eta = 0.7$

(c) Task schedule at $\eta = 0.83$

(d) Task schedule at $\eta = 0.75$

Figure 1. (a) Task arrival times and deadlines (NOT a task schedule). (b) Task schedule at a constant slowdown equal to the utilization, $\eta = 0.70$, and job $\tau_{1,2}$ misses its deadline. (c) Feasible schedule using density as the constant slowdown factor, $\eta = 0.83$, however not optimal. (d) Task schedule at the optimal constant slowdown of $\eta = 0.75$.

A constant slowdown need not be optimal when the task deadline is less than the period. As seen in the Figure 1, there is inherent idle time even at the optimal constant slowdown. This motivates the computation of uniform slowdown factors for the tasks. Furthermore, different tasks can have different power characteristics, and assigning slowdown factors based on task characteristics can be more energy efficient.

3. Constant Static Slowdown

In this section, we propose algorithms to compute the constant slowdown factor under EDF scheduling when the task deadlines can be less than the period ($D_i \leq T_i$). First, we present known feasibility tests for periodic task systems which form the basis of our algorithms.

Theorem 1 *[19] A task set of n periodic tasks, is feasible under EDF scheduling, if the density,* $\Delta = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$.

Theorem 2 *[6] A task set of n periodic tasks, arranged in non-decreasing order of their relative deadlines, is feasible under EDF scheduling if :*

$$\overset{\forall i}{i = 1, \ldots, n} \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{1}{D_i} \sum_{k=1}^{i} \frac{T_k - D_k}{T_k} \cdot C_k \leq 1 \qquad (3)$$

Theorem 1 and 2 are sufficient feasibility conditions. These tests are efficient and run in *linear* time, however not optimal. Theorem 3 by Baruah *et. al.* gives an optimal test when the system utilization is strictly less than 1.

Theorem 3 *[3] A task set is feasible if the intensity of all intervals* $[0,t], t \leq t_{max} = \frac{U}{1-U}\{max(T_i - D_i)\}$*, is less than or equal to 1. Thus the feasibility problem for synchronous systems on one processor is solvable in time* $O(\frac{U}{1-U}\{max(T_i - D_i)\})$.

By Theorem 3, it follows that the constraints for the feasibility of the task set can be specified as :

$$\forall t, t \leq t_{max} : \frac{1}{t} \sum_{i=1}^{n} \left(\lfloor \frac{t - D_i}{T_i} \rfloor + 1 \right) \cdot C_i \leq 1 \qquad (4)$$

The important point to note is that when we consider slowdown, the values of t_{max} depends on the utilization under slowdown, U_η, and is given by $t_{max} = \frac{U_\eta}{1-U_\eta}\{max(T_i - D_i)\}$. Zheng *e. al.* [36] also present a similar result as given by Theorem 3, where they check the intensity of all intervals $[0,t], t \leq t'_{max} = \frac{1}{1-U}\{\sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i)\}$. Note that the t_{max} given in Theorem 3 is just an upper bound of t'_{max}, where each $(T_i - D_i)$ term is approximated by the maximum over all $(T_i - D_i)$ terms.

We extend Theorem 1 and 2 to compute constant slowdown factors as given by Theorem 4 and 5. The proof of the results follows directly from Theorem 1 and 2.

Theorem 4 *Given n independent periodic tasks, the feasibility of the task-set is guaranteed at a constant slowdown of* η*, if* $\frac{1}{\eta} \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$.

Theorem 5 *Given n periodic tasks, arranged in non-decreasing order of their relative deadlines, the task set is feasible at a constant slowdown of* η*, if*

$$\overset{\forall i}{i = 1, \ldots, n} \frac{1}{\eta} \left(\sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{1}{D_i} \sum_{k=1}^{i} \frac{T_k - D_k}{T_k} \cdot C_k \right) \leq 1 \qquad (5)$$

Theorem 5 is a stronger result than Theorem 4 [6], however not optimal. The best slowdown satisfying Theorem 5 can be computed efficiently and we refer to it as the *Devi Test Algorithm (DTA)*, named after the author who proposed the feasibility test. Note that feasibility test given by Theorem 3 is optimal, however it does not compute the optimal slowdown factor. The optimal constant slowdown factor for a periodic task set can be computed as given by Theorem 6.

Theorem 6 *For a synchronous task system,* $\forall_i : \phi_i = 0$*, the maximum intensity over all interval* $[0,t], 0 < t \leq H$*, where H is the hyper-period of the task set, is the optimal constant slowdown factor.*

The proof of the result is present in [13]. It is known that the intensity function can increase only at discrete points represented by the set $S = \{t_{(i,k)} = kT_i + D_i | i = 1, \ldots, n; k \geq 0\}$ [3]. Thus it suffices to check the intensity of the intervals $[0,t]$ with $t \in S$. However, the cardinality of the set S can be exponential in the number of tasks, resulting in a worst case exponential time complexity.

While we propose algorithms for synchronous task systems, note that the computed slowdown factors can be used independent of the task phase. It is known that the maximum intensity interval of a synchronous task set is an upper bound on the maximum intensity interval for the system, irrespective of the task phase [19, 3]. Thus the results in this paper can be applied all periodic task systems.

Corollary 1 *The slowdown factors computed for a synchronous system imply feasibility of the periodic task set, independent of the individual task phase.*

3.1. Bisection Method

We are interested in an efficient algorithm to compute the optimal constant slowdown. The feasibility test given by Theorem 3 is much faster compared to the algorithm given by Theorem 6. Note however, that the feasibility test cannot be directly used to compute slowdown factors. We observe that performing a binary search over the range of slowdown factors can result in a faster algorithm. It is important to note that the time value t_{max} (in Theorem 3) is proportional to $\frac{U_\eta}{1-U_\eta}$, where U_η is the system utilization under slowdown. As we slowdown the system, the utilization of the system increases. As the utilization approaches 1, t_{max} tends to infinity. Thus in the worst case, we may have to check all intervals up to the hyper-period of the task set, which requires worst case exponential time. To avoid the explosion of t_{max}, we impose an additional constraint on the processor utilization, $U_\eta \leq 1 - \varepsilon_u$. Since ε_u is a constant, it bounds t_{max} to $\varepsilon_u^{-1}\{max(T_i - D_i)\}$. We present a pseudo polynomial time algorithm with this additional constraint on utilization.

The algorithm begins with computing upper and lower bounds on the slowdown factor. The upper bound on the

constant slowdown is $\eta_u = min(\Delta, 1)$, where Δ is the system density. The lower bound on the constant slowdown is system utilization at maximum speed. However, at this slowdown, the utilization becomes 1 and t_{max} tends to infinity. To bound t_{max}, we compute the slowdown η_l which bounds the utilization to $1-\varepsilon_u$ and is given by $\eta_l = \frac{U}{1-\varepsilon_u}$. We perform a binary search in the range $[\eta_l, \eta_u]$ to compute the optimal constant slowdown. This technique is called the *bisection method* and is described in Algorithm 1. In each iteration, we test the feasibility of the system at a slowdown of $\eta_m = \frac{\eta_l+\eta_u}{2}$ by checking whether the intensity of all intervals $[0,t]$ is ≤ 1, with $t \leq t_{max} = \frac{U_{\eta_m}}{1-U_{\eta_m}}\{max(T_i - D_i)\}$. The feasibility test is given by Algorithm 2. If the system is feasible, we update the upper bound to η_m, $\eta_u \leftarrow \eta_m$. If the system is infeasible, we update the lower bound to η_m, $\eta_l \leftarrow \eta_m$. This completes one iteration. We compute a new η_m in each iteration of the algorithm. The number of iterations is polynomial in the binary representation of η and we represent this bound by k_η. Thus the loop in line 4 of Algorithm 1 can be bounded by k_η. Since we bound the processor utilization, t_{max} is proportional to $max(D_i - T_i)$ and we have a pseudo polynomial time algorithm.

Algorithm 1 Bisection-Method($\tau_1, \ldots \tau_n$)

1: $\eta_{soln} \leftarrow 1.0$; {Initialization}
2: $\eta_l \leftarrow \frac{U}{1-\varepsilon_u}$; {Lower bound on $\eta : U_{\eta_l}$ is $1-\varepsilon_u$}
3: $\eta_u \leftarrow min(\Delta, 1)$; {Upper bound on η}
4: **for** ($count \leftarrow 1; count < k_\eta; count \leftarrow count + 1$) **do**
5: $\quad \eta_m \leftarrow (\eta_l + \eta_u)/2$;
6: $\quad$ **if** (Feasibility-Test(η_m)) **then**
7: $\quad\quad \eta_u \leftarrow \eta_m$;
8: $\quad\quad \eta_{soln} \leftarrow \eta_m$;
9: $\quad$ **else**
10: $\quad\quad \eta_l \leftarrow \eta_m$;
11: $\quad$ **end if**
12: **end for**
13: return η_{soln};

Algorithm 2 Feasibility-Test(η)

1: $U_\eta = \frac{1}{\eta}\sum_{i=1}^{n} \frac{C_i}{T_i}$; { Utilization at slowdown η}
2: $t_{max} = \frac{U_\eta}{1-U_\eta}$; {$t_{max}$ value at η}
3: **if** ($U_\eta > 1-\varepsilon_u$) **then**
4: $\quad$ return FALSE;
5: **end if**
6: {Feasibility constraints on slowdown}
7: **if** ($\forall_{t<t_{max}} : \frac{1}{\eta_i}\sum_{i=1}^{n}(\lfloor \frac{t-D_i}{T_i} \rfloor + 1) \cdot C_i \leq t$) **then**
8: $\quad$ return TRUE;
9: **else**
10: $\quad$ return FALSE;
11: **end if**

If the utilization at the solution computed by the bisection method is $1-\varepsilon_u$, then we have an approximate solution. Otherwise, we have the optimum solution to the problem in pseudo polynomial time, an exponential improvement over the worst case computation time for the optimal constant slowdown.

4. Uniform Slowdown Factors

In this section, we compute uniform slowdown factors as opposed to a constant slowdown factor. Under *uniform slowdown*, all instances of a task have the same static slowdown factor, however different tasks can have different slowdown factors. Assigning different slowdown factors based on the task characteristics is energy efficient, especially when the task-set is diverse with tasks having different power characteristics [1].

4.1. Optimization Problem

We formulate the energy minimization problem as an optimization problem. Let $\vec{\eta} \in \mathbb{R}^n$ be a vector representing the task slowdown factors, where i^{th} element of the vector represents the slowdown η_i for task τ_i. Let the power consumption of the task τ_i as a function of slowdown be represented by $f_i(\eta)$. The optimization problem is to compute the optimal vector $\eta^* \in \mathbb{R}^n$ such that the system is feasible and the total energy consumption of the system is minimized. The total energy, E, is a function of $\vec{\eta}$ and is given below.

$$E(\vec{\eta}) = \sum_{i=1}^{n} \frac{C_i}{T_i} \frac{f_i(\eta_i)}{\eta_i} \tag{6}$$

The constraints set for the feasibility of the system are described separately for each method.

4.1.1. Devi Test Optimization The best known *polynomial sized* constraints for optimizing the energy function are those given by Theorem 2. Considering slowdown, the constraints are as follows:

$$\overset{\forall i}{i = 1, \ldots, n;} \quad \sum_{k=1}^{i} \frac{1}{T_k}\frac{C_k}{\eta_k} + \frac{1}{D_i}\sum_{k=1}^{i} \frac{T_k - D_k}{T_k} \cdot \frac{C_k}{\eta_k} \leq 1 \tag{7}$$

Since a slowdown factors is the normalized frequency, we have the implicit constraint $\eta_{min} \leq \eta_i \leq 1$. The above constraints form a sufficient feasibility test. The result follows directly from Theorem 2. We refer to the optimization problem under the constraints given by Equation 7 as the *Devi Test Optimization (DTO)* method.

4.1.2. Constraints for Optimal Solution The constraints in Equation 7 are linear in the number of tasks, however the constraints are not optimal. To compute an optimal solution, we consider the intensity constraints given by Theorem 3. The constraint C^t given by Equation 8 specifies that

the intensity of interval $[0,t]$ under slowdown be less than or equal to 1.

$$C^t : \frac{1}{t}\sum_{i=1}^{n}(\lfloor\frac{t-D_i}{T_i}\rfloor+1)\cdot\frac{C_i}{\eta_i} \leq 1 \quad (8)$$

The intensity of an interval depends on the number of task instances in the interval. The number of instances of task τ_i which contribute to the intensity of the interval $[0,t]$ is given by $\sigma_i(t) = (\lfloor\frac{t-D_i}{T_i}\rfloor+1)$ and the execution time of each instance is C_i/η_i. Equation 8 must be true for all t, $0 < t \leq H$, where H is the hyper-period of the task set. Note that the constraint set contains only the constraints C^t, where $t \in S = \{t_{(i,k)} = kT_i + D_i | i = 1,\ldots,n; k \geq 0\}$. In addition, we also have the implicit constraints: $\eta_{min} \leq \eta_i \leq 1$.

4.1.3. Constraints with bounded processor utilization To compute the optimal solution, the constraint set contains all C^t; $t \leq H$, resulting in exponential number of constraints. Similar to the bisection method, we constraint the system utilization by $U_\eta \leq 1-\varepsilon_u$, to reduce the constraint set. This reduces the constraint set to include all constraints C^t; $t \leq \varepsilon_u^{-1}\{max(T_i - D_i)\}$, resulting in pseudo polynomial number of constraints. However, solving a system with pseudo polynomial constraints is computationally intensive and it requires time, two orders magnitude larger than the proposed ellipsoid method [13].

Note that, to check the feasibility of a slowdown vector $\vec{\eta}$, we need not check all the constraints in the system. It suffices to check the constraints given by Theorem 3, where the number of constraints depend on the utilization U_η. The number of constraints *vary* with each vector and checking the minimal required constraints for a given vector can lead to faster algorithms. In the ellipsoid method, the constraints are not specified explicitly and it is well suited for problems of this nature.

4.2. Ellipsoid Method (Algorithm)

In this section, we present a high level description of the ellipsoid algorithm [7]. We begin with a description of the terms used in the ellipsoid method, and then apply it to our energy minimization problem. The exact definitions are given in [8].

4.2.1. Background A *weak optimization problem* is to compute a solution that is close-to-optimum under specified performance guarantees. An *oracle-polynomial time algorithm* is an algorithm that calls an oracle algorithm a polynomial number of times. A *weak separation oracle* is an algorithm that decides if a vector is in the feasible space and if not, it generates a hyper-plane that approximately separates the feasible space from the vector with some specified performance guarantees. We state the theorem which solves the weak optimization problem given that we can solve the weak separation problem.

Theorem 7 *[8] There exists an oracle-polynomial time algorithm that solves the weak optimization problem for every circumscribed convex body $(K;n,R)$ given by a weak separation oracle. ($(K;n,R)$ represents a convex body $K \in \mathbb{R}^n$ and is contained in a sphere with center as the origin and radius R.)*

The algorithm that computes the weak optimum solution is the ellipsoid method. The ellipsoid method applies to problems where the feasible space and the optimization function are convex, and the gradient of the optimization function can be computed. We show our problem formulation satisfies the above properties for all convex differential power functions, $f_i(\cdot)$ [13]. The main result of the ellipsoid method is as follows. If we have a subroutine (called the separation oracle) that checks the feasibility of a vector $\vec{\eta}$ and generates a separating hyper-plane if the vector is not feasible, then we can compute a close to optimum solution in a polynomial number of calls to the separation oracle. The detailed theorems and proofs stating this result are present in [8]. By Theorem 7, we can compute a weak optimum solution in polynomial number of calls to the separation algorithm (separation oracle) using the ellipsoid method.

4.2.2. Separation Oracle (algorithm) We present the separation oracle used by the ellipsoid method to compute slowdown factors. The separation oracle is based on the feasibility test given by Theorem 3. Let $\vec{\eta}$ represent the task slowdown factors. To bound the running time of the feasibility test, we impose an additional constraint on the utilization, $U_\eta \leq 1-\varepsilon_u$. The number of constraints to check is proportional to $\frac{U_\eta}{1-U_\eta}$. If a constraint C^{t_i} is violated for vector u, then the hyper-plane $\nabla C^{t_i}(\vec{\eta})(\vec{\eta}-u)$ satisfy the property of the separating hyper-plane [8]. C^t is differentiable and the separating hyper-plane is computed by evaluating the derivative of C^t at vector u. This gives a pseudo polynomial time separation oracle.

4.2.3. Geometric Interpretation of Ellipsoid Method We give a geometric interpretation of ellipsoid method. We start with an ellipsoid (convex body) containing the feasible space and the given optimization function. We check the feasibility of the center of the ellipsoid. If it is not feasible, the separation oracle returns a separating hyper-plane, H, that cuts the ellipsoid into two halves (feasibility cut). If the center of the ellipsoid is feasible, we compute the gradient of the optimization function at the center. This gradient hyper-plane splits the ellipsoid into two halves (optimality cut). In both cases, we can identify the non-optimal half

by the property of convex functions. We include the optimal half into a new ellipsoid. In each iteration, the volume of the ellipsoid decreases by a fixed ratio inversely proportional to n. After a polynomial number of steps, the volume of the ellipsoid is very small (ε_{volume}) and we have a close-to-optimal solution. The ellipsoid method is explained in Algorithm 3. NewEllipsoid() constructs the new ellipsoid and its center as shown in line 12. The details are given in [7].

Algorithm 3 Ellipsoid Method($\tau_1, \ldots \tau_n$)

1: Set all elements of η_{soln} to 1.0; {Initialization}
2: $K \leftarrow$ sphere around origin to include all feasible slowdown factors: $0 < \eta_i \leq 1$; {Initial Ellipsoid}
3: $\vec{\eta}_c \leftarrow$ origin; {Center of the ellipsoid}
4: **while** ($volume(K) > \varepsilon_{volume}$) **do**
5: **if** (Feasibility-Test ($\vec{\eta}_c$)) **then**
6: $h \leftarrow \nabla E(\vec{\eta})(\vec{\eta} - \vec{\eta}_c)$; {The gradient of the energy function (optimality-cut)}
7: $\eta_{soln} \leftarrow \eta_c$;
8: **else**
9: Let C^x be the violated constraint;
10: $h \leftarrow \nabla C^x(\vec{\eta})(\vec{\eta} - \vec{\eta}_c)$; {The gradient of the constraint C^x (feasibility-cut)}
11: **end if**
12: $(K_{new}, \eta_{new}) \leftarrow NewEllipsoid(K, h)$;
13: $(K, \eta_c) \leftarrow (K_{new}, \eta_{new})$;
14: **end while**
15: return η_{soln};

Algorithm 4 Feasibility-Test($\vec{\eta}$):

$U_\eta = \sum_{i=1}^{n} \frac{1}{\eta_i} \frac{C_i}{T_i}$; {Utilization at slowdown $\vec{\eta}$}
$t_{max} = \frac{U_\eta}{1-U_\eta}$; {$t_{max}$ value at $\vec{\eta}$ }
Constraint Set is :
$C^u : \sum_{i=1}^{n} \frac{1}{\eta_i} \frac{C_i}{T_i} \leq 1 - \varepsilon_u$ {Utilization constraint}
$\forall_{t<t_{max}}; C^t : \frac{1}{t} \sum_{i=1}^{n} (\lfloor \frac{t-D_i}{T_i} \rfloor + 1) \cdot \frac{C_i}{\eta_i} \leq 1$ {Feasibility constraints}
$\forall_{i=1}^{n}; C^i : \eta_{min} \leq \eta_i \leq 1$ {Processor constraints}
if (all constraints are satisfied) **then**
return TRUE;
else
return FALSE;
end if

5. Experimental Results

Simulation experiments were performed to evaluate our proposed techniques. We considered several task sets, each containing 10-20 randomly generated tasks. We note that such randomly generated tasks are a common validation methodology in previous works [29, 2, 1, 6]. Tasks were assigned a random period and WCET in the range [20000,50000] and [100,5000] respectively. We tried to keep the hyper-period low by rounding the periods to a multiple of 1000. To generate task with deadlines smaller than the period, the deadlines were decreased by 0% to 25% of the task period, in steps of 5%.

We use a power model based on the dynamic power consumption of CMOS circuits [25] as given by Equations 1 and 2. We note, however that our algorithm can be applied to more sophisticated power models, particularly as leakage becomes a significant contribution [5]. Recent processors have low operating voltages, and we use an operating voltage range of $0.6V$ and $1.8V$. The threshold voltage is assumed to be $0.36V$ and $\alpha = 1.5$. We have normalized the operating speed and support discrete voltage levels in steps of 0.05 in the normalized range.

We compared the energy consumption for the following techniques presented in the paper:

- Devi Test Algorithm (DTA)
- Bisection Method (BM)
- Devi Test Optimization (DTO)
- Ellipsoid Method (EM)

We set ε_u to a value of 0.01 in both *BM* and *EM*. This results in utilizing up to 99.9% of the processor, while having practical run-times. The computed slowdown factor was mapped to the smallest discrete level greater than or equal to it. Note that DTA and DTO are sufficient feasibility tests and a feasible task set can be declared as infeasible, in which case we execute all tasks at the maximum speed ($\eta = 1$).

5.1. Identical Power Characteristics

The slowdown factor can be efficiently computed by the DTA method, with a run time polynomial in the number of task. However, it is not energy efficient and has maximum energy consumption. The computation of the optimal intensity inspects all intervals up to the hyper-period of the task set which can be very large. In our experiments, in spite of setting task periods as multiples of 1000, the hyper-period interval was too large to evaluate the intensity of all intervals up to the hyper-period. With tasks having identical power characteristics, the *BM* and *EM* have very close energy consumption. Similar is the case for *DTA* and *DTO*. Since, the *BM* computes a near optimal slowdown, the slowdown computed by *BM* is always better than *DTA*.

Figure 2 shows the percentage energy gains of the *BM* over *DTA*, as a function of the utilization at maximum speed

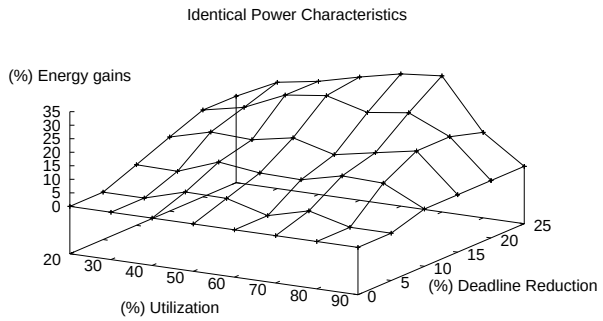


Figure 2. Percentage Energy savings of the Bisection Method (BM) over the Devi Test Algorithm (DTA) as the utilization under maximum speed is varied.

and the deadline reduction. All tasks are assumed to execute up to their WCET. It is seen that the *BM* outperforms *DTA* as the deadline are made stricter. With a decrease in relative task deadlines, the slowdown factor computed by *DTA* shoots up and tasks are executed at higher speed consuming more energy. The *BM* method continues to compute near optimal slowdown factor to result is energy gains. For higher utilization, a small decrease in relative deadline makes the system infeasible under *DTA* and we set $\eta = 1$. However, the slowdown computed by *BM* continues to increase and the gains seem to decrease at utilization of 80% and 90%. At lower utilization (at maximum speed), the density continues to increases with a decrease in deadline and hence there is a steady increase in gains. The gains seem to lower as utilization decreases due to the relation between power and slowdown. An improvement δ in the slowdown factor, results in more gains when the slowdown is higher. For this reason, the gains seem to decrease as the utilization falls below 60%. Thus in all cases, *BM* performs better with a decrease in deadline. We see as much as 35% energy gains over the DTA with the average energy savings being 20%.

5.2. Varying Power Characteristics

Due to the diverse nature of the tasks in a system, tasks can have distinct power characteristics [1]. The problem formulation in Section 4 works for all convex differential power characteristics. However, for experimental results we restrict power characteristics to be *linear* [1], where tasks have a constant power coefficient. We say a task has power coefficient *k* to represent a task with a power consumption *k* times the base case. This is equivalent to tasks having a

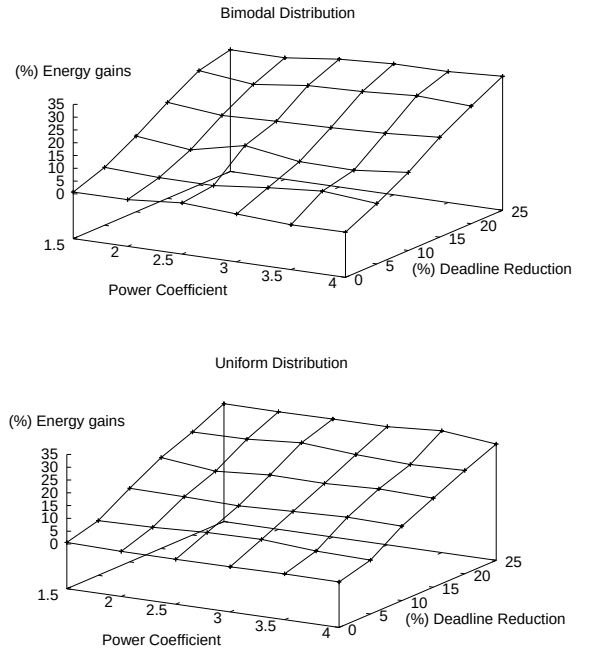


Figure 3. Percentage Energy gains of the Ellipsoid Method (EM) over the Devi Test Optimization method (DTO) for varying task power characteristics.

switching capacitance, C_{eff} in Equation 1, *k* times the base case. The workload for each task is set to its worst case execution time. We consider the following two distributions for the power coefficients:

- **Bimodal Distribution**: where there are two types of tasks in the system, with 50% having a power coefficient of 1 and the others having a power coefficient *k*.
- **Uniform Distribution**, where the power function coefficients of the tasks are uniformly distributed between 1 and *k*. We vary *k* in the range $[1,4]$.

We compare the energy consumption of the EM and DTO for task sets with a utilization between 60%-80% and execution time set to their WCET. Figure 3 shows the percentage gain of EM over DTO. Since both methods compute slowdown factors considering the task power characteristics, the energy gains do not vary with the power coefficient *k* and its distribution. The energy savings depends on the amount of slack that the slowdown method can identify. As the deadline is decreased, DTO identifies less slack and the EM performs better. The EM uses a near optimal feasibility test and utilizes the maximum available slack. Similar gains are seen for the case of uniform and bimodal distribution of power coefficients.

5.3. Dynamic Slowdown

Dynamic reclamation techniques results in further energy savings by reclaiming the run-time slack that arises due to variations in the task execution time. We use the Generic Dynamic Reclaiming Algorithm (*GDRA*) [1] by Aydin *et. al*. The authors have shown that a run-time slack of a higher priority task can be utilized by lower priority jobs, while ensuring all task deadlines. The details are present in [1].

We vary the best case execution time (bcet) of a task from 100% to 10% of its WCET (wcet). Tasks were generated by a Gaussian distribution with mean, $\mu = (wcet + bcet)/2$ and a standard deviation, $\sigma = (wcet - bcet)/6$. We performed experiments on tasks with varying utilization and power characteristics and observed the same trend. As before, we compare *BM* with *DTA* for the case of identical power characteristics. For the case of varying task power characteristics, we compare EM with *DOA*. In all the techniques, we use the *GDRA* technique over the static slowdown factors to compute dynamic slowdown. The gains are shown in Figure 4. As expected, the energy gains increase with the decrease in deadline. It is seen that the average gains steadily increase as the execution time decreases. Thus, identifying the maximum slack through the computation of static slowdown factors helps in dynamic reclamation as well.

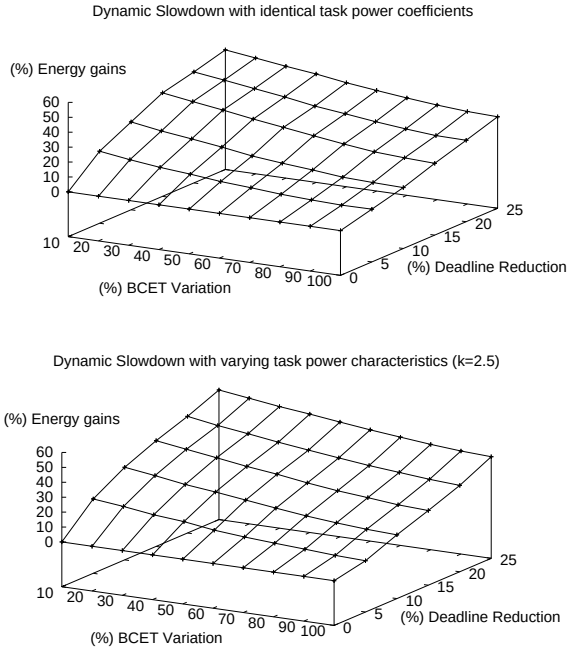


Figure 4. Percentage Energy gains with Dynamic Slack Reclamation scheme. The top graph compares the gains of *BM* over *DTA* with the dynamic reclamation scheme. The gains of *EM* over *DTO* along with dynamic reclamation are shown in the bottom graph.

5.4. Computation Time

The computation times for the various techniques are of different orders of magnitude. In our examples, the hyperperiod was usually too large and computing the optimal intensity was not possible. *BM* computed the solution in orders of milliseconds. It took 1 to 10 seconds to compute the solution using EM. Each iteration of the ellipsoid method requires matrix computations to compute the new ellipsoid and its center, which is computation intensive. However, note that the matrix computations in *EM* require time polynomial in the number of tasks, making our technique scalable. The *DTA* runs in linear time and the computation is negligible. The computation time for DTO is also small, since it has only linear number of constraints. We conducted the experiments on a Sun workstation. Since the computations are performed off-line, we justify a computation time of few milliseconds to seconds for energy gains.

6. Conclusions and Future Work

We have presented algorithms to compute static slowdown factor under EDF scheduling, when the task deadlines are smaller than the task periods. We see that identifying slack by means of static slowdown factors results in energy savings through static and dynamic slowdown. We proposed the bisection method to compute constant static slowdown factors and an algorithm based on ellipsoid method to compute uniform slowdown factors. Experimental results show on an average 20% energy gains over the known slowdown techniques. The average gains extend to 40% with dynamic slowdown. The algorithms have a pseudo polynomial time complexity and are practically fast. The techniques are energy efficient and can be easily implemented in an RTOS. This will have a great impact on the energy utilization of portable and battery operated devices.

In our future work, we plan to extend these techniques to compute optimal discrete task slowdown factors.

Acknowledgments

The authors would like to specially thank George Lueker for explaining the ellipsoid method and it applicability to the problem. We acknowledge support from National Science Foundation (Award CCR-0098335) and from Semiconductor Research Corporation (Contract 2001-HJ-899). We would also like to thank the reviewers and the members of CECS for their comments on the paper.

References

[1] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of EuroMicro Conference on Real-Time Systems*, Jun. 2001.

[2] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2001.

[3] S. K. Baruah, R. R. Howell, and L. E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. In *IEEE Transactions on Computers*, 1991.

[4] G. C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, 1995.

[5] J. A. Butts and G. S. Sohi. A static power model for architects. In *Intl. Symposium on Microarchitecture*, 2000.

[6] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proceedings of EuroMicro Conference on Real-Time Systems*, Jun. 2003.

[7] M. Grotschel, L. Lovasz, and A. Schrijver. Geometric algorithms and combinatorial optimization. In *Combinatorica*, pages 169–97, 1981.

[8] M. Grotschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.

[9] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 46–51, Aug. 2001.

[10] F. Gruian and K. Kuchcinski. LEneS: task scheduling for low-energy systems using variable supply voltage processors. In *Proceedings of the Asia South Pacific Design Automation Conference*, Jan. 2001.

[11] Intel StrongARM Processor. Intel Inc. *(http://www.arm.com/armtech/StrongARM)*.

[12] Intel XScale Processor. Intel Inc. *(http://developer.intel.com/design/intelxscale)*.

[13] R. Jejurikar and R. Gupta. Optimized slowdown in real-time task systems. In *CECS Technical Report #04-10, University of California Irvine*, Apr. 2004.

[14] R. Jejurikar and R. Gupta. Dual mode algorithm for energy aware fixed priority scheduling with task synchronization. In *Workshop on Compilers and Operating System for Low Power*, Sept. 2003.

[15] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, Jun. 2004.

[16] 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 *Proceedings of Design Automation and Test in Europe*, Mar. 2002.

[17] W. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the Design Automation Conference*, pages 125–130, 2003.

[18] Y. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *EcuroMicro Conf. on Real Time Systems*, Jun. 2003.

[19] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[20] J. Luo and N. Jha. Power-conscious joint scheduling of periodic task graphs and a periodic tasks in distributed real-time embedded systems. In *Proceedings of International Conference on Computer Aided Design*, pages 357–364, Nov. 2000.

[21] P. Mejia-Alvarez, E. Levner, and D. Mosse. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 2(4), Nov. 2003.

[22] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of 18th Symposium on Operating Systems Principles*, 2001.

[23] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the Design Automation Conference*, pages 828–833, Jun. 2001.

[24] G. Quan and X. Hu. Minimum energy fixed-priority scheduling for variable voltage processors. In *Proceedings of Design Automation and Test in Europe*, Mar. 2002.

[25] J. M. Rabaey, A. Chandrakasan, and B. Nikolić. *Digital Integrated Circuits*. Printice Hall, 2003.

[26] C. Rusu, R. Melhem, and D. Mosse. Maximizing rewards for real-time applications with energy constraints. In *ACM Transactions on Embedded Computer Systems*, accepted.

[27] C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2002.

[28] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the Design Automation Conference*, Jun. 1999.

[29] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of International Conference on Computer Aided Design*, pages 365–368, Nov. 2000.

[30] Transmeta Crusoe Processor. Transmeta Inc. *(http://www.transmeta.com/technology)*.

[31] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison Wesley, 1993.

[32] F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 374–382, 1995.

[33] H. Yun and J. Kim. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *Trans. on Embedded Computing Sys.*, 2(3):393–430, 2003.

[34] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2002.

[35] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the Design Automation Conference*, 2002.

[36] Q. Zheng and K. G. Shin. On the ability of establishing real-time channels in point-to-point packet-switched networks. *IEEE Transactions on Communications*, 42(2/3/4):1096–1105, Feb/Mar/Apr. 1994.