Computing Static Slowdown Factors under EDF Scheduling when Deadline less than Period

Ravindra Jejurikar Rajesh K. Gupta

Center for Embedded Computer Systems,
Department of Information and Computer Science,
University of California at Irvine,
Irvine, CA 92697
E-mail: {jezz,rgupta}@ics.uci.edu

CECS Technical Report #02-36

December 13, 2002

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. This paper describes computation of slowdown factor for a task set with an underlying dynamic priority scheduler such as Earliest Deadline First (EDF) scheduler. A constant slowdown equal to the processor utilization is optimal for periodic tasks with deadline equal to period. The slowdown factors are non-trivial when the deadlines are not equal to the task period. The density *[20] of the system can be used as a constant slowdown factor, however it is not the optimal slowdown. We propose an algorithm to compute the optimal constant slowdown factor. This algorithm has a worst case exponential time complexity and we present a pseudo polynomial time algorithm to compute the constant slowdown factor that is close to the optimal. Different tasks having different slowdown factors can be more energy efficient. We assume all instances of a task have the same slowdown and call it the* uniform *slowdown factors. We formulate the problem as an optimization problem where the total system energy is to be minimized. We use the ellipsoid method[7] to compute uniform static slowdown factors for the tasks. This technique extends to tasks with varying power characteristics more precisely when the power consumption of a task is a convex [23] function. The simulation results for our test examples show on an average 20% - 30% energy gains over the known slowdown techniques.*

Contents

List of Figures

List of Tables

1 Introduction

Power is one of the important metrics for optimization in the design and operation of embedded systems. There are two primary ways to reduce power consumption in embedded computing systems: processor shutdown and processor slowdown. Slowdown using frequency and voltage scaling is more effective than shutdown in reducing the power consumption. Scaling the frequency and voltage of a processor leads to an increase in the execution time of a job. In real-time systems, we want to minimize energy while adhering to the deadlines of the tasks. Power and deadlines are often contradictory goals and we have to judiciously manage time and power to achieve our goal of minimizing energy. DVS (Dynamic Voltage Scaling) techniques exploit the idle time of the processor to reduce the energy consumption of a system. We deal with computing the voltage schedule for a periodic task set.

In this paper, we focus on the system level power management via computation of static slowdown factors as opposed to dynamic slowdown factors computed at run time. We assume a real-time system where the tasks run periodically in the system and have deadlines. These tasks are to be scheduled on a single processor system based on a preemptive scheduling policy. Our work applies to dynamic priority scheduling schemes and we consider the Earliest Deadline First (EDF) [20] [5] scheduling policy. Our aim is to schedule the given task set and the processor speed such that all tasks meet their deadlines and the energy consumption is minimized. We compute static slowdown factors for the tasks to minimize the total energy consumption of the system.

Most of the earlier work deals with independent task sets. Shin et al. [29] have computed uniform slowdown factors for an independent periodic task set. In this technique, rate monotonic analysis is performed on the task set to compute a constant static slowdown factor for the processor. Gruian [9] observed that performing more iterations gives better slowdown factors for the individual task types. Yao, Demers and Shanker [31] presented an optimal off-line speed schedule for a set of N jobs. The time complexity of their algorithm is $O(N^2)$ and can be reduced to $O(N\ log^2 N)$ by the use of segment trees [25]. The analysis and correctness of the algorithm is based on an underlying EDF scheduler, which is an optimal scheduler [20]. An optimal schedule for tasks with different power consumption characteristics is considered by Aydin, Melhem and Mossé [1]. The same authors [2] prove that the utilization factor is the optimal slowdown when the deadline is equal to the period. Quan and Hu [26] [27] discuss off-line algorithms for the case of fixed priority scheduling. In our earlier work, we address the problem of computing static slowdown factors in the presence of task synchronization. Both Rate Monotonic Scheduling (RMS) [15] and EDF scheduling [14] have been addressed in this context.

Since the worst case execution time (WCET) of a task is not usually reached, there is dynamic slack in the system. Pillai and Shin [24] recalculate the slowdown when a task finishes before its worst case execution time. They use the dynamic slack while meeting the deadlines. Low-power scheduling using slack reclamation heuristic is studied by Aydin et al. [2] and Kim et al. [17].

Scheduling of task graphs on multiple processors has also been considered. Luo and Jha [22] 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. [32] have given a framework for task scheduling and voltage scheduling of dependent tasks on a multi-processor system. They have formulated the voltage scheduling problem as an integer programming problem. They prove the voltage scheduling problem for the continuous voltage case to be polynomial time solvable.

Earlier work on computation of slowdown factors for the tasks deals with independent tasks where

the task deadline is equal to the period. The schedulability tests for independent tasks is given by Liu and Layland [19]. We compute slowdown factors for the case when the task deadline is less than or equal to its period. We present algorithms to compute constant slowdown factors and uniform slowdown factors for the tasks. We use the ellipoid method to compute the soltions close to the optimum solution under good performance guarantees. Our work extends to tasks with varing power characteristics, more precisely all power characteristics that are convex [23] functions. We gain as much as 30% to 35% energy savings over the known techniques.

The rest of the paper is organized as follows: Section 2 formulates the problem with motivating examples. In Section 3 and 4, we present slowdown algorithms to minimize the energy consumption under a dynamic priority scheduling policy such as EDF scheduling. The experimental results are given in Section 5. 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 periodic task set of n periodic real time tasks is represented as $\Gamma = \{\tau_1, ..., \tau_n\}$. A 3-tuple $\tau_i =< T_i, D_i, C_i >$ is used to represent each task τ_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, given it is the only task running in the system. All tasks are assumed to be independent and preemptive.

The tasks are scheduled on a single processor which supports variable frequency and voltage levels. Every frequency level has a power consumption value and is also referred to as power state of the processor. Our aim is to schedule the given task set and the processor speed such that all tasks meet their deadlines and the energy consumption is minimized. The processor speed can be varied to minimize energy usage. 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. We assume the speed of the processor can be varied over a continuous range. If the processor speed is a constant value over the entire time interval, it is called a *constant slowdown*. The execution time of a job is proportional to the processor speed. The goal is to minimize the energy consumption while meeting deadlines. In this paper, we ignore the time and energy overhead incurred in changing the frequency and voltage of the processor.

Each invocation of the task is called a *job* and the k^{th} invocation of task τ_i is denoted as $\tau_{i,k}$. A job J_k is represented by a 3-tuple $< a_k, d_k, e_k >$ where a_k is the arrival time of the job, d_k is the absolute deadline of the job with $d_k > a_k$ and e_k is the maximum number of cycles (WCET) required by the job. The time interval $[a_k, d_k]$ is referred to as the *job interval* and e_k is the weight of the interval. The *hyper-period* of a periodic task set is defined as the least common multiple (lcm) of the periods of all the tasks.

We list the definitions and terms used in the rest of the paper. The *utilization factor* for a task τ_i is defined as $u_i = C_i/T_i$. The processor utilization (U) for a task set is the sum of the utilization factors for each task. $U = \sum_{i=1}^{n} u_i \leq 1$ is a necessary condition for the feasibility of any schedule [20]. Let η_i be the slowdown factor associated with task τ_i. The utilization of a task under slowdown is defined analogously as $U = \sum_{i=1}^{n} \frac{1}{\eta_i} \frac{C_i}{T_i}$. The *density of a task* is defined as the ratio of the execution time C_k to the minimum of

period and deadline, $density(\tau_i) = \frac{C_i}{D_i}$. The *density of the system* is defined as the sum of the density of each task in the system. It is denoted by $\Delta = \sum_{i=1}^{n} density(\tau_i)$. The phase ϕ_i for a periodic task T_i is the release time of the *first instance (job $\tau_{i,1}$)* of the task. A set of tasks are said to be in phase if they have the same phase. Such a system is referred to as a *synchronous* task system. We deal with synchronous systems in this paper.

Assuming all jobs are represented by their job intervals along the time line we define the following.

- **Intensity of an interval**: The intensity of an interval $I = [z, z']$ denoted by $g(I)$ is defined [31] as

$$g(I) = \frac{\sum_k e_k}{z' - z} \tag{1}$$

  where the sum is over all job intervals j_k with $[a_k, b_k] \subseteq [z, z']$ i.e. all jobs with their intervals lying completely within $[z, z']$.
  g(I) is a lower bound on the average processing speed in the interval I by any feasible schedule.

- **Critical Interval**: The interval I^* that maximizes $g(I)$ is called the *critical interval* for the job set J. The set of jobs $J_{I^*} = \{k | [a_k, b_k] \subseteq I^*\}$ is called the *critical job set*.

  Any time interval is represented by an ordered pair (a, b) and it is always true that $b \geq a$.

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. Voltage and frequency levels are tightly coupled. When we change the speed of a processor we change its operating frequency. We proportionately change the voltage to a value which is supported at that operating frequency. The important point to note is when we perform a slowdown we change both the frequency and voltage of the processor. We use the terms slowdown state and power state interchangeably. We assume that the frequency can be varied continuously from f_{min} to the maximum supported frequency f_{max}. We normalize the speed to the maximum speed to have a continuous operating range of $[\eta_{min}, 1]$, where $\eta_{min} = f_{min}/f_{max}$.

2.3 Motivating example

Consider a simple real time system with 2 periodic tasks having the following parameters :

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

The task set is shown in Figure 1(a). The jobs for each task are shown at their arrival time with their WCET at full speed. We have explicitly shown the deadlines where the deadline is different from the period. The jobs are to be scheduled on a single processor by an EDF scheduler. The task set is feasible at full speed under EDF scheduling . For the above example the system density, $\Delta = (1/2 + 1/3) = 0.83$. A constant slowdown of $s = 0.83$ increases the system density to 1. If the system density is less than or equal to 1, the system is feasible. We show that the processor utilization cannot be used as a slowdown factor. The processor utilization U for this task set is $(1/2 + 1/5) = 0.7$. If the processor utilization U is used as a slowdown factor, *job* $J_{1,2}$ misses its deadline. This is shown in Figure 1(b). Three units of work has to be done in first 4 time units. At a slowdown of 0.7, it requires $3/0.7 = 4.285$ time units and

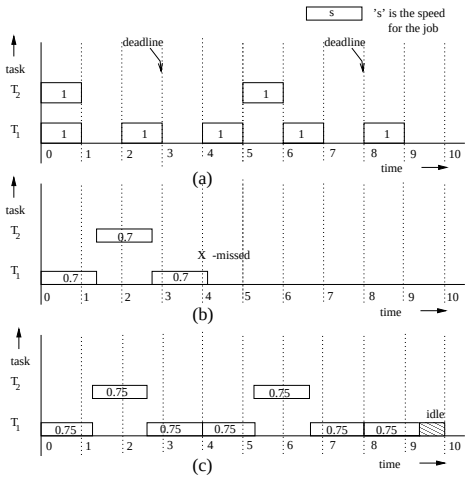

Figure 1. Motivation for Optimal Static slowdown techniques (a) Task arrival times and deadlines. (b) Slowdown $s = 0.70$, task $T_{1,2}$ misses deadline. (c) Optimal Constant Slowdown $s = 0.75$.

one task misses its deadline. It is clear that the utilization cannot be used as a slowdown factor. Note that there are three jobs of unit workload to be finished within the interval $(0, 4)$. So a lower bound on the constant slowdown factor is $3/4 = 0.75$. A schedule with a constant slowdown of $s = 0.75$ is shown in part (c). This is the *optimal constant* slowdown. A constant slowdown lower than 0.75 will result in a deadline miss.

It is seen that the utilization of task τ_1 is higher than that of task τ_2. Executing the task τ_1 at a lower speed will save energy. If we increase the speed of τ_2 and lower the speed of τ_1 by a small amount such that the system is feasible, we will execute at a lower speed for a longer time and save energy. A constant slowdown need not be optimal when the task deadline is less than the period. Assigning different slowdown factors to tasks can be more energy efficient. This property is more evident when the tasks have different power characteristics.

3 Constant Static Slowdown

We consider EDF scheduling on a uniprocessor system where the task deadlines can be less than the period $(D_i \leq T_i)$. In this section, we present algorithms to compute the constant slowdown factor. There is an tiem and energy overhead associated with changing power states and a constant slowdown is preferred as it eliminates this overhead. A constant slowdown is a desired feature if the resource does not support run time change in the operating speed. Algorithms [3] exist to check the feasibility of a task set. Given a feasible task set at maximum speed, our work deals with computing static slowdown factors to minimize the total energy consumption of the system.

3.1 Density slowdown

For the case $D \leq p$, no known algorithm can efficiently (in polynomial time) check the feasibility of a task set. A task set is schedulable [20] if the *density* of the system, $\Delta \leq 1$. If $\Delta < 1$, we set the

slowdown factor $\eta = \Delta$, else ($\Delta \geq 1$) we let the processor run at full speed ($\eta = 1$). Given a feasible system, a constant slowdown of $\eta = min(\Delta, 1)$ guarantees feasibility of the system. We call this constant slowdown as the *density slowdown*.

3.2 Optimal constant slowdown factors

The density slowdown is not optimal and we present an algorithm to compute the optimal constant slowdown factor for a periodic task set.

Theorem 1 *The maximum intensity over all interval $[0,t]$, $0 < t \leq H$, where H is the hyperperiod of the task set, is the optimal constant slowdown factor for a task set.*

For a synchronous task system [3], the intensity of the interval $[t_1, t_2]$ is less than the intensity of the interval $[0, (t_2 - t_1)]$. Thus it suffices to compute the intensity of the intervals $[0,t]$ to compute the maximum intensity interval. Since the task set repeats every hyperperiod, the maximum intensity interval up to the hyperperiod, is the maximum intensity interval. Theorem 1 gives an algorithm to compute the optimal constant slowdown for n periodic tasks. This gives an exponential time algorithm to compute the maximum intensity which is the optimal constant slowdown for a task set. The intensity function increases only at discrete point represented by the set $S = \{t_{(i,k)} = kT_i + D_i | i = 1, ..., n; k \geq 0\}$. It sufficies to check the intensity of the intervals $[0,t]$ s.t. $t \in S$.

3.3 Bisection method

Computing the optimal constant slowdown can take exponential time in the worst case. However, the feasibility of a task set can be tested in pseudo polynomial time provided the processor utilization, $U < 1$. The algorithm is given by Baruah et al. [3].

Theorem 2 *[3] : The 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 $O(\frac{U}{1-U}\{max(T_i - D_i)\})$.*

The algorithm works when the utilization of the system is strictly less than 1. In their result, the system utilization U is considered as a system constant. $(T_i - D_i)$ need not be polynomial in the problem size and Theorem 2 gives a pseudo polynomial time algorithm to check the feasibility of a task set. The algorithm only checks for the feasibility and does not compute the maximum intensity for the task set. We are interested in computing the maximum intensity interval. An interval $[0,t]$ where $t > t_{max}$ can be the maximum intensity interval. So the algorithm by Baruah et al. [3] cannot be used to compute the constant slowdown factor for the system. We use the result to compute the constant slowdown factor.

The time value t_{max} depends on the system utilization and it is proportional to $\frac{U}{1-U}$. 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 have to check all intervals up to the hyperperiod of the task set, which requires exponential time. We present a pseudo polynomial time approximation algorithm with an additional constraint on the processor utilization.

3.3.1 Bisection Algorithm

Let U be the processor utilization at a slowdown of η_i for task τ_i. We impose an additional constraint on the utilization, $U \leq 1 - \varepsilon_u$. Since ε_u is a constant, it bounds t_{max} to $\varepsilon_u^{-1}\{max(T_i - D_i)\}$. We use the pseudo polynomial time algorithm given by Baruah et al. [3] to test the feasibility of the system.

We perform a binary search on the constant slowdown factor. Let U_m be the processor utilization at no slowdown. At a slowdown of $\eta = U_m$, the processor utilization is 1 and this is the lower bound on the constant slowdown factor. However, we impose an additional constraint on the processor utilization, $U \leq 1 - \varepsilon_u$. Let $\eta_l > U_m$ be the constant slowdown when the utilization is $1 - \varepsilon_u$. With the constraint on the utilization, η_l is a lower bound on the constant slowdown factor. If Δ is the density of the system, the upper bound on the constant slowdown is $\eta_u = min(\Delta, 1)$. We perform a binary search in the range $[\eta_l, \eta_u]$ to compute the optimal constant slowdown in this range. We call this algorithm the *bisection method*.

In this bisection method, we first compute the upper and lower bounds η_u and η_l respectively. We check for the feasibility of the system at a slowdown of $\eta_m = \frac{\eta_l + \eta_u}{2}$. If the system is feasible, we update the upper bound to η_m, $\eta_u = \eta_m$. If the system is infeasible, we update the lower bound to η_m, $\eta_l = \eta_m$. This completes one iteration. We compute a new η_m in each iteration. The number of iterations is polynomial in the binary representation of η. The feasibility of the system at a slowdown factor of η_m is checked by the algorithm given by Theorem 2. Since we bound the processor utilization, t_{max} is proportional to $max(D_i - T_i)$ and we have a pseudo polynomial time algorithm.

Let η be the solution returned by the bisection algorithm. If the utilization at η is $1 - \varepsilon_u$ then its an approximate solution. Otherwise, we have the optimum solution to the problem in pseudo polynomial time, an exponential improvement over the optimal constant slowdown. An important point about this approximation algorithm is that we can efficiently check whether the solution computed by the algorithm is the optimal solution.

4 Uniform Slowdown Factors

Constant slowdown factors are not always energy efficient, and we compute uniform slowdown factors for the tasks. Different tasks can have different slowdown factors, however all instances of a task have the same slowdown factor and we call it the *uniform slowdown factor*.

4.1 Power Delay Characteristics

The number of cycles, C_i that a task τ_i needs to complete is a constant during Voltage Scaling. The processor cycle time, the task delay and the dynamic power consumption of a task vary with the supply voltage V_{DD}.

$$P_{switching} = C_{eff} V_{DD}^2 f \tag{3}$$

$$Cycle\ Time\ \alpha \frac{1}{f} = k \frac{V_{DD}}{(V_{DD} - V_{TH})^{\alpha}} \tag{4}$$

where k is a device related parameter, V_{TH} is the threshold voltage, C_{eff} is the effective switching capacitance per cycle and α ranges from 2 to 1.2 depending on the device technology.

We normalize the operating voltage V_{DD} to the maximum operation voltage, V_{MAX}, supported by the processor. Let V_{dd} represent the task voltage normalized to V_{MAX}. The normalized cycle time CT, the

normalized task delay d_i and the normalized task energy E_{dd} as a function of the normalized voltage V_{dd} is represented as:

$$E_i(V_{dd}) = V_{dd}^2 \cdot C_i \tag{5}$$

$$CT(V_{dd}) = V_{dd}(\frac{1-V_{th}}{V_{dd}-V_{th}})^{\alpha} \tag{6}$$

$$d_i(V_{dd}) = C_i \cdot CT(V_{dd}) \tag{7}$$

where V_{th} is the normalized threshold voltage.

4.2 Optimization Problem

We formulate the energy minimization problem as an optimization problem. Let $v \in \mathbb{R}^n$ be a vector representing the voltages V_i of task τ_i. The optimization problem is to compute the optimal vector $v^* \in \mathbb{R}^n$ such that the system is feasible and the total energy consumption of the system is minimized. The energy consumed by task τ_i at voltage V_i is given by Equation 5. Since the task set repeats every hyperperiod, we minimize the energy consumption up to the hyperperiod, H, of the task set. Let $N_i = \frac{H}{T_i}$ be the number of instances of task τ_i up to the hyperperiod of the task set. The total energy consumption of the system up to the hyperperiod is given by Equation 8. The total energy, E, is a function of the voltage vector, $v \in \mathbb{R}^n$. The optimization problem is :

minimize :

$$E(v) = \sum_{i=1}^{n} N_i \cdot V_i^2 \cdot C_i \tag{8}$$

under the constraints :

$$V_{min} < V_i \leq 1 \tag{9}$$

$$C^t : \sum_{i=1}^{n} (\lfloor \frac{t-D_i}{T_i} \rfloor + 1) \cdot d_i(V_i) \leq t \tag{10}$$

Equation 9 constraints the normalized voltage V_i of each task. The operating voltage range is between the normalized minimum voltage V_{min} and the normalized maximum voltage $V_{max} = 1$. The constraint C^t given by Equation 10 specifies that the work load in the interval $[0,t]$ be less that or equal to t, which means the intensity of interval $[0,t]$ be less than or equal to 1. Equation 10 must be true for all t, $0 < t \leq H$, where H is the hyperperiod of the task set. The intensity of an interval depends on the number of task instances in the interval. The number of instances of task τ_i in the interval $[0,t]$ is given as $\sigma_i(t) = (\lfloor \frac{t-D_i}{T_i} \rfloor + 1)$. The cycle time at voltage V_i is given in Equation 6. Since H can be exponential in the problem size, the constraint set can be exponential in the problem size.

4.3 Varying Number of Constraints

The Constraints C^t, $0 < t \leq H$ guarantee the feasible of the task set at vector v. However we need not check all the constraints to test the feasibility of a vector v. Baruah et al. [3] have given a pseudo polynomial time algorithm to check the feasibility of a task system when the utilization is less than 1. They prove that checking the constraints C^t, $0 < t \leq t_{max} = \frac{U}{1-U}\{max_i(T_i - D_i)\}$, guarantee the feasibility of the task set. They consider the utilization U as a system constant. However U is a function of the

vector v and not a constant under voltage scaling. Let $v \in \mathbb{R}^n$ be the vector representing the voltages for each task. The utilization of the system with the task voltages represented by vector v, U_v is as given as:

$$U_v = \sum_{i=1}^{n} \frac{C_i}{T_i} \cdot CT(V_i) \tag{11}$$

The utilization of the system increases as we lower the task voltages. The number of constraints to check the feasibility of a vector are not constant. For each vector $v \in \mathbb{R}^n$, the number of constraints is proportional to $\frac{U_v}{1-U_v}$. If we use a convex programming solver [6], we have to explicitly enumerate all the constraints in the system. All the constraints are checked to decide the feasibility of a vector. Even at low processor utilization, we check all constraints. This leads to checking redundant constraints in the system. The ellipsoid method is better suited for problems of this nature. In the ellipsoid method, the constraints are not specified explicitly. We need a subroutine which checks the feasibility of a vector v and if it is in infeasible, generate a hyperplane that separates the feasible space and the vector v.

4.4 Ellipsoid Method

The ellipsoid method is well suited for our optimization problem and we use the ellipsoid method to compute a *close-to-optimum* solution with some performance guarantees. We explain the terms used in the the rest of the paper. The exact definitions are given in [8] [7]. 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 hyperplane that approximatly 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 3 *[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 result applies to optimizing convex functions whose gradients can be computed [8]. The theorem tells the existence of an algorithm that makes a polynomial number of calls to the separation oracle (separation algorithm) to compute a close-to-optimum solution. The *basic ellipsoid method* [7] is used to solve the optimization problem using the separation algorithm. We now explain the ellipsoid method.

4.4.1 Geometric Interpretation of Ellipsoid Method

We explain the geometry behind the ellipsoid algorithm. We start with a ellipsoid (convex body) containing the feasible space and the optimization function. We check for the feasibility of the center of the ellipsoid. If it is not feasible, the separation oracle returns a separating hyperplane, H that cuts the the ellipsoid into two halves. We select the half containing the feasible space and include it in a new ellipsoid. If the center of the ellipsoid is feasible, we compute the gradient of the optimization function

at the center. This gradient hyperplane splits the ellipsoid into two halves, one containing the optimal solution. By the property of convex functions, we can identify the non-optimal half. We cut the half of the ellipsoid which does not contain the optimal solution. 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 are we have a solution close-to-optimal solution.

4.4.2 Convex Minimization

We show that computing the voltages that minimizes the energy function is a convex minimization [8] problem. Each constraint C^t is a convex function of voltages [13]. Since the intersection of convex bodies is convex, the feasible space is a convex body. The optimization function is also convex. The proofs are given in [13]. Thus we have to minimize a convex function over a convex body. The feasible space is enclosed in the sphere of radius V_{max}. The optimization function E is differentiable and by Theorem 3 we compute a weak optimum solution in polynomial number of calls to the separation algorithm (separation oracle). We use the ellipsoid algorithm [7] to compute the weak optimum.

4.4.3 Separation algorithm

The algorithm to check the feasibility of a task set at a task delay of d_i for task τ_i is given by Baruah et al. [3]. The running time of the algorithm is given by Theorem 2 and depends on the processor utilization. The result holds when the processor utilization is less than 1. The processor utilization increases as we decrease the voltage. As the utilization approaches 1, t_{max} approaches infinity. Thus we may have to check all intervals up to the hyperperiod, leading to worst case exponential time. To bound the running time of the feasibility test, we constraint the utilization to be less than $1-\varepsilon_u$.

If a constraint C^{t_i} is violated for vector u, then the hyperplane $\nabla C^{t_i}(u)(v-u)$ satisfy the property of the separating hyperplane [23]. C^t is differentiable and the separating hyperplane is computed by evaluating the derivative of C^t at vector u. This gives a pseudo polynomial time separation oracle. If we do not bound the processor utilization, we may have to test the feasibility when the utilization is close to 1. This computation can take exponential time. Even if the optimum system utilization is not close to 1, we may have to check the feasibility of vectors where the utilization is close to 1.

4.5 Tasks with Different Power Characteristics

The nature of the tasks in a system vary and tasks have different power characteristics [1]. For example, audio and video application tasks have very different power consumption characteristics. We must consider the task power characteristics to minimize the energy consumption of a system. The constant slowdown techniques compute the slowdown factors based purely on the timing requirements. The slack is uniformly divided among all tasks to lower the energy consumption. The slack can be efficient utilized to further reduce the energy consumption.

The constant slowdown techniques cannot be extended to incorporate tasks with different power characteristics. The ellipsoid method can easily incorporate the power characteristics of the tasks. The total energy function E, given by Equation 8, can be modified to reflect the total energy consumption. Since

the ellipsoid method assumes a convex differentiable function to be minimized, power characteristics of this nature can be handled by the ellipsoid method.

5 Experimental Results

We used the following three application sets for our experiments: Avionics task set [21], INS (Inertial Navigation Control) task set [4] and CNC(Computer numerical control) task set [16]. These task sets are also used in [18] [28]. Since we consider the case $D \leq p$, we have generated new task sets by decreasing the deadlines of the original task sets. Almost all the tasks in the examples have their deadlines equal to the period. We have obtained new examples by decreasing the deadline as a percentage of the original deadline. In the new examples, the new deadlines are set to 100%, 95%, 90%, 85%, 80% & 75% of the original deadline. We compute the energy consumption of the system for a time period equal to the hyper-period of the task set.

The energy and delay characteristics are given by Equations 3 and 4. Recent processors [11], [12] have low operating voltages. We have used an operating voltage range of $0.9V$ and $1.8V$. The threshold voltage is assumed to be $0.6V$ and $\alpha = 1.5$. We compare the energy consumption for the following techniques:

- Density Slowdown
- Optimal Constant Slowdown (OptConst)
- Slowdown by Bisection Method
- Slowdown by Ellipsoid Method

5.1 Uniform Power Characteristics

When the tasks have a uniform power characteristic, the power consumption of a task depends only on the operating voltage. The density slowdown factor is easy to compute. However, it is not energy efficient and has the maximum energy consumption. We compare the energy consumption of the slowdown techniques to that of the density slowdown method. For each example, the percentage energy gains over the density slowdown method are shown in Figure 2. The workload for each task is its worst case execution time. It is seen from the graphs that the the Optimal constant, Bisection and the Ellipsoid methods have comparable energy gains under uniform power characteristics. The optimal constant slowdown and the ellipsoid method, each perform better in some cases. The bisection method is an approximation of the optimal constant slowdown and has comparative lower energy gains.

The computation times for the three techniques are shown in Table 1. We conducted the experiments on a Sun Ultra 5 work station running SunOS. The computation for the density slowdown is negligible and not shown in the table. Though the optimal constant slowdown has a worst case exponential complexity, the hyperperiod is not exponential and it requires the least time. Since the bisection method checks the feasibility of a task set at many points, it takes more time to compute the result. The ellipsoid method requires the maximum amount of time. The computation time is three orders magnitude compared to that of the optimal constant slowdown. Each iteration of the ellipsoid method requires matrix

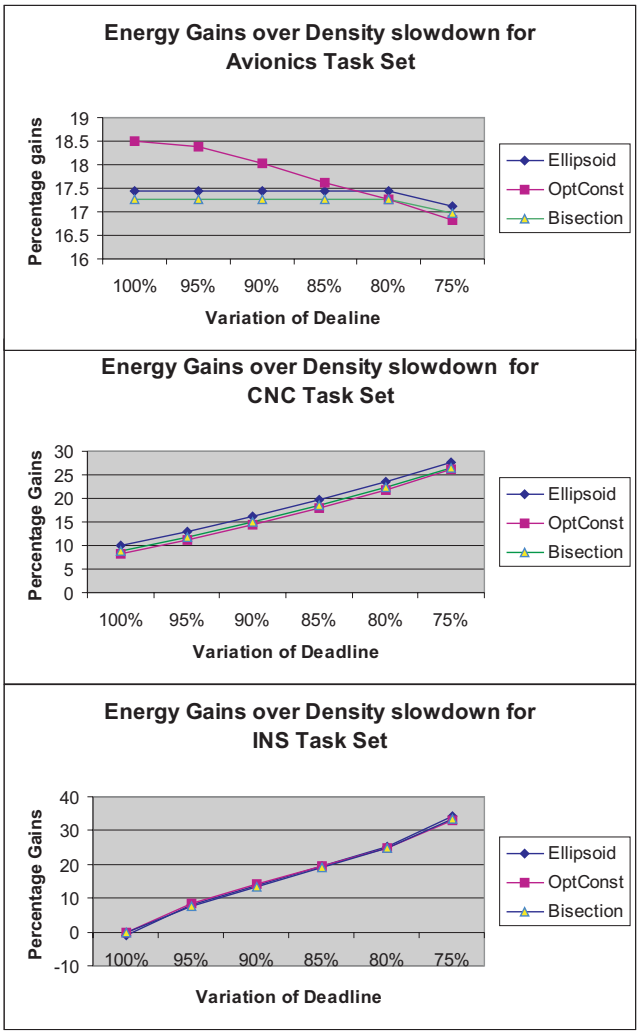

Figure 2. Percentage Energy savings of the slowdown techniques over the density slowdown algorithm for the three task sets:

(a) Avionics task set (b) CNC task set (c) INS task set

computations. We performs matrix multiplication and matrix transpose operations to compute the new ellipsoid and its center. Each iteration is expensive and results in a large computation time. The size of the matrix depends on the number of tasks in the system. As the number of tasks increase, matrix operations require more time. The Avionics task set has the maximum number of tasks and results in the maximum computation time as seen from Table 1. Since the computations are performed off-line, we justify a larger computation time for the energy gains.

5.2 Different Power Characteristics

Due to the diverse nature of the tasks in a system, tasks can have distinct power characteristics. Scenarios where tasks have different power characteristics are given in [1]. We compare the energy efficiency of the techniques for tasks with different power characteristics. Only the ellipsoid method considers

Table 1. Computation time for the slowdown methods at 100% task deadline

Example	Ellipsoid	Optimal Constant	Bisection
Avionics	20.6 sec	0.02 sec	0.16 sec
CNC	1.59 sec	0.004 sec	0.01 sec
INS	2.6 sec	0.02 sec	0.22 sec

the power characteristics in the computation of slowdown factors. The weak optimal solution can be computed for all convex differential power characteristics. For experimental results we restrict to *linear* power characteristics. We consider the following two distributions similar to that by Aydin et. al [1].

- **Uniform Distribution**, where the coefficients of the power function coefficients of the tasks are uniformly distributed between 1 and k. In our experiments, $k \in [1, 8]$.
- **Bimodal Distribution**, represents the case where there are two types of tasks in the system, one with low power function coefficient and the other with high power function coefficient. Task sets are created with 60% of the tasks having power function coefficients of 1, and the remaining 40% tasks having a high power function coefficient k. We vary k in the range $[1, 8]$ for experimental results.

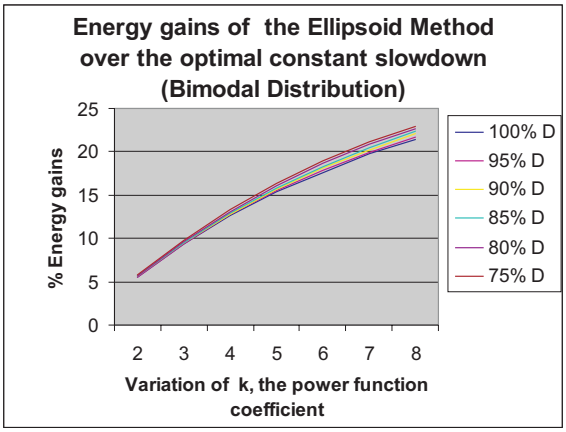

Figure 3. Percentage gains of the ellipsoid algorithm over the optimal constant slowdown for the CNC task set

We compare the energy consumption of the ellipsoid method to that of the optimal constant slowdown. We consider the CNC task set to compare the energy consumption. Figure 3 shows the percentage gains for the bimodal distribution. It is seen that the energy gains increases with the increases in the power function coefficient k. The ellipsoid method takes into account the power characteristics and hence results in more energy efficient slowdown factors. The energy gains increase as the deadline decreases. With the decrease in deadline, the feasible space becomes smaller. Moreover, tasks having different slowdown factors exploit the slack time to a greater extent. So a constant slowdown is not as energy efficient as the uniform slowdown. The diffence is small for the bimodal case.

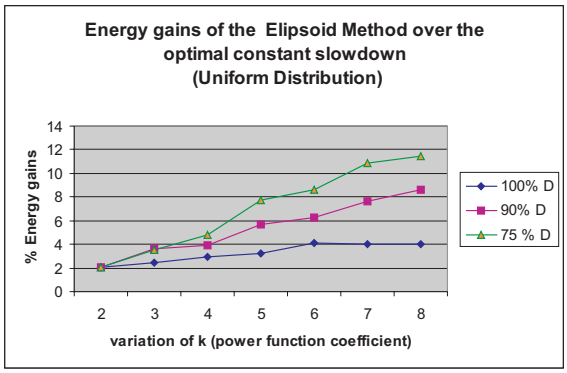

Figure 4. Percentage gains of the ellipsoid algorithm over the optimal constant slowdown for the CNC task set

We use the CNC task set to compare the energy gains for the uniform distribution. Figure 4 shows the energy gains over the optimal constant slowdown for a uniform distribution. The energy gains increase with the power function coefficient k. The ellipsoid method is energy efficient because the slowdown factors depend on the power function coefficient k. It is seen from Figure 4 that as the deadline decreases the energy gains are significantly larger.

6 Conclusions and Future Work

We present algorithms to compute static slowdown factor under EDF scheduling for a periodic task set. In this paper, we consider the case where the task deadlines are less than the task period. Experimental results show that the computed slowdown factors save on an average 20%-25% energy over the known techniques. The algorithms have the same time complexity as the algorithm to check the feasibility of the task set. We use the ellipsoid method to compute static slowdown factors that are close to the optimal solution. The energy gains are significantly greater when the tasks have different power characteristics. We gain as much as 20% more energy savings over the optimal constant slowdown. 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.

As technology is improving, the leakage power dissipation will be a significant part of the power consumption of the system. We plan to consider the effects of leakage current in our power model. As a future work, we plan to compute the slowdown factors when the processor supports discrete voltage levels. We plan on implementing the techniques in an RTOS such as eCos and measuring the power consumed on a real processor.

Acknowledgments

We would like to thank George Lueker for the discussions on formulating the problem and helping understand the ellipsoid method. The authors acknowledge support from National Science Foundation (Award CCR-0098335) and from Semiconductor Research Corporation (Contract 2001-HJ-899).

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 *Euromicro Conference on Real-Time Systems*, Delft, Holland, June 2001.

[2] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium*, London, England, December 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] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. In *IEEE Transaction on Software Engineering*, volume 21, May 1995.

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

[6] Computational Optimization Laboratory. Copl-lp. University of Iowa. http://dollar.biz.uiowa.edu/col/.

[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 *International Symposium on Low Power Electronics and Design*, pages 46–51, 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*, 2001.

[11] Intel StrongARM. AMD Inc. (http://www.amd.com/armtech/StrongARM).

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

[13] R. Jejurikar and R. Gupta. Computing static slowdown factors under edf scheduling when deadline less than period. In *CECS Technical Report #02-36, University of California Irvine,* Dec. 2002.

[14] R. Jejurikar and R. Gupta. Energy aware edf scheduling with task synchronization for embedded real time operating systems. In *International Workshop on Compilers and Operating Systems for Low Power*, 2002.

[15] R. Jejurikar and R. Gupta. Energy aware task scheduling with task synchronization for embedded real time systems. In *International Conference on Compilers Architecture and Synthesis for Embedded Systems*, 2002.

[16] N. Kim, N. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual assesment of visual real-time systems: a case study on cnc controller. In *IEEE Real-Time Systems Symposium*, Dec. 1996.

[17] 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 *Design Automation and Test in Europe*, 2002.

[18] P. Kumar and M. Srivastava. Predictive strategies for low-power rtos scheduling. In *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 343–348, 2000.

[19] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. In *Journal of the ACM*, pages 46–61, 1973.

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

[21] C. Locke, D. Vogel, and T. Mesler. Building a predictable avionics platform in ada: a case study. In *Proceedings IEEE Real-Time Systems Symposium*, 1991.

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

[23] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Printice Hall, 1982.

[24] 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.

[25] F. P. Preparata and M. l. Shamos. *Computational Geometry, An Introduction*. Springer Verlag, 1985.

[26] 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, June 2001.

[27] G. Quan and X. Hu. Minimum energy fixed-priority scheduling for variable voltage processors. In *Design Automation and Test in Europe*, pages 782–787, March 2002.

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

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

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

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

[32] 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.

A Appendix

A.1 Convex Minimization Problem

We prove that the in the minimization problem given by Equations 8, 9 and 10 is a convex minimization problem. We use the following result of convex functions :

- If β is a constant with $\beta \geq 1$ or $\beta \leq 0$ then a function over x, x^{β} is convex [23].
- The sum of two convex functions is convex [23].
- Given a convex function $f(x)$ and a positive constant c, $c \cdot f(x)$ is a convex [23].

We want to minimize the total system energy $E(v)$. It is a function of the task voltage V_i and is given by :

$$E(v) = \sum_{i=1}^{n} N_i \cdot V_i^2 \cdot C_i$$

where v is a vector in $\mathbb{R}^n$ representing the voltage values of the n tasks in the system.

By the above results, V_i^2 is convex. The function $N_i \cdot V_i^2 \cdot C_i$, is convex since N_i and C_i are constants. The energy function $E(v)$ is the sum of n convex functions and hence convex. This proves the convexity of function $E(v)$

We now prove that each constraint C^t is convex. Each constraint C^t is represented as :

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

If the function $d_i(V_i)$ is convex, then the constraint C^t is the sum of convex functions and hence convex. We now prove that $d_i(V_i)$ is convex.

$$d_i(V_i) = C_i \cdot V_i (\frac{1 - V_{th}}{V_i - V_{th}})^{\alpha}$$

Since C_i, V_{th} and α are constants, we need to prove convexity of $\frac{V_i}{(V_i - V_{th})^{\alpha}}$. Since $V_i > V_{th}$, we can shift the origin to V_{th} to have,

$$d_i(V_i) = \frac{V_i + V_{th}}{V_i^{\alpha}}$$

$$= V_i^{1-\alpha} + V_{th} \cdot V_i^{-\alpha}$$

Since $\alpha > 1$, $(1 - \alpha)$ and $(-\alpha)$ are negative and $V_i^{1-\alpha}$ and $V_i^{-\alpha}$ are convex. Thus $d_i(V_i)$, a sum of convex functions is convex. This proves that all the constraints are convex in nature. The intersection of convex constraints results in a convex body. This proves that the feasible space is a convex body and the function to minimize is convex to have a convex minimization problem.

A.2 Example Task Sets

The examples used in the experiments are given below. We give the examples with their original deadlines and show how we generate new examples with smaller deadlines.

A.2.1 Original Periodic task set examples

These are periodic task sets and the periods, deadlines and WCETs are specified.

```
INS (Inertial Navigation Control) task set
------------------------------------------
 Period  Deadline WCET
 2500  2500  1180
 40000  40000  4280
 625000  625000  10280
 1000000  1000000  20280
 1000000  1000000  100280
 1250000  1250000  25000



CNC(Computer Numerical Control) task set
----------------------------------------
 Period  Deadline WCET
 2400  2400  35
 2400  2400  40
 2400  2400  165
 2400  2400  165
 9600  4000  570
 7800  4000  570
 4800  4800  180
 4800  4800  720



Avionics task set
-----------------
```

Period	Deadline	WCET
200000	5000	3000
25000	25000	2000
25000	25000	5000
40000	40000	1000
50000	50000	3000
50000	50000	5000
59000*	59000*	8000
80000	80000	9000
80000	80000	2000
100000	100000	5000
200000	200000	1000
200000	200000	3000
200000	200000	1000
200000	200000	1000
200000	200000	3000
1000000	1000000	1000
1000000	1000000	1000

- **Slight modification in Avionics example**

Note : In the avionics task set there is a task with period 59000. This causes the hyperperiod of the task set to be very large and the number of jobs to be considered increases drastically. We tried to compute the optimal slowdown factors with no modification. However the process did not complete till a few days. So we decreased the period of that task. This will ensure the correctness of the slowdown factors and we compute the optimal slowdown factors for the modified task set. We set the new period and deadline for that particular task to 50000. The WCET is left unchanged. We ran the experiments with this modification.

A.2.2 Decreasing the Deadline

For each of the examples above, we generate examples where the tasks have deadlines less than the period. The deadlines are varied as 100% 95% 90% 85% 80% 75% of the original deadline. For example, when the deadlines are decreased to 95% of the original deadline, the new deadlines are computed as $D_{new} = 0.95 * D_{orig}$, where D_{new} and D_{orig} are the new and original deadlines for the task respectively.