Computing optimal static slowdown factors for periodic tasks under EDF Scheduling

Ravindra Jejurikar and Rajesh Gupta

ICS

TECHNICAL REPORT

Technical Report #02-02
January 25, 2002

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425

Information and Computer Science
University of California, Irvine

Computing optimal static slowdown factors for periodic tasks under EDF scheduling

Ravindra Jejurikar Rajesh K. Gupta

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

Technical Report #02-02

January 25, 2002

Abstract

Frequency scaling of a processor based on slowdown factors, gives considerable power savings. The slowdown factors are computed from the task set executing on the processor. These factors are non-trivial for periodic tasks with deadlines not equal to their period. This paper describes how to compute static slowdown factors for a task set with an underlying earliest deadline first(EDF) scheduler. We give a density slowdown *method, which efficiently computes a constant slowdown factor. The constant slowdown factor not being optimal, we compute the optimal slowdown factors by looking at all the jobs up to the hyper-period of the task set. This is based on Yao's optimal algorithm for non-periodic tasks. Dynamic slowdown techniques further enhance the power savings. We show that dynamic slowdown factors can be easily incorporated over these static slowdown factors, thereby increasing the total energy savings.*

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. Especially in portable battery operated devices, energy consumption of the device is a very important factor. Reducing the frequency decreases the switching activity thereby minimizing power. The quadratic dependence of power on voltage has led to voltage scaling techniques for energy savings. In real-time systems we want to minimize energy while adhering to the deadlines. Power and deadlines are often contradictory goals and we have to judiciously manage time and power to achieve our goal of minimizing energy.

Towards minimizing the energy consumption of an embedded system, a lot of work has been done at various levels including hardware level [20], system design [16], compiler [3, 8, 17], operating system [4, 13] and application level. In this paper we focus on the system level power management via computation of static slowdown factors and dynamic factors. We assume a real-time system where the tasks run periodically in the system and have deadlines. These jobs are to be scheduled on a single processor based on a preemptive scheduler such as EDF [21] or rate monotonic scheduler(RMS) [6]. The percentage of time for which a processor is used is called the *utilization factor*. The processor speed can be varied to minimize energy usage. This schedule of the processor speed is called a *speed schedule*. If the speed schedule is a constant value over the entire time interval, it is called a *constant slowdown factor*. The slowdown factor is the ratio of the scheduled speed to the maximum speed. If the jobs are scheduled on a variable speed processor, the execution time of the job varies depending on the processor speed. The goal is to have a speed schedule for the processor which minimizes the energy consumption while meeting deadlines.

Shin and Choi [18] have presented a power conscious fixed priority scheduler for hard real time systems. Due to the periodic nature of the jobs they know the arrival time of the next job. If there is only one job in the ready queue, they slow down the processor so that the job finishes just before the next job arrival. Further enhancements [19] were given, where the authors have exploited the idea of uniform slowdown. They perform rate monotonic analysis on the task set to compute a constant static slowdown factor for the processor. Gruian [7] observed that only one iteration was performed to compute the constant slowdown factor. Performing more iterations gives better slowdown factors for the individual task types.

Yao, Demers and Shanker [22] presented an optimal off-line speed schedule for a set of N jobs. The running time of their algorithm is $O(N^2)$ and can be reduced to $O(N \log N)$ by the use of segment trees. The analysis and correctness of the algorithm is based on an underlying EDF scheduler, which is an optimal scheduler. An optimal schedule for tasks with different power consumption characteristics is considered by Aydin, Melhem and Mossé [1]. These authors [2] have proven that the utilization factor is the optimal slowdown when the deadline is equal to the period. Quan and Hu [15] discuss off-line algorithms for the case of fixed priority scheduling. Devices like sensors are periodic in nature with a response time smaller than the period. Thus we consider periodic tasks where the deadlines are less than the period. In this paper we consider algorithms to compute the constant slowdown factor and optimal slowdown factors.

The worst case execution time (WCET) of a task is not usually reached and there is dynamic slack in the system. Pillai and Shin [14] recalculate the slowdown when a task finishes before its worst case execution time. They use the dynamic slack while meeting the deadlines.

All the above techniques guarantee meeting the deadlines. Systems where it is critical to meet all the

deadlines are termed as hard real-time systems. On the other hand, we have systems like communication devices and multimedia where we can afford to miss a few deadlines. Such systems are termed as soft real-time systems. One can have aggressive power saving techniques for soft-real time systems. In these systems the run time characteristics of the tasks can be exploited for more energy savings. Kumar and Srivastava [10] extended the work by Shin [18] by predicting the execution time of the job. A comprehensive work is done by Raghunathan and Srivastava [16] explaining a generalized framework to incorporate static and dynamic slowdown factors.

Our contribution is as follows: We have considered the problem of scheduling periodic tasks under EDF scheduling with the deadline less that the period. We compute the constant static slowdown factors. When deadline is less than the period this constant factor is not optimal. We have given an off-line algorithm to compute the optimal slowdown factors, which exploits Yao's algorithm for scheduling non-periodic tasks. Thus we compute an optimal speed schedule for the periodic tasks.

The rest of the paper is organized as follows: Section 2 formulates the problem with a motivational example. In section 3, we explain the algorithms and prove their correctness. Section 4 deals with using dynamic slowdown factors over the static slowdown factors. The implementation and experimental results are given in section 5 and 6 respectively. We conclude in section 7.

2 Preliminaries

In this section we first introduce the necessary notation and formulate the problem. This is followed by a motivational example.

2.1 System Model

We consider a set $T = \{T_1, ..., T_n\}$ of n periodic real time tasks. Each task T_i is represented by a triplet (p_i, D_i, e_i) where p_i is the period of the task, D_i is the relative deadline, and e_i is the WCET for the task. The tasks are assumed to be independent and preemptive. The tasks are scheduled on a single processor, which supports multiple frequencies. 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 is minimized.

Each invocation of the task is called a *job* and the j^{th} invocation of task T_i is denoted as $T_{i,j}$. The *density of a task* is defined as the ratio of the execution time e_k over the minimum of period and deadline, $density(T_i) = e_i/min(p_i, D_i)$ The *density of the system* is defined as the sum of the densities of all the tasks in the system and is denoted by Δ. The utilization factor for a task T_i is defined as $u_i = e_i/p_i$ and the processor utilization for a task set, U is the sum of the utilization factors for each task. $U \leq 1$ is a necessary condition for the feasibility of any schedule [21].

2.2 Variable speed Processors

A wide range of processors support variable voltage and frequency levels. Voltage and frequency levels are in a way coupled together. 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, that 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 speed can be varied continuously from S_{min} to the maximum supported speed S_{max}. We can normalize speed to the maximum speed to have a continuous operating range of $[s_{min}, 1]$, where $s_{min} = S_{min}/S_{max}$.

We define a *slowdown function* $f(t)$ over a time interval. A slowdown function over a time interval $[t_s, t_f]$ is a function $t \rightarrow s$, from time to slowdown factor, where $t \in [t_s, t_f]$, t has integral values and $s \in [s_{min}, 1]$. This function tells us the slowdown value at a time instance t. The slowdown function we are considering has values at discrete time intervals. The speed cannot change within a time interval $[t_i, t_{i+1}]$. A slowdown function is used to indicate changes in processor speed. The function $f(t)$ can be stored as a set of (t_i, s_i) pairs, where time t_i indicates a change in the slowdown factor to value s_i. We define the *size of the slowdown function* as the minimum number of (t, s) pairs needed to represent the slowdown function. The size of the slowdown function is also referred to as the size of the solution.

2.3 Motivational example

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

$$J_1 = \{2, 2, 1\}, J_2 = \{5, 4, 1\} \tag{1}$$

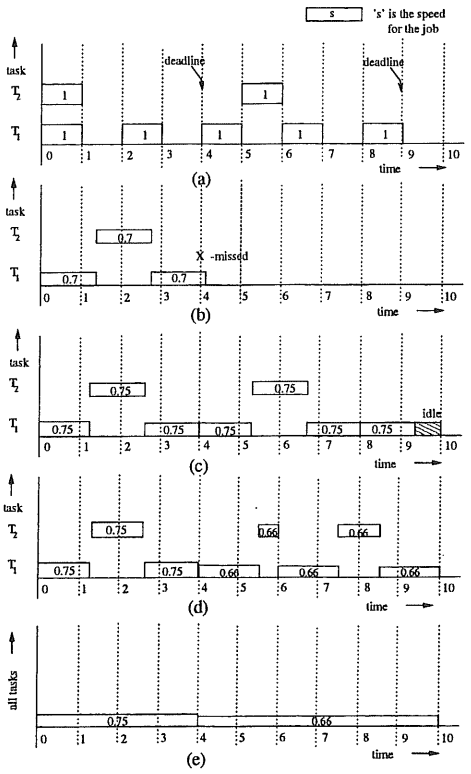

Figure 1. (a) Task arrival times and deadlines. (b) Slowdown $s = 0.70$, task $T_{1,2}$ misses deadline. (c) Slowdown $s = 0.75$ and inherent slack. (d) Optimal speed schedule. (e) Optimal slowdown function.

This 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 they are different from the period.

The jobs are assumed to be scheduled on a single processor by an EDF scheduler, which is an optimal scheduler [21]. The processor utilization U for this task set is $(1/2+1/5) = 0.7$ and there is a feasible schedule at full speed. If the processor utilization U is used as a slowdown factor, it can be seen that *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 one task misses its deadline. It is clear that the utilization cannot be used as a slowdown factor. For the above example the system density, $\Delta = (1/2+1/4) = 0.75$. For a slowdown factor $s = 0.75$, we guarantee meeting all deadlines however there is inherent slack in the system. The system remains idle in the time interval $[9.33, 10]$ as shown in figure 1(c).

We can utilize this slack to get improved slowdown factors. The slowdown values are a function over time, $f(t)$. The optimal slowdown is shown in part(d) of figure 1. The slowdown factor is 0.75 in the interval $[0,4]$ and 0.66 in the interval $[4,10]$. It is seen that there is one context switch at $t = 6$. If we lower the speed at any point, it will result in a deadlines miss. The slowdown function for the optimal solution is shown in part(e). The slowdown function for the time interval $[0,10]$ is represented as $\{(0,0.75),(4,0.66)\}$ and has a size of 2.

3 Static Slowdown

We compute optimal slowdown factors for a system with EDF as the scheduling policy. We first talk about computing a constant slowdown factor, followed by computing the optimal slowdown factors.

3.1 Constant slowdown factor

A constant slowdown for the processor is a desired feature. There is an overhead associated with changing power states and a constant slowdown is preferred as it eliminates this overhead.

3.1.1 Deadline $\geq$ period

It is known [21] that a system of independent, preemptive tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a resource (processor) if and only if their total utilization (U) is less than or equal to 1. Given a task set with $D = p$, we can slowdown the processor by a factor of U [2]. This slowdown will increase the processor utilization to 1. We can schedule the task set at this slowdown state and guarantee feasibility. This is the optimal slowdown factor for the task set.

For the case deadline greater than the period, setting the slowdown factor to U will ensure the completion of the jobs by the end of their period. As $D \geq p$, none of the deadlines are missed. Furthermore, the processor utilization is a lower bound on the slowdown factor (by definition of utilization factor), This implies that $s = U$ is optimal for the case $D > p$.

Thus for the case of $D \geq p$, $s = U$ is *optimal* and can be efficiently computed.

3.1.2 Deadline $<$ period and Density Slowdown

For the case $D < p$, a necessary and sufficient condition for the task set to be schedulable is not known. The task set is schedulable if the *density* of the system, $\Delta \leq 1$ [21].

If $\Delta < 1$, we set the slowdown factor $s = \Delta$, else ($\Delta \geq 1$) we let the processor run at full-speed, $s = 1$. In other words the slowdown of the system is $s = min(\Delta, 1)$. We call this constant slowdown as the *density slowdown.*

Lemma 1 *Under EDF scheduling, slowing down the processor speed to* $s = min(\Delta, 1)$ *will not introduce any deadline misses.*

Proof 1 *We lower the speed of the processor, only for the case* $\Delta < 1$. *After slowdown, the new density of the system increases to 1. As the system is schedulable when* $\Delta \leq 1$, *the task set will be schedulable and will not introduce any deadline misses. Thus slowing down the processor does not introduce deadline misses.*

3.2 Optimal slowdown algorithm

Given a task set, we are interested in computing the optimal slowdown function $f(t)$. In the previous section we have seen that the constant static slowdown factors are optimal for the case $D \geq p$. In this section we compute optimal slowdown for $D < p$, based on Yao's optimal off-line scheduling algorithm for aperiodic tasks [22].

3.2.1 Yao's Optimal Schedule algorithm

Given a set of N jobs with their arrival times and relative deadlines, the optimal slowdown for this job set can be computed by Yao's *Optimal schedule* algorithm [22]. In the algorithm the jobs are represented as intervals called *job intervals.* A job interval is represented as (a_i, e_i, d_i), a_i being the arrival time of the job, d_i the deadline and e_i the number of cycles (workload) of the job. The solution is a slowdown function based on an EDF scheduler. The algorithm runs in time polynomial in N. The details and the correctness of the algorithm are given in [22]. The algorithm takes as an input a set of jobs over a time period and returns a slowdown function $f(t)$ over this time period.

3.2.2 Optimal Slowdown when $D < p$

The procedure to compute the optimal slowdown factors when $D < p$ is shown in figure 2. In line(1) of figure 2, we compute the hyper-period, the least common multiple (lcm) of the periods of all the tasks. For each task T_i, consider all jobs up to the hyper-period. Form job intervals for all these jobs. This is done in lines$(4-6)$ in figure 2. We compute the optimal slowdown factors for this job set by Yao's algorithm, which is line(8). It computes a slowdown function $f(t)$, giving optimal slowdown values in the interval $[0, hyperperiod - 1]$.

Yao's algorithm is optimal for the given set of jobs i.e. all jobs up to the hyper-period in our case. The same job sequence will be repeated in every future hyper-period interval. The same slowdown function is used in every hyper period interval. As the same job sequence is repeated, we have an optimal slowdown function.

```
Optimal-EDF-Schedule(task set T):
{
(1) hyper-period = lcm(all p_i in T)
(2) Job set J = φ; (empty set)
(3) for each task T_i in T {
(4)     J_i = all J_{i,j}, j = 0,1,...hyper-period/p_i − 1
(5)     job interval for J_{i,j} is
                (j * p_i, e_i, j * p_i + D_i);
(6)     J = J + J_i
(7) }
(8) Yao's Optimal-Schedule(J);
(9) return optimal slowdown function;
}
```

Figure 2. Algorithm for computing optimal power schedule under EDF scheduling

We look up to the hyper-period to compute the optimal-schedule. This can potentially have an exponential number of jobs and result in a large computation time. However this computation is done off-line.

3.2.3 Implementing within RTOS

The computation to find the solution could be large, depending on the example. However for implementing the optimal slowdown policy in an RTOS, we only need the slowdown function. The important metric for this technique is the size of the solution, which indicates the voltage changes along the time line. We have seen in our experiments that the size of the solution is small. On a average the size is only of the order of tens. We store the solution in the operating system and the memory overhead is only a few hundred bytes.

4 Dynamic slowdown

The worst case execution time of a job is rarely reached. This results in the dynamic slack in the system. We can use this dynamic slack to further slow down the processor to have more energy savings. Dynamic slowdown factors computation techniques [16, 14] are of two types. One which guarantees meeting all deadlines [14] and the other where we are allowed to miss some deadlines [10]. In the first type, the WCET is always assumed and the present dynamic slack is utilized. Some techniques

[10] anticipate the coming slack to perform aggressive power management. These methods can end up missing a few deadlines.

We use the technique given by Kumar and Srivastava [10] to compute dynamic slowdown factors. This technique is meant for soft real-time systems. In this technique a history of the execution times of the jobs is maintained and this information is used to predict the next execution time. The WCET is divided into slots, each indicating a window of the execution time. At the completion of each job, the slot referring to its execution time is incremented. The prediction is the weighted average of these slots. The dynamic slowdown factor s_d as a function of the predicted execution time e_{pred} is given as $s_d = e_{pred}/e_i$. We use dynamic slowdown factors over the static factors. We can incur some deadline misses due to wrong predictions.

5 Implementation

We have written a simulator in *parsec* [11], a C based discrete event simulator. We have implemented the scheduling techniques and algorithms in this simulator.

5.1 Simulator

We have implemented a generic simulator to enable us use the same simulator for all the scheduling algorithms. The simulator is as shown in figure 3. It consists of two entities, the *task manager* and the *Real Time operating system(RTOS)*. The task manager has the information of the entire task set. It generates jobs for each task type depending on its period and sends it to the RTOS entity.

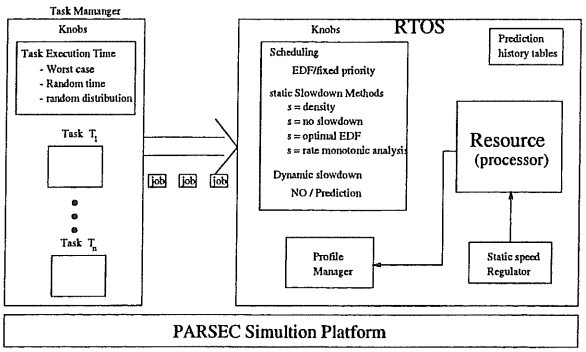

Figure 3. Generic simulator

The RTOS is the heart of the simulator. It schedules the jobs on the resource(processor) and checks for deadline misses. It changes the processor from one slowdown state to another to minimize the energy consumed. The operating system supports APIs to change the scheduling policy and to select a particular static and dynamic slowdown algorithm. For the implementation of our optimal EDF slowdown technique, we also need a *static speed regulator*. The slowdown function is stored here and this entity issues commands to change the speed of the processor. It changes the speed according to the solution. The *profile manager* profiles the energy consumed by each task and calculates the total energy consumption of the system. It keeps track of all the relevant parameters viz. energy consumed, missed deadlines,

voltage changes and context switches. The OS also manages the history tables needed in predicting execution times of tasks.

6 Experimental Results

We use the power model as given in [16, 10]. The power P as a function of slowdown is given by

$$P = f(s) = 0.248 * s^3 + 0.225 * s^2 + 0.0256 * s + \sqrt{311.16 * s^2 + 282.24 * s} * (0.0064 * s + 0.014112 * s^2).$$

The above equation is obtained by substituting $V_{dd} = 5V$ and $V_{th} = 0.8V$ and equating the power and speed equations given below. The speed s is the inverse of the delay. The details are given in [16].

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

$$Delay = \frac{kV_{dd}}{(V_{dd} - V_{th})^2} \alpha \frac{1}{f} \tag{3}$$

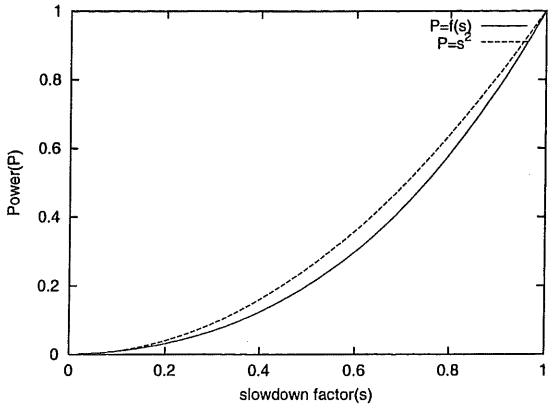

Figure 4. Power function $f(s)$ vs. s^2

The plot of the power function in shown in figure 4. It can be seen that it tracks s^2 closely. The power and speed relation is accurately derived from the switching capacitances and the relation between gate delay and the operating voltage and frequency. A detailed explanation is given in the referred papers.

We compare the processor energy usage for the following techniques:

- **Density slowdown**: This algorithm sets a constant slowdown factor $s = min(\Delta, 1)$.
- **Optimal slowdown** : Here the optimal slowdown factors are computed by looking at jobs up to the hyper-period of the task set.
- **RMA slowdown** : The algorithm to compute slowdown factors for each task is given by Gruian [7]. It computes the static factors by performing Rate Monotonic Analysis (RMA), looking at a window size of the maximum of the task periods. The case of $D < p$ is also considered. This algorithm has good results in practice.

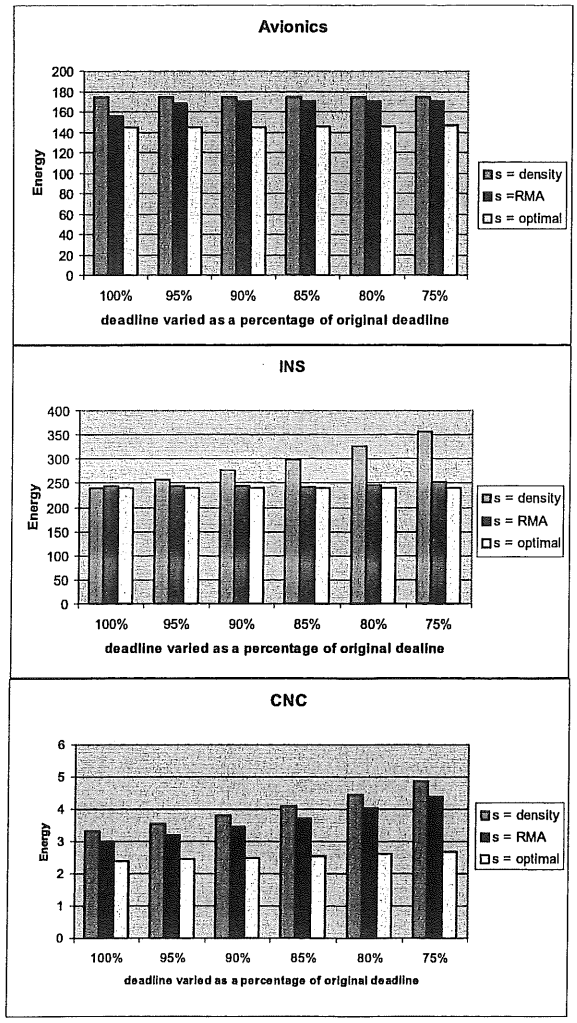

Figure 5. Examples with deadline varied as a percentage of original deadlines

The above algorithms were used on three application sets, Avionics task set [12], INS (Inertial Navigation Control) task set [5] and CNC(Computer numerical control) task set [9], which are the task sets used in [10, 18].

Since we are considering the case $D \leq p$, we have generated new task sets by decreasing the deadlines of the original task sets. Almost all the task sets in the examples have deadlines equal to period. We have obtained the 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 ran the simulation up to the hyper-period of the task set and have computed the energy consumed in this time period. The results of each example are shown in their respective graphs in figure 5. The percentage gains obtained by the optimal algorithm over the density algorithm and the RMA algorithm is shown in figure 6. It can be seen that the percentage gains increase as the deadline is further decreased.

We have as much as 45% gains over density slowdown and 38% gain over the RMA slowdown. We have an average savings of 20%. The percentage gains in the INS example are not very high. In this example there is little workload with big deadlines. Even after decreasing the deadlines, the processor

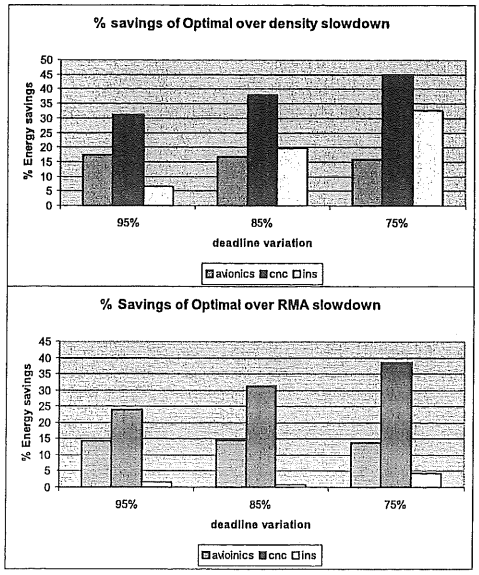

Figure 6. percentage gains over the two algorithms

Table 1. Computation time of optimal algo

Deadline(D)	Avionics	INS	CNC
$D = 1.00 * D_{orig}$	1.3846 sec	1610.34 sec	1.472 sec
$D = 0.90 * D_{orig}$	4.6817 sec	1633.32 sec	2.9515 sec
$D = 0.85 * D_{orig}$	4.6653 sec	1642.31 sec	2.9541 sec
$D = 0.80 * D_{orig}$	4.6654 sec	2123.24 sec	3.383 sec
$D = 0.75 * D_{orig}$	8.7276 sec	3542.03 sec	3.4105 sec

usage is not increased and there is less slack compared to the other algorithms. This has resulted in small gains. We achieve more gains as the deadline is further decreased. In all the other examples the energy savings are clearly seen.

The optimal slowdown has about 40% gains compared to the other two algorithms we have considered. These are large gains and will make the device energy efficient and increase the battery life if the deadlines are less than the period.

6.1 Computation time

The computation time for the optimal algorithm is of higher order magnitude compared to the density slowdown and RMA slowdown algorithms. Computation time depends on the number of jobs arising from the hyper-period (N). This could be exponentially large. We ran the experiments on a sparc SUNW, Sun-Blade-100 running SunOS. The computation time for the examples are given in table 1.

The computation time for the density slowdown method is almost negligible, as we only need to

execute n division and n addition operations. The RMA algorithm takes time of the order of tens of milliseconds. From the table it is seen that the off-line computation cost increases with the reduction of the deadline. However this extra computation cost is compensated by higher energy savings. For the INS example the hyper-period is equally large as the other examples. However there is a task with a small period, which results in a large number of jobs (N) up to the hyper-period. As the running time is a function of N, the running time is large.

6.2 Dynamic slowdown

We performed experiments with dynamic slowdown techniques to utilize the dynamic slack. We used the execution time prediction technique [10] to calculate the dynamic factors. Similar to the work in [10] and [18], we vary the best case execution time (BCET) of a task as a percentage of its WCET. Tasks were generated by a Gaussian distribution with mean, $\mu = (WCET + BCET)/2$ and a standard deviation, $\sigma = (WCET - BCET)/6$. It is seen that the deadline misses are comparable to the other techniques and we have more energy savings. We continue to save about 20% more energy.

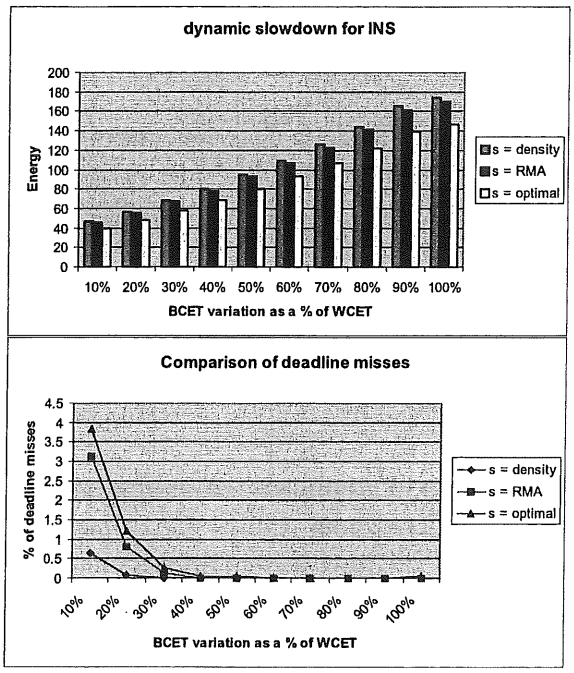

Figure 7. Dynamic Slowdown for INS task set

We show the energy consumption and the percentage of missed deadlines in figures 7 and 8. Figure 7 is a comparison for the INS task set where the deadline is reduced to 90% of the original deadline. It is seen that as we increase the BCET, the energy consumption increases. As we generate task sets with a Gaussian distribution with μ and σ proportional to the BCET, their workload increases with BCET and in more energy usage. It can be seen that the optimal algorithm continues to be the most energy efficient one. It is clear that we will end up missing some deadlines as we are always operating at the critical point. Any misprediction will lead to a deadline miss. However it can be seen that percentage deadline misses is very close to that of the RMA algorithm. As the BCET increases the percentage of deadline

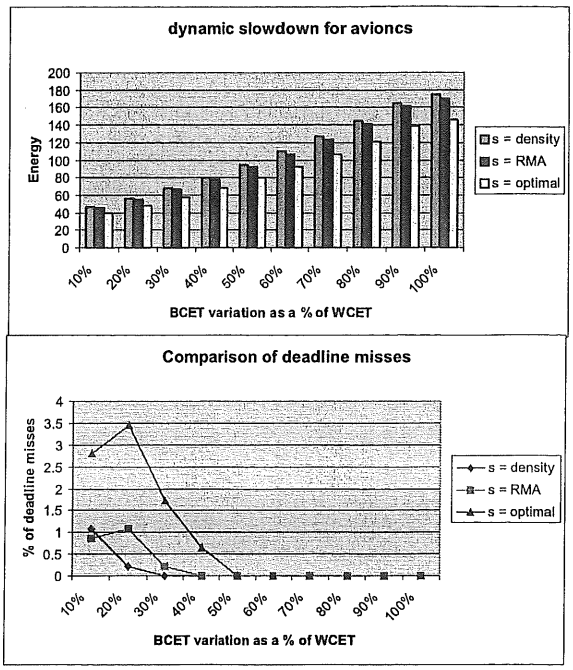

Figure 8. Dynamic Slowdown for Avionics task set

misses keeps decreasing and all the algorithms have almost no deadline misses. Thus for higher BCET values we get the energy gains for free. To support our claims we also show the results for the Avionics task set with the deadlines reduced to 80%. It is shown in figure 8.

7 Conclusion

In this paper, we have given an algorithm to computed the optimal slowdown factors for a periodic task set on a variable speed processor. Experimental results show that we save as much as 30%-40% energy compared to known techniques. This will have a great impact on the energy utilization of portable and battery operated devices. Though the computation of the optimal factors can take long, the solution is generally small and can be stored in the system.

We would like to find out whether the optimal speed schedule can be computed faster. We have computed optimal slowdown factors for an EDF scheduler. In future, we would be interested in computing optimal schedules for other scheduling policies. We will also be implementing the techniques in a RTOS like eCos and compare the results.

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] A. Azevedo, R. Cornea, I. Issenin, R. Gupta, N. Dutt, A. Nicolau, and A. Veidenbaum. Architectural and compiler strategies for dynamic power management in the copper project. In *IWIA 2001 International Workshop on Innovative Architecture*, Maui, Haiwai, January 2001.

[4] L. Benini and G. De Micheli. System-level power optimization: techniques and tools. In *Proceedings of the International Symposium of Low Power Electronics and Design*, 1999.

[5] 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.

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

[7] 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.

[8] C.-H. Hsu, U. Kremer, and M. Hsiac. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In *International Symposium on Low Power Eletronics and Design*, August 2001.

[9] 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.

[10] 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.

[11] P. C. Laboratory. Parsec: A c-based simulation language. University of Califronia Los Angeles. http://pcl.cs.ucla.edu/projects/parsec.

[12] 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.

[13] T. Pering and R. Brodersen. Energy efficient voltage scheduling for real-time operating systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session*, June 1998.

[14] 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.

[15] 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.

[16] V. Raghunathan, P. Spanos, and M. Srivastava. Adaptive power-fidelity in energy aware wireless embedded systems. In *IEEE Real-Time Systems Symposium*, 2001.

[17] D. Shin and J. Kim. A profile-based energy-efficient intra-task voltage scheduling algori thm for hard real-time applications. In *International Symposium on Low Power Electronics and Design*, August 2001.

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

[19] 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.

[20] J. Tschanz, S. Narendra, Z. Chen, S. Borkar, M. Sachdev, and V. De. Comparative delay and energy of single edge-triggered & dual edge-tri ggered pulsed flip-flops for high-performance microprocessors. In *International Symposium on Low Power Electronics and Design*, August 2001.

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

[22] 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.

A Appendix

A.1 Periodic task set examples

The examples used in the experiments are given below. 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.