Energy Aware EDF Scheduling with Task Synchronization for Embedded Real Time Systems

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 #02-24
Aug 10, 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. For the Earliest Deadline First (EDF) scheduling, the problem of DVS in the presence of task synchronization has not yet been addressed. We compute slowdown factors for tasks which synchronize for access to shared resources. Tasks synchronize to enforce mutually exclusive access to these resources and can be blocked by lower priority tasks. We compute static slowdown factors for the tasks which guarantee meeting all the task deadlines. Our simulation experiments show on an average 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 or voltage scaling is more effective in 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. 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 the Earliest Deadline First (EDF) [13] scheduler. The tasks access shared resources in a mutually exclusive manner. Tasks need to synchronize to enforce mutual exclusion. We compute static slowdown factors in the presence of task synchronization to minimize the energy consumption of the system.

Shin et al. [22] have computed uniform slowdown factors for an independent 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 [6] observed that performing more iterations gives better slowdown factors for the individual task types. Yao, Demers and Shanker [25] 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 [17]. The analysis and correctness of the algorithm is based on an underlying EDF scheduler, which is an optimal scheduler [13]. An optimal schedule for tasks with different power consumption characteristics is considered by Aydin, Melhem and Mossé [1]. The same authors [2] have proven that the utilization factor is the optimal slowdown when the deadline is equal to the period. Quan and Hu [18] [19] discuss off-line algorithms for the case of fixed priority scheduling.

Since the worst case execution time (WCET) of a task is not usually reached, there is dynamic slack in the system. Pillai and Shin [16] 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 estimation heuristic [8] is studied by Kim et al.

All the above techniques assume the tasks to be independent in nature. Scheduling of task graphs on multiple processors has also been considered. Luo and Jha [14] 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 [7]. Zhang et al. [26] 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.

In real life applications, tasks access the shared resources in the system. Due to this task synchronization, tasks can be blocked for a shared resource. Jejurikar and Gupta [**?**] have addressed the computation of slowdown factors for task with task synchronization. The authors compute static slowdown factors for tasks scheduled by a Rate Monotonic Scheduler (RMS) [12]. In this paper, we consider the case of EDF scheduling on a uniprocessor system. We compute static slowdown factors in the presence of task synchronization. We gain as much as 40% to 60% energy savings over the known techniques.

The rest of the paper is organized as follows: Section 2 formulates the problem with a motivating example. In Section 3, we give the slowdown algorithms in the presence of task synchronization. The implementation and experimental results are given in Section 4. Section 5 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, and C_i is the WCET for the task, given it is the only task running in the system. The system has a set of shared resources. Access to the shared resources are mutually exclusive in nature and the accesses to the resources have to be serialized. Common synchronization primitives include semaphores, locks and monitors [23]. We assume that semaphores are used for task synchronization. All tasks are assumed to be preemptive, however the access to the shared resources need to be serialized. Due to the resource sharing, task can be *blocked* by lower priority tasks.

When a task has been granted access to a shared resource, it is said to be executing in its *critical section*. The k^{th} critical section of task τ_i is represented as $z_{i,k}$. Each task specifies the access to the shared resources and the worst case execution time of each critical section. With the specified information we can compute the maximum blocking time for a task. The blocking time for tasks depends upon the resource access protocol being used. Let B_i be the maximum blocking time for task τ_i under the given resource access protocol. We assume critical sections of a task are properly nested.

Each invocation of the task is called a *job* and the k^{th} invocation of task τ_i is denoted as $\tau_{i,k}$. 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 consumption is minimized. The processor speed can be varied to minimize energy usage. The *slowdown factor* at a given instance is the ratio of the scheduled speed to the maximum processor speed . 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.

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 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 normalize the speed to the maximum speed to have a continuous operating range of $[s_{min}, 1]$, where $s_{min} = S_{min}/S_{max}$.

2.3 Motivating example

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

$$\tau_1 = \{8, 8, 2\}, \tau_2 = \{15, 15, 7\} \tag{1}$$

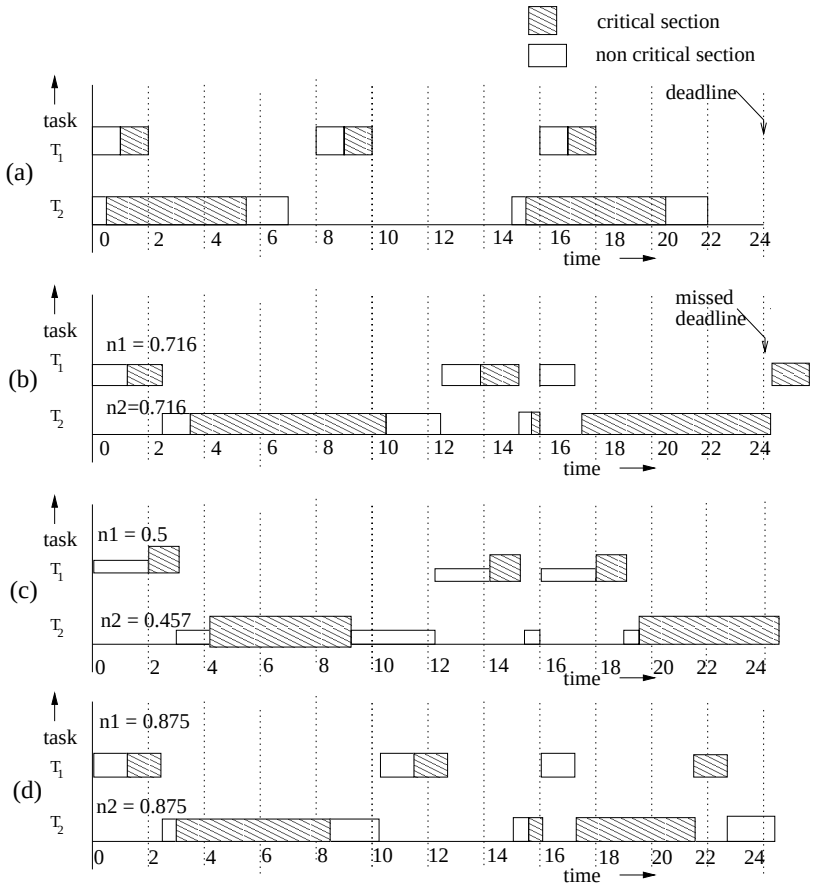

Figure 1. Motivation for Static slowdown techniques (a) Task arrival times and deadlines (period=deadline) with critical sections. (b) Processor Utilization as the static slowdown factor: $\eta = \frac{2}{8} + \frac{7}{15} = 0.716$, job $\tau_{1,3}$ misses deadline. (c) Slowdown of $\eta_1 = 0.5, \eta_2 = 0.457$ with critical section at maximum speed. (d) Uniform constant slowdown of $\eta = \frac{7}{8} = 0.875$, meets deadlines while observing blocking.

Both tasks access a shared resource through a semaphore *S*. The critical section for task τ_1 is $z_{1,1} = [1, 2]$ and that for τ_2 is $z_{2,1} = [0.5, 5.5]$. This task set is shown in Figure 1(a). The jobs for each task are shown at their arrival time with their workload. The jobs are to be scheduled on a single processor

by an Earliest Deadline First (EDF) scheduler. The task set is schedulable at full speed. We cannot compute slowdown factors ignoring the blocking factors. The processor utilization of the given task set is, $U = \frac{2}{8} + \frac{7}{15} = 0.716$, allowing for a uniform slowdown of $\eta = 0.716$. However job $\tau_{1,3}$ misses its deadline, as it is blocked by task $\tau_{2,2}$ for 6.67 time units. This is shown in Figure 1(b). Thus we need to consider the blocking times to compute the slowdown factors for the task.

We consider executing the critical sections at no slowdown and compute the slowdown for the task set. Task τ_1 can be blocked for 5 time units and has 1 unit of critical section, allowing its non critical section a slowdown of $\eta_1 = \frac{1}{8-(1+5)} = 0.5$. With $\eta_1 = 0.5$, task τ_2 can be slowed down by $\eta_2 = \frac{2/15}{1-(3/8+5/15)} = 0.457$. At this slowdown all tasks meet their deadline. This schedule is shown in Figure 1(c). Having a uniform slowdown for the entire task can be more energy efficient. Since task τ_1 can be blocked for up to 5 time units and $C_1 = 2$, a constant slowdown of $\eta = \frac{2}{8} + \frac{5}{8} = 0.875$ guarantees τ_1 meeting the deadlines. At this slowdown τ_2 also meets all deadlines and is shown in Figure 1(d).

We use the simplistic power model of $P = \eta^2$ to compare the energy consumption. We compute the energy consumed by both tasks. From Figure 1(d) energy consumed by both tasks is $E = 9.\frac{8}{7}(\frac{7}{8})^2 = 7.875$. The energy consumed from Figure 1(c) is $E = 7 + 1.\frac{2}{1}(\frac{1}{2})^2 + 2.\frac{1}{0.457}.(0.457)^2 = 8.414$. It is usually the case that the constant static slowdown is more energy efficient than running the critical section at full speed.

3 Static Slowdown Factors

We compute static slowdown factor for a system with an underlying Earliest Deadline First Scheduler. In this section, we give an algorithm to compute the static slowdown factors for tasks which share the resources in the system. We assume that the access to the shared resources is granted in mutual exclusion [23] by the use of semaphores [23]. The schedulability test of independent tasks is given by Liu and Layland [12]. With the deadline equal to period, a slowdown equal to the processor utilization factor is the optimal slowdown for the task set. EDF is an optimal scheduling policy when all tasks are independent of each other. However in real-life applications, tasks share the resources in the system. This could lead to tasks being blocked for a particular resource. Blocking of tasks can cause priority inversion [21] and result in deadline misses. The problem of scheduling tasks in the presenc of resource sharing is NP-hard [15, 5, 24]. Resource access protocols have been designed to bound the blocking times and *sufficient* schedulability tests have been given in the presence of maximum blocking times. Resource access protocols such as *priority inheritance protocol*, *priority ceiling protocol* and *priority limit protocol* [21] deal with the case of fixed priority scheduling. *Dynamic Priority Ceilings* [4], *Stack Resource Protocol* and *Minimal Stack Resource Protocol* [3] have been designed to handle tasks with dynamic priorities which encompasses EDF scheduling. Any resource management protocol can be used to manage the access to the resource. Let B_i be the *maximum blocking time* for task τ_i under the given resource access protocol.

3.1 EDF Scheduling

Let $\Gamma = \{\tau_1, \cdots, \tau_n\}$ be the tasks in the system ordered in non-decreasing order of their deadline. The task set is schedulable if the condition :

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

is satisfied. We give two methods to compute static slowdown factors for periodic task set. One method computes slowdown factors for the tasks with the critical sections being executed at maximum speed. The other method computes a constant slowdown for the entire periodic task set. The non-critical and critical sections of each task have a uniform slowdown factor.

3.2 Critical Section at Maximum Speed (CSMS)

We compute the static slowdown factors for the tasks with all critical sections being executed at full speed. We make a distinction between the critical and non-critical section of a task. Let C_i^{ncs} and C_i^{cs} be the non-critical section and critical section of task τ_i respectively $(C_i^{ncs} + C_i^{cs} = C_i)$. Using Equation 2, we compute static slowdown factors for all the tasks. Tasks are ordered in non-descending order of their deadline (i.e. in non-increasing order of their preemption level). We compute the slowdown factors in an iterative manner, from the higher to the lower priority tasks. An index q points to the latest task that has been assigned a slowdown factor. Initially, $q = 0$. Each of the task τ_i, $q < i \leq n$ has to be assigned a slowdown factor. Each task τ_i exactly meets its deadline if:

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

Note that the tasks τ_r, $1 \leq r \leq q$ have already been assigned a slowdown factor η_r. For the rest of the tasks we assume that they will use the same and yet to be computed slowdown factor, η_p as given in Equation 3. We compute a new slowdown factor for all tasks τ_p, $q < p \leq n$. There is a task with index m for which the slowdown factor is the largest among all other tasks: $\eta_m = max_p(\eta_p)$. Note that this is not necessarily the last task, n. Having the index m, all tasks between q and m can be slowed down by a factor equal to the slowdown factor of task $\tau_m = max_p(\eta_p)$. Thus, we assign the tasks τ_r, $q < r \leq m$, a slowdown factor of $\eta_r = max_p(\eta_p)$. The algorithm terminates when all tasks have been assigned a slowdown factor.

3.3 Constant Static Slowdown (CSS)

A constant slowdown for the processor is a desired feature. There is an overhead associated with changing power states and a constant slowdown eliminates this overhead. A constant slowdown is desired especially if the resource does not support run time change in the operating speed. Each task τ_i exactly meets its deadline if:

$$\frac{1}{\eta_i}(\frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{C_k}{D_k}) = 1 \tag{4}$$

A slowdown of $\eta = max_i(\eta_i)$ gives a constant static slowdown for all the tasks.

3.4 Examples

We compute the slowdown factors for the example in Section 2. The task set is $\tau_1 = \{8,8,2\}, \tau_2 = \{15,15,7\}$ and their blocking factors are $B_1 = 5$ and $B_2 = 0$.

We compute the uniform constant slowdown:

$$\eta_1 = \frac{2+5}{8} = \frac{7}{8} = 0.875 \text{ and}$$
$$\eta_2 = \frac{2}{8} + \frac{7+0}{15} = 0.716$$

This gives a constant static slowdown of $\eta = 0.875$.

The slowdown factors with critical sections at maximum speed are: Iteration 1,

$$\frac{1}{\eta_1}(\frac{1}{8}) + \frac{1+5}{8} = 1, \text{ gives } \eta_1 = 0.5 \text{ and}$$
$$\frac{1}{\eta_2}(\frac{1}{8} + \frac{2}{15}) + \frac{1}{8} + \frac{5}{15} = 1 \text{ gives } \eta_2 = 0.476$$

Thus we assign a slowdown of $\eta_1 = 0.5$ for the non-critical section.

Iteration 2,

$$\frac{1}{\eta_2}(\frac{2}{15}) + \frac{1}{0.5}(\frac{1}{8}) + \frac{1}{8} + \frac{5}{15} = 1, \text{ gives } \eta_2 = 0.457$$

This gives a slowdown of $\eta_1 = 0.5$ and $\eta_2 = 0.457$.

3.5 Computation time

Given the tasks are sorted in non-decreasing order of their deadline, the CSS algorithm has a linear time complexity. If we have to perform a sort on the task set it will increase the complexity to $O(n \log n)$. Each iteration of the CSMS algorithm has the same time complexity as that of the CSS algorithm. In practice, the number of iterations are only a few. Almost all examples have only one iteration giving a linear time complexity. But in the worst case there can be n iterations. Thus the worst case complexity of the CSMS algorithm is $O(n^2)$. Thus both the algorithms have polynomial time complexity.

4 Experimental Results

We have written a simulator in *parsec* [10], a C based discrete event simulation language. We have implemented the scheduler and the slowdown algorithms in this simulator. The simulator block diagram is shown in Figure 2. It consists of two main 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.

The RTOS is the heart of the simulator. It schedules the jobs on the resource(processor) and checks for deadline misses. The jobs access the shared resource by the resource access protocol. The static speed regulator changes the speed of the processor at run-time. 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.

We use the power model as given in [20] [9] to compute the energy usage of the system. The power P as a function of slowdown is given by

$$P = f(s) = 0.248 * s^3 + 0.225 * s^2 + 0.0256 * s +$$

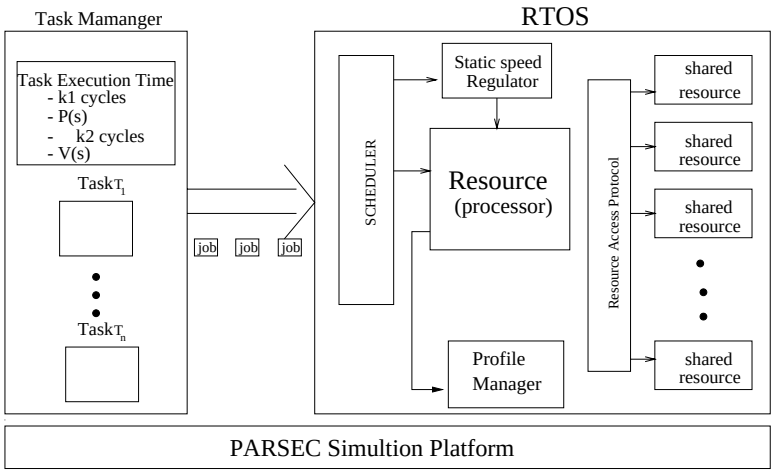

Figure 2. Generic simulator

$$\sqrt{311.16 * s^2 + 282.24 * s} * (0.0064 * s + 0.014112 * s^2) \tag{5}$$

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.

$$P_{switching} = C_{eff} V_{dd}^2 f \tag{6}$$

$$Delay = \frac{kV_{dd}}{(V_{dd} - V_{th})^2} \alpha \frac{1}{f} \tag{7}$$

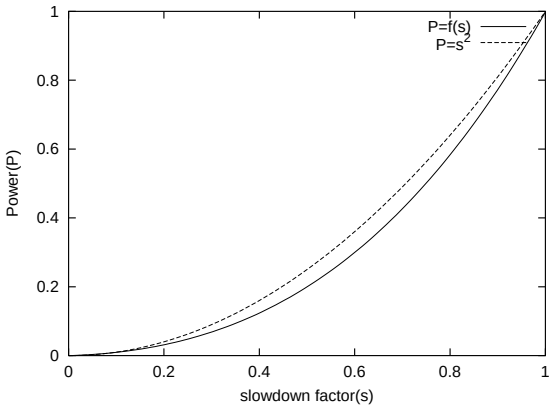

Figure 3. Power function $f(s)$ vs. s^2

The plot of the power function in shown in Figure 3. It is seen that it tracks s^2 closely. The switching capacitance and the relation between gate delay and the operating speed are used to accurately derive the power function.

4.1 Static slowdown

We compare the processor energy usage for the following techniques:

- **Critical Section at Maximum Speed (CSMS)**: The algorithm to compute the slowdown factors for each task is discussed in Section 3. The static factors are computed by performing EDF analysis with no slowdown for the critical sections. The case of $D < p$ is also considered.

- **Constant Static Slowdown (CSS)**: A constant static slowdown is computed for all the tasks including the critical sections. The algorithm is given in the Section 3.

  We compare the results of our algorithm to the static slowdown algorithm for independent tasks by Aydin et al. [2] which is the processor utilization factor. Since all tasks have the same slowdown, the blocking time will increase by the same factor and we guarantee deadlines. (If tasks have different slowdown factors, the blocking time can increase more than expected and lead to deadline misses.) We transform the task set to an independent task set.

- **Transformation I (T1)**: For each task τ_i, the execution time C_i is increased by its blocking time B_i. Since a task can experience a maximum blocking time of B_i, it is guaranteed to meet its deadline in the presence of blocking (provided all blocking tasks have the same or higher slowdown). The transformed task set is $\Gamma' = \{\tau_1', \cdots, \tau_n'\}$ where each task $\tau_i' =< T_i, D_i, (C_i + B_i) >$. The transformed tasks can be considered independent and we compute slowdown factors. A constant slowdown for all tasks guarantees deadlines.

- **Transformation II (T2)**: We add a new task called the blocking task τ_b in the system. Let $C_b = max_i(B_i)$ and $T_b = min_i(T_i)$, then the blocking task $\tau_b =< T_b, T_b, C_b >$. This task is assigned the highest priority task in the system. Given the sorted list of tasks in descending order of priority, τ_b is added at the head of the list. By adding task τ_b with highest preemption level, utilization factor of C_b will be added in the slowdown computation of each task τ_i. Satisfying the schedulability task for this transformed task set satisfies the schedulability test given in Equation 2. A constant slowdown for all tasks guarantee deadlines. Thus the computed slowdown factors will guarantee meeting all deadlines.

The above algorithms were used for three application sets given in the Prototyping Environment for Embedded Real Time Systems [11] (PERTS) software. The application sets are from various domains and comprise of *Flight Control System (FCS)*, *End to End Scheduling (EES)*, and *Multiple Resource Scheduling (MRS)*. A task set on multiple resources is converted to an equivalent task set by scaling the execution period.

Each system (example) has resources which are shared by the tasks in a mutually exclusive manner. We have used the Dynamic Priority Ceiling Protocol (DPCP) [4] to manage the resource accesses and have computed the maximum blocking time for each task under this protocol. The slowdown factors have been computed using the various algorithms and the task set is simulated for a time period equal to the hyper-period of the task set. The energy consumption is shown in Table 1. It is seen that the CSS algorithm performs better than the other algorithms in all the examples. It does better than the CSMS where a slowdown is computed for the non critical sections of all the tasks. A uniform slowdown is more energy efficient if an equal amount of slack is utilized (due to slowdown). The amount of slack utilized by the CSMS algorithm is not much greater than the the slack utilized by the CSS algorithm. So

Table 1. Energy Consumption

example	CSS	CSMS	T2	T1
FCS	2806.04	3056.12	3625.89	4362.43
EES	881.65	1258.93	945.71	1314.52
MRS	1410.81	2142.16	1697.05	2298.86

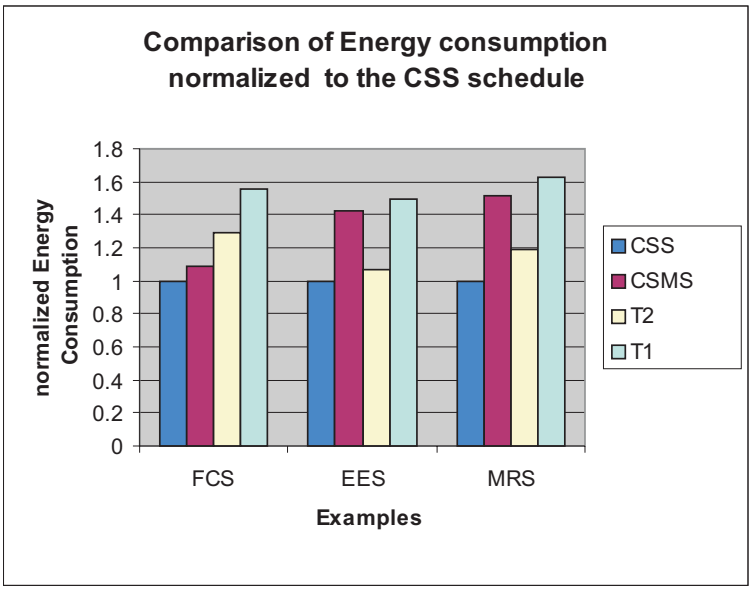

Figure 4. Normalized energy consumption for the slowdown methods

an uniform slowdown is more energy efficient. Figure 4 shows the energy consumption of each method normalized to the energy consumption of the CSS algorithm.

The slowdown factors computed by T1 are worse compared to CSS as the blocking factors are added to each task. This increases the processor utilization factor of each task proportional to the blocking factor. This adds up to an additional (unnecessary) blocking time in the analysis, leading to a higher (worse) slowdown factor. This results in a lot of slack in the system and T1 has the worst energy consumption. Energy consumption of T2 is closer to that of CSS. The workload of the blocking task in T2 is the maximum over the blocking factors of each task. Since the blocking task has the minimum period, it contributes to a considerable increases in utilization factor, leading to worse slowdown factors. In two of the three examples, energy consumption of CSMS is greater than that of T2. Running the critical section at full speed is usually not energy efficient.

5 Conclusions and Future Work

In this paper, we have given algorithms to compute static slowdown factor for a periodic task set. We take into consideration the effect of blocking that arises due to task synchronization. Experimental

results show that the computed slowdown factors save on an average 25%-30% energy over the known techniques. The algorithms have the polynomial time complexity. The techniques and very energy efficient and can be easily implemented in a RTOS. This will have a great impact on the energy utilization of portable and battery operated devices.

We plan to further exploit the static and dynamic slack in the system to make the system more energy efficient. As a future work, we plan to compute the slowdown factors when the processor supports discrete voltage levels. We will be implementing the techniques in a RTOS such as eCos and measure the power consumed on a real processor.

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] T. P. Baker. Stack-based scheduling of realtime processes. In *RealTime Systems Journal*, pages 67–99, 1991.

[4] M. Chen and K. Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. In *Real Time Systems Journal*, pages 325–346, 1990.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. W. H. Freeman & Co., 1979.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] R. T. S. Laboratory. Prototyping environment for real-time systems (perts). University of Illinois at Urbana Champaign (UIUC). http://pertsserver.cs.uiuc.edu/software/.

[12] 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.

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

[14] 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.

[15] A. K. Mok. *Fundamental Design Problems of Distributed Systems for Hard Real-Tima Environment*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts., 1983.

[16] 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.

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

[18] 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.

[19] 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.

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

[21] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. In *IEEE Transactions on Computers*, pages 1175–85, 1990.

[22] 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.

[23] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley and Sons, Inc., 2001.

[24] J. A. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. In *IEEE Transactions on Computers*, 1994.

[25] 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.

[26] 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

We used the following examples in our experiments. They were used in the PERTS tool at UIUC.

- Fight Control System
- Multi Processor System
- End to End Scheduling

Some of the examples used multiple processors. We scaled the execution periods to map them to a single processor system. The examples are given on the next page.

A.1 Task Description Format (TDF)

We have defined a Task Description Format (TDF) to describe the tasks. We can define the semaphore P and V opeations and the critical sections can be specified. It is used to specify all the task properties. The format is intutive and easily readable. All the examples are given in TDF.