Dual-Mode Frequency Inheritance Algorithm for Energy Aware Task Scheduling with Task Synchronization

Ravindra Jejurikar Cristiano Pereira 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,cpereira,rgupta}@ics.uci.edu

CECS Technical Report #03-07

Feb 28, 2003

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. In this paper, we allow tasks to have different slowdown factors based on the task characteristics. We introduce the idea of frequency inheritance which is required to guarantee the task deadlines. We present the Dual Mode Frequency Inheritance (DMFI) algorithm under the Earliest Deadline First (EDF) scheduling policy. The two modes of operation are the independent mode and the synchronization mode. We prove that it is sufficient to execute in the synchronization mode for a shorter interval than that in the previous work, resulting in more energy savings. We formulate the problem of computing the slowdown factors for tasks as an optimization problem to minimize the total energy consumption of the system. Our simulation experiments show on an average 10% 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 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 compute a voltage schedule for a periodic task set to minimize energy usuage.

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 scheduled on a single processor system based on EDF scheduling. Tasks synchronize to enforce mutual exclusive access to the shared resources. We compute static slowdown factors in the presence of task synchronization to minimize the energy consumption of the system.

Most of the earlier work deals with independent task sets. Shin et al. [7] have computed uniform slowdown factors for an independent periodic task set. Yao, Demers and Shanker [9] presented an optimal off-line speed schedule for a set of N jobs. An optimal schedule for tasks with different power consumption characteristics is considered by Aydin, Melhem and Mossé [1].

Scheduling of task graphs on multiple processors has also been considered. Luo and Jha [4] have considered scheduling of periodic and aperiodic task graphs in a distributed system. Zhang et al. [11] 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, a higher priority task can be blocked for a shared resource and miss its deadline due to priority inversion. Low power scheduling in the presence of task synchronization [3] and non-preemptive sections [10] has been considered. Previous work assumes that all tasks have a constant static slowdown factor. In this paper, we present a Dual Mode frequency inheritance algorithm which allows tasks to have different slowdown factors. We compute the static slowdown factors by formulating the problem as a convex minimization problem. We gain as much as 15% energy savings over the known techniques.

The rest of the paper is organized as follows: Section 2 formulates the problem and in Section 3, we present the dual mode frequency inheritance algorithms for EDF scheduling In Section 4, we formulate the computation of slowdown factors as an optimization problem. The experimental results are given in Section 5 and Section 6 concludes the paper with future directions.

2 Preliminaries

In this section, we introduce the necessary notation and formulate the problem.

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 $< T_i, D_i, C_i >$ 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 that it is the only task running in the system. A preemption level $\pi(\tau)$ is associated with each task τ. The essential property of preemption levels is that if a job τ_h preempts a job τ_l then $\pi(\tau_h) > \pi(\tau_l)$. The relative deadline of a job can be used to define the preemption levels [2]. Each invocation of the task is called a *job*. A priority function $P(J)$ is associated with each job such that if job J has a higher priority than J' then $P(J) > P(J'$. Under EDF scheduling, shorter the absolute deadline higher the priority.

The system has a set of shared resources. Access to the shared resources are mutually exclusive in nature. Due to the resource sharing, a task can be *blocked* by lower priority tasks. Let B_i be the maximum blocking time for task τ_i under the given resource access protocol. We assume that semaphores are used for task synchronization. When a task has been granted access to a shared resource, it is said to be executing in its *critical section* We assume critical sections of a task are properly nested [6]. A critical section is called a *blocking section* is a higher priority job is blocked for the completion of the critical section. If no higher priority job is blocked when a job is executing, the job is said to be executing in its *non-blocking section*.

The tasks are scheduled on a single processor which supports variable frequency and voltage levels. All tasks are assumed to be preemptive, however access to the shared resources need to be serialized. The processor speed can be varied to minimize energy usage. 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 *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 that the speed of the processor can be varied over a continuous range. In this paper, we ignore the time and energy overhead incurred in changing the processor speed.

2.2 Variable Speed Processors

A wide range of processors 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 assume that the frequency can be varied continuously from the minimum frequency 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 $\eta_{min} = f_{min}/f_{max}$. We also assume that the voltage be continuously within its operating voltage range.

3 Static Slowdown Factors

We compute static slowdown factor for a system with an underlying Earliest Deadline First scheduler. The problem of scheduling tasks in the presence of resource sharing is NP-hard [8]. 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. The *Stack Resource Protocol* [2] has been designed

to handle tasks scheduled under EDF policy. Our work is based on the use of Stack Resource protocol to manage the access to the resources. Let B_i be the *maximum blocking time* for task τ_i based on the SRP.

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 :

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

is satisfied. This is a *sufficient* schedulability test.

Independent Mode: Assuming the task are independent (in the absence of blocking), let η_i^I be the slowdown factors for task τ_i.

Theorem 1 *Given n independent periodic tasks, the task set is feasible at a slowdown factor of η_i^I for task τ_i if,*

$$\sum_{i=1}^{n} \frac{1}{\eta_i^I} \frac{C_i}{D_i} \leq 1 \tag{2}$$

The task set is schedulable under EDF scheduling if its density if less than 1 (Appendix **??**) This is a sufficient schedulability test.

Synchronization Mode: Let B_i be the maximum blocking encountered by each task τ_i, under a given resource access protocol and let η_i^S be the slowdown factors in the presence of blocking.

Theorem 2 *A task set of n periodic tasks sorted in non-decreasing order of their relative deadlines is feasible at a slowdown factor of η_i^S for each task τ_i if :*

$$\overset{\forall i}{i = 1, ..., n} \quad \frac{1}{\eta_i^S} \frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{1}{\eta_k^S} \frac{C_k}{D_k} \leq 1 \tag{3}$$

and the critical section of every blocking task, τ_b, inherits a slowdown of $max_{j=m}^{b} \eta_j^S$, where τ_m is the blocked job with the highest preemption level.

The property of frequency inheritance is essential to guarantee all deadlines. The details of the proof are present in Appendix **??** (Inheriting a slowdown of η_m^S, the slowdown of task τ_m does not suffice (Appendix B)

Blocking slowdown factor η_m^B: We define the *blocking slowdown factor* $\eta_m^B(b)$ for task τ_m, as the slowdown factor that the blocking critical section of task τ_b must inherit to guarantee deadlines of the higher priority tasks. As given by Theorem 2, $\eta_m^B(b) = max_{j=m}^{b} \eta_j^S$. Since the blocking slowdown factor of a task depends on the blocking task, a task may have to maintain blocking slowdown factors for each blocking task. We would like to have only one blocking slowdown factor per task and we use the maximum of all blocking slowdown factors for a task. This maximum is called the*blocking slowdown factor*, η_m^B, and is computed as the maximum synchronization slowdown of all the lower or equal priority tasks.

$$\eta_m^B = max_{j=m}^{n-1} \eta_j^S \tag{4}$$

3.2 Dual Mode Algorithm

We propose a *dual mode* algorithm for energy aware scheduling with synchronization. The two modes of operation are *independent mode* and *synchronization mode*. The DMFI algorithm is based on the use of Stack Resource Protocol (SRP). The slowdown factors of η_i^I and η_i^S are used by task τ_i in the independent and synchronization mode respectively. The system starts in the independent mode. When a new job arrives in the system, the protocol checks if this job is blocked. If a lower priority job J_b blocks the new job in the independent mode, then the system enters the synchronization mode. In the synchronization mode all jobs execute at a slowdown of η_i^S. If the executing job is blocking other jobs, the critical section (of the blocking job) *inherits* the *blocking slowdown factor* of the blocked task with the maximum preemption level. All tasks execute in synchronization mode until the job J_b executes a non-blocking section. If the job J_b terminates at its outermost blocking critical section, tasks execute in the synchronization mode until a task with lower priority than J_b executes or when the system becomes idle. The above is implemented as follows. When the system enters the synchronization mode, the priority of the blocking job $\mathcal{P}(J_b)$ is marked as shown in $line\ (5)$ of the algorithm. The system changes back to independent mode, when it executes *a non-blocking section* of a lower or equal priority task. The dual mode frequency inheritance algorithm is given in Figure 1 The function *inheritSpeed()* inherits the blocking slowdown factor of the maximum preemption level blocked task. *setSpeed()* sets the CPU speed to the specified speed. We say a job executed in synchronization mode if it begins execution in the synchronization mode. Such a job executes to completion in the synchronization mode. Otherwise we say that the job executed in independent mode. Note that some critical sections of this job can execute in the synchronization mode. Since the slowdown in independent mode η_i^I is smaller than or equal to the slowdown in synchronization mode η_i^S, this leads to energy savings.

3.2.1 Motivating example

We give an example to show that different tasks can have different slowdown factor. Consider a simple real time system with 3 periodic tasks having the following parameters: $\tau_1 = \{5, 5, 2\}, \tau_2 = \{15, 15, 3\}, \tau_3 = \{20, 20, 4\}$ If $B_1 = 3$, $B_2 = 1$ and B_3=0, then the tasks can have slowdown factors of $\eta_1^I = 0.8$ $\eta_1^S = 1.0$ $\eta_2^I = 0.8$ $\eta_2^S = 0.8$ $\eta_3^I = 0.8$ $\eta_3^S = 0.8$. With these slowdown factors, task τ_1 executes at different speeds depending on the system mode. However task τ_2 and τ_3 can execute at the same speed in both modes. We compare the execution of this task set the Dual Speed algorithm [10]. The dual speed sets the high speed to $H = 1$ and the low speed to $L = 0.8$. Under the dual speed algorithm, task τ_2 executes at a speed of 1.0 in the synchronization mode, however it suffices to execute it at a slowdown of 0.8. Thus having different slowdown factors for the individual tasks can be more energy efficient.

Theorem 3 *A task set of n periodic tasks sorted in non-decreasing order of their relative deadlines can be feasibily scheduled with the Dual Mode Frequency Inheritance algorithm with slowdown factors of η_i^I and η_i^S for each task τ_i if :*

$$\sum_{i=1}^{n} \frac{1}{\eta_i^I} \frac{C_i}{D_i} \leq 1 \tag{5}$$

On arrival of a new job J_i :

(1) **if** (J_i is Blocked)
(2) Let J_b be the job blocking J_i
(3) **if** $(mode == INDEP)$
(4) $mode = SYNC;$
(5) $MarkedPrio = \mathcal{P}(J_b);$
(6) **endif**
(7) **endif**

On execution of each job J_i :

(1) **if** (Jobs blocked on J_i)
(2) setSpeed (inheritSpeed(J_i));
(3) **else**
(4) **if** $(mode == SYNC$ **and** $\mathcal{P}(J_i) \leq MarkedPrio)$
(5) $mode = INDEP;$
(6) $MarkedPrio = -\infty;$
(7) **endif**
(8) **if** $(mode == INDEP)$ setSpeed (η_i^I);
(9) **else** setSpeed (η_i^S); **endif**
(10) **endif**

Figure 1. Dual Mode Frequency Inheritance (DMFI) Algorithm under EDF scheduling

$$\overset{\forall i}{i = 1, ..., n} \quad \frac{1}{\eta_i^S}\frac{B_i}{D_i} + \sum_{k=1}^{i}\frac{1}{\eta_k^S}\frac{C_k}{D_k} \leq 1 \tag{6}$$

$$\forall i \quad \eta_i^S \geq \eta_i^I \tag{7}$$

Proof: We prove the above by contradiction. Suppose the claim is false and a task instance misses its deadline. Let t be the first time that a job misses its deadline. let t' be the the latest time before t such that there are no pending jobs with arrival times before t' and deadlines less than or equal to t. Since no requests can arrive before system start time ($time = 0$), t' is well defined. Let $\mathcal{A}$ be the set of jobs that arrive in $[t, t']$ and have deadlines in $[t, t']$. There exists a k such that $\mathcal{A} \subseteq \{\tau_1, ...\tau_k\}$. By choice of t', there are pending requests of jobs in $\mathcal{A}$ at all times during the interval $[t', t]$. Thus the system in never idle in the interval. By the EDF priority assignment, the only jobs that are allowed to *start* in $[t, t']$ are in $\mathcal{A}$. If a job not present in $\mathcal{A}$ executes in $[t, t']$, it must be holding some resource allocation at time t' that is blocking a job in $\mathcal{A}$. The Stack Resource Policy guarantees at most one job not in $\mathcal{A}$ can block the jobs in $\mathcal{A}$. The blocking job can execute for as long as it takes to exit its outermost nontrivial critical section. Note that the blocking job may execute more than once if it has more that one non-preemptable resource simultaneously. However the maximum execution time of τ_b in this interval is its outermost critical section.

We consider both cases with and without the blocking job.

Case I: Only jobs in $\mathcal{A}$ are executed during $[t, t']$. Let $X = t - t'$. Since all the jobs are periodic in nature and the jobs in $\mathcal{A}$ arrive at or after t', the number of executions of each task τ_i in $\mathcal{A}$ in the interval X is bounded by $\lfloor\frac{X-D_i}{T_i}\rfloor + 1$. Since the slowdown factors satisfy Equation 7, the slowdown of each task is at least η_i^I. Since a task misses its deadline at time t, the execution time for the jobs in $\mathcal{A}$ exceeds the interval length X. Therefore,

$$\sum_{i=1}^{k}\frac{1}{\eta_i^I}(\lfloor\frac{X - D_i}{T_i}\rfloor + 1)C_i > X$$

Since $\frac{X}{T_i} \geq \lfloor\frac{X}{T_i}\rfloor$, we have

$$\sum_{i=1}^{k}\frac{1}{\eta_i^I}(\frac{X - D_i + T_i}{XT_i})C_i = \sum_{i=1}^{k}\frac{1}{\eta_i^I}(1 + \frac{T_i - D_i}{X})\frac{C_i}{T_i} > 1$$

Since $D_i \leq X$, $\forall\ i = 1, ..., k$, it implies

$$\sum_{i=1}^{k}\frac{1}{\eta_i^I}(1 + \frac{T_i - D_i}{D_i})\frac{C_i}{T_i} = \sum_{i=1}^{k}\frac{1}{\eta_i^I}\frac{C_i}{D_i} > 1$$

which contradicts with Equation 5.

Case II: Let J_b be the job blocking a job in $\mathcal{A}$. J_b executes at time t' with deadline greater than t and $X < D_b$. The processor demand during $[t', t]$ is larger than that in the first case due to the execution of J_b at time t'. If $\mathcal{A} \subseteq \{\tau_1, ...\tau_k\}$, then $D_k < X$ and $k < b$. The total length of the time that J_b executes in $[t, t']$ is bounded by its outermost critical section. In particular, the maximum execution time of J_b in $[t, t']$ at full speed is bounded by B_k [2]. For every blocked task τ_i, the blocking critical section inherits a frequency of $max_{j=b}^{i}\eta_j^S$. Thus the blocking time is bounded by $\frac{1}{\eta_k^S}B_k$. Only the outermost critical section of J_b and the tasks in $\mathcal{A}$ execute in the interval $[t, t']$. All tasks in the interval execute in the

synchronization mode at a slowdown of η_i^S for task $\tau_i \in A$. The total execution time of these jobs is given by $\frac{1}{\eta_k^S}B_k + \sum_{i=1}^{k}\frac{1}{\eta_i^S}(\lfloor\frac{X-D_i}{T_i}\rfloor + 1)C_i$ Since a task misses its deadline at time t, the execution time for the jobs in $\mathcal{A}$ and the outermost critical section of J_b exceeds X, the length of the interval. Therefore,

$$\frac{1}{\eta_k^S}B_k + \sum_{i=1}^{k}\frac{1}{\eta_i^S}(\lfloor\frac{X-D_i}{T_i}\rfloor + 1)C_i > X$$

Since $\frac{X}{T_i} \geq \lfloor\frac{X}{T_i}\rfloor$, we have

$$\frac{1}{\eta_k^S}\frac{B_k}{X} + \sum_{i=1}^{k}\frac{1}{\eta_i^S}(\frac{X-D_i+T_i}{T_iX})C_i > 1$$

$$= \frac{1}{\eta_k^S}\frac{B_k}{X} + \sum_{i=1}^{k}\frac{1}{\eta_i^S}(1+\frac{T_i-D_i}{X})\frac{C_i}{T_i} > 1$$

$D_i \leq X\ \forall i = 1, ..., k$ it implies ,

$$\frac{1}{\eta_k^S}\frac{B_k}{D_k} + \sum_{i=1}^{k}\frac{1}{\eta_i^S}(1+\frac{T_i-D_i}{D_i})\frac{C_i}{T_i} = \frac{1}{\eta_k^S}\frac{B_k}{D_k} + \sum_{i=1}^{k}\frac{1}{\eta_i^S}\frac{C_i}{D_i} > 1$$

which contradicts with Equation 6.

3.3 Modified Stack Resource Protocol

The Stack resource protocol (SRP) [2] is commonly used to manage the shared resources. SRP minimizes the blocking time of each task to at most one critical section. We use the same definitions of *resource ceiling* $\lceil R\rceil$ and the *current system ceiling* $\overline{\pi}$ as given in the SRP [2]. Under SRP, the preemption test for a job J is $\pi(J) > \overline{\pi}$ and a job blocks if $\pi(J) \leq \overline{\pi}$. Let s_m be the semaphore having the maximum resource ceiling and let J_{s_m} be the job holding the semaphore s_m. The preemption level of a new job $\pi(J)$ can be lower than the system ceiling in two cases: (1) $\pi(J) \leq \pi(J_{s_m})$: a lower priority job J arriving in the system. (J is not blocked) (2) $\overline{\pi} \geq \pi(J) > \pi(J_{s_m})$: A job J with a preemption level higher than than of J_{s_m} but not higher than the current system ceiling $\overline{\pi}$. Job J is blocked if it has a higher priority than J_{s_m}. When a job has access to a semaphore (critical section), the protocol does not differentiate between a lower priority job and a blocked job. We need to distinguish between the two cases since the mode is changed to synchronization mode when a task is blocked in the independent mode.

We present a modified *blocking test* which checks for a lower priority job holding a resource. A job J is blocked if a lower priority job is holding a semaphore s and the current resource ceiling of s is higher than or equal to the preemption level of job J_s. (If a resource is free, its resource ceiling is lower than the preemption level of every job) Thus we check if there exists a resource s which satisfies:

$$\pi(J) \leq \lceil s\rceil \textbf{ and } \mathcal{P}(J) > \mathcal{P}(J_s)$$

If such a semaphore exists then the job J is blocked and added to the blocked queue. If not, it is added to the ready queue. This modified blocking test is important to detect if a job is truly blocked, since this may result in changing to a higher power state. A blocked task is unblocked when the system ceiling drops below the preemption level of a task. The modification presented in this section retains all the properties [2] of the basic SRP.

4 Computing Slowdown Factors

Computation of slowdown factors when tasks have uniform power characteristics is considered in [3] and [10]. Tasks have different power characteristics [1] and computing slowdown factors taking the task power characteristics minimizes the system energy. We formulate the problem as an optimization problem to compute the slowdown factors for the tasks. We describe the power delay characteristics that are used in formulating the optimization problem.

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}.

$$P_{switching} = C_{eff} V_{DD}^2 f \tag{8}$$

$$Cycle\ Time\ (CT)\ \alpha \frac{1}{f} = k \frac{V_{DD}}{(V_{DD} - V_{TH})^\alpha} \tag{9}$$

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. The slowdown factor is the inverse of the cycle time and $\eta = 1/CT$

4.2 Convex Minimization Problem

We formulate the energy minimization problem as an optimization problem. We normalize the voltage and slowdown factors to the maximum operating speed. We compute normalized voltage levels for the tasks such that the conditions in Theorem 3 are satisfied. Let $v \in \mathbb{R}^{2n}$ be a vector representing the normalized voltages V_i^S and V_i^I of task τ_i. V_i^S represents the task voltage in the synchronization mode and V_i^I represents the task voltage in independent mode. The optimization problem is to compute the optimal vector $v^* \in \mathbb{R}^{2n}$ such that the system is feasible and the total energy consumption of the system is minimized. Let $f_i(V)$ be the normalized energy consumption of task τ_i as a function of the normalized voltage V. Since part of these tasks execute in synchronization mode and the rest in independent mode, the total energy consumption depends on the fraction of the tasks that execute in synchronization mode. Let δ_i^S be the fraction of the total number of task instances of task τ_i that execute in synchronization mode, then $(\delta_i^I = 1 - \delta_i^S)$ is the fraction of the task instances executed in independent mode. The total energy consumption of the system E, is a function of the voltage vector, $v \in \mathbb{R}^{2n}$ and is given by Equation 10. The optimization problem is a convex minimization problem:

minimize :

$$E(v) = \sum_{i=1}^{n} \delta_i^I \cdot f_i(V_i^I) \cdot \frac{C_i}{T_i} + \sum_{i=1}^{n} \delta_i^S \cdot f_i(V_i^S) \cdot \frac{C_i}{T_i} \tag{10}$$

with $f_i(V) = V^2$ for the uniform power characteristics case.

under the constraints :

$$\sum_{i=1}^{n} \frac{1}{\eta_i^I} \frac{C_i}{D_i} \leq 1 \tag{11}$$

$$\overset{\forall i}{i = 1, ..., n} \quad \frac{1}{\eta_i^S} \frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{1}{\eta_k^S} \frac{C_k}{D_k} \leq 1 \tag{12}$$

$$\forall i \quad \eta_{min} \leq \eta_i^I \leq \eta_i^S \leq 1 \tag{13}$$

where η is a function of V given by $\eta = 1/CT$ Equation 11 constraints the utilization of the system when the tasks are independent. Equation 12 enforces the feasibility of the task set in the presence of task synchronization. Equation 13 constraints the slowdown in the synchronization mode to be greater than or equal to the slowdown in the independent mode. The normalized slowdown factors are between the normalized minimum frequency η_{min} and 1. The cycle time at voltage V_i is given in Equation 9. The constraint given by Equations 11, 12, and 13 are convex (Appendix A) The optimization function depends on the power characteristics $f_i(V)$ of the task. For all convex power characteristics the optimization function is convex. Thus we have a convex minimization problem.

The number of tasks executed in each mode depends on the task slowdown factors. We do not know the initial value of δ_i^S to be used in the optimization function. Initially we assume $\delta_i^S = 0.05$ and compute slowdown factors. We simulated the task set at the computed slowdown factors to get δ_i^S values from the simulation. The updated δ_i^S values did not change the task slowdown factors. In our experiments we assume $\delta_i^S = 0.05$ in the optimization function. We could use a iterative loop of computation of slowdown factors and updating the δ_i^S values obtained from simulation. We also ignore the energy overhead due to frequency inherits in the optimization function.

Given there are n tasks in the system, the number of variables are $2n$ and the number of constraints are $2n + 1$. Thus the number of variables and constraints are linear in the problem size.

5 Experimental Setup

Simulation experiments were performed to evaluate our proposed technique with task sets of 10-15 tasks. We used a mixed workload with task periods belonging to one of the three ranges [2000,5000], [500,2000] and [90,200]. The WCET's for the three ranges were [10,500], [10,100] and [10,20] respectively. The tasks were uniformly distributed in these categories with the period and WCET of a task randomly selected within the corresponding ranges. The number of semaphores (within [0,2]) and the position of the critical sections within each task were selected randomly. The length of the critical sections were chosen to be $CSperc * WCET$, where $CSperc$ is the size of the critical section as a percentage of the WCET. We vary $CSperc$ up to 30% of the WCET in steps of 3%.

Due to the diverse nature of the tasks in a real system, tasks can have distinct power characteristics [1] and it is energy efficient to have different slowdown factors for the tasks. For experimental results we restrict power characteristics to be *linear*. However the problem formulation in Section 4 works for all convex differential power characteristics. We consider the following distributions similar to the ones presented by Aydin et. al [1].

Identical Distribution: where all tasks have the same power function coefficient.

Bimodal Distribution: represents the case where there are two types of tasks in the system, with 50% having a low power function coefficient of 1 and the others having a high power function coefficient k.

Uniform Distribution: where the coefficients of the power function of the tasks are uniformly distributed between 1 and k.

We vary k in the range $[1, 8]$ for experimental results. The energy and delay characteristics are given by Equations 8 and 9. We have used an operating voltage range of $0.6V$ and $1.8V$. The threshold voltage is assumed to be $0.36V$ and $\alpha = 1.5$. We compare the energy gains of the *Dual Mode Frequency Inheritance (DMFI)* Algorithm over the Dual Speed(DS) Algorithm [10]. The dual speed algorithm has

two speeds of H and L corresponding to the synchronization and independent mode respectively. We show the average gains of DMFI over DS taken over 5-10 different task sets.

5.1 Experimental Results

In the first set of experiments we have identical power characteristics and vary the blocking percentage. Figure 3 shows the percentage gains of DMFI algorithm over the DS algorithm. It is seen that the DMFI algorithm performs better than the DS algorithm as blocking increases. Blocking is not significant up to 15% and the tasks are feasible at a slowdown equal to the system utilization which makes it perform no better than DS. Both algorithms have identical slowdown factors and power consumption is identical. With increased blocking, DMFI uses different slowdown factors for the task to have energy gains. To see the effects of task utilization on the power consumption, we vary it from 90% to 50%. As utilization decreases we have more room for obtaining better slowdown factors and this results in energy gains upto 70% utilization. As utilization drops further both algorithms have the same performance.

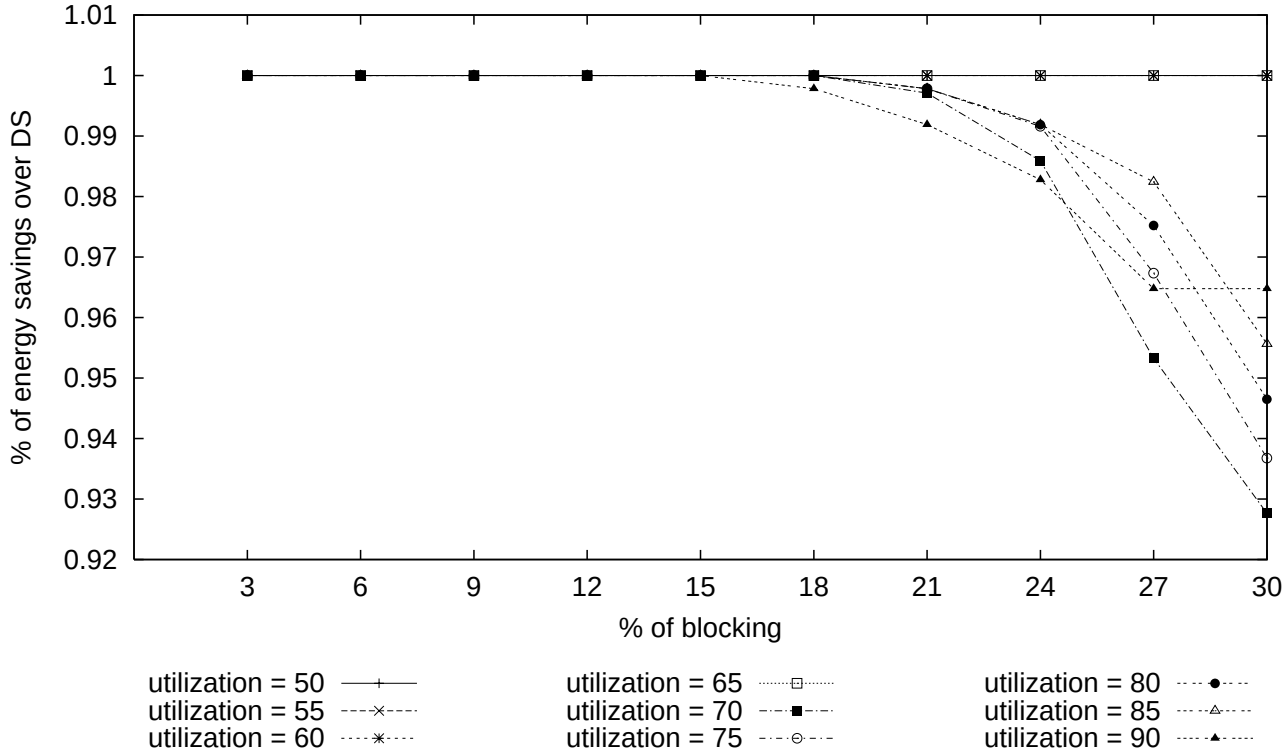

Figure 2. Average percentage Energy savings of DMFI algorithm over the DS algorithm for Identical distribution:

We explain the side-effects of having different slowdown factors for task. It is beneficial to have different slowdown factors based on the power characteristics of the tasks. However due to slowdown inheritance, a task may inherit a slowdown factor higher than the its slowdown factor.Thus executing parts of the tasks at high speed and parts at low speed consumes more energy that a constant slowdown. This leads to an additional overhead which we call the *inheritance overhead*. We can even consume more energy if the inheritance overhead overcomes the energy savings due to the different slowdown

factors. The inheritance overhead is not a part of the optimization function. However is not clear how the optimization function can take this overhead into account. It is not clear which job instances will inherit a higher slowdown, what slowdown factor will be inherited and how long the will inherit.

We explain the above behavior using Figure 3 which is one instance of a task set of our experiments with bimodal distribution with $k = 5$. We execute the tasks at the H speed of the dual speed algorithm to compare the energy gains of DMFI over DS. We call this the *H-speed algorithm (HS)*. Figure 3 shows the gains of the DMFI and DS to the HS algorithm. The thick-dotted line represents the energy gains of DMFI over DS. It is seen that the DMFI performs better than the DS, however the percentage improvement decrease from 3% to 18% blocking due to the inheritance overhead. However as blocking increases the different slowdown factors result in more energy savings than the DS algorithm.

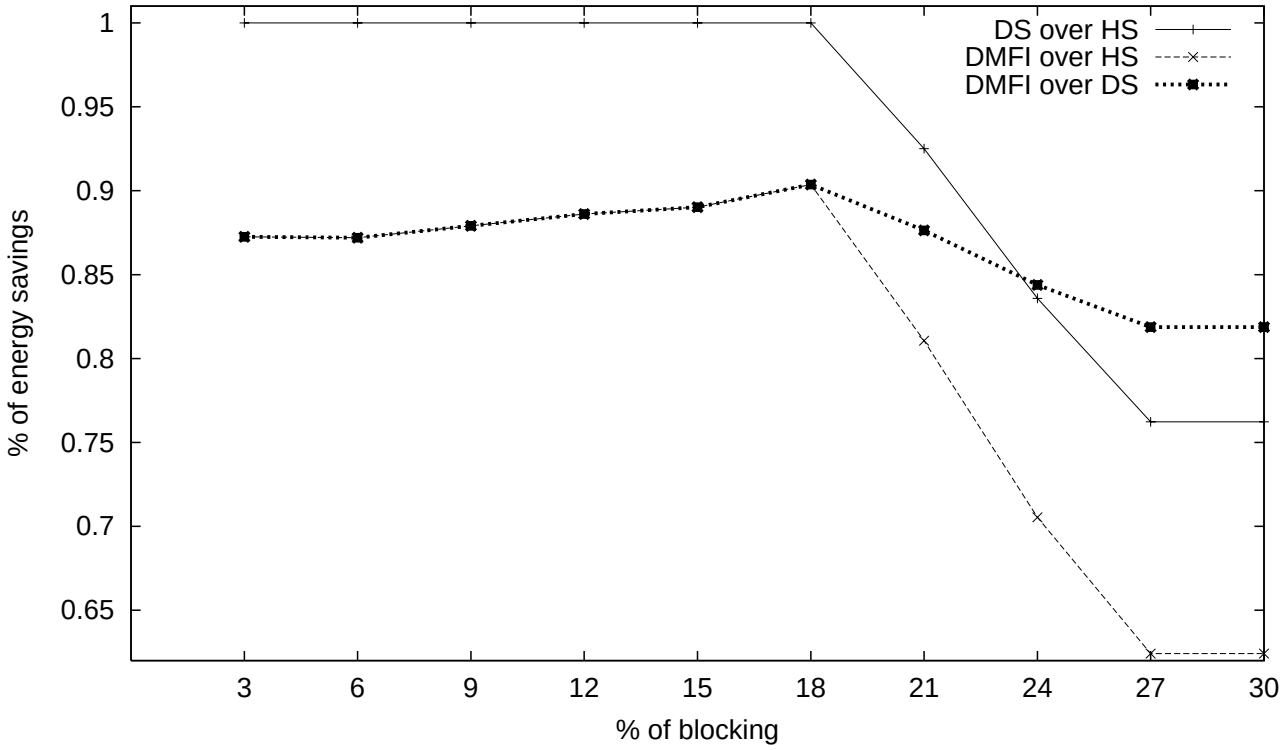

Figure 3. Percentage gains of DMFI and DS over HS and the percentage gains of the DMFI over DS algorithm for a particular task set with bimodal distribution $(k = 5)$.

Figure 4 shows the second set of experimental results where we vary the task power characteristics according to the bimodal distribution along with the blocking percentage. It is seen that as the power coefficient k increases the energy gains increase. For $k = 2$ and $CSperc$ 3 and 6, DMFI consumes slightly more energy due to inheritance overhead. The percentage gains over DS do not steadily increase with blocking percentage due to the inheritance overhead.

Figure 5 shows the energy gains for the uniform distribution case. The energy overhead due to inheritance decreases the percentage gains. As seem in the figure, the percentage gains decrease from 9% to 18%. The gains increase from 3% to 9% and from 18% to 30%. As seen in the figure, there are a few

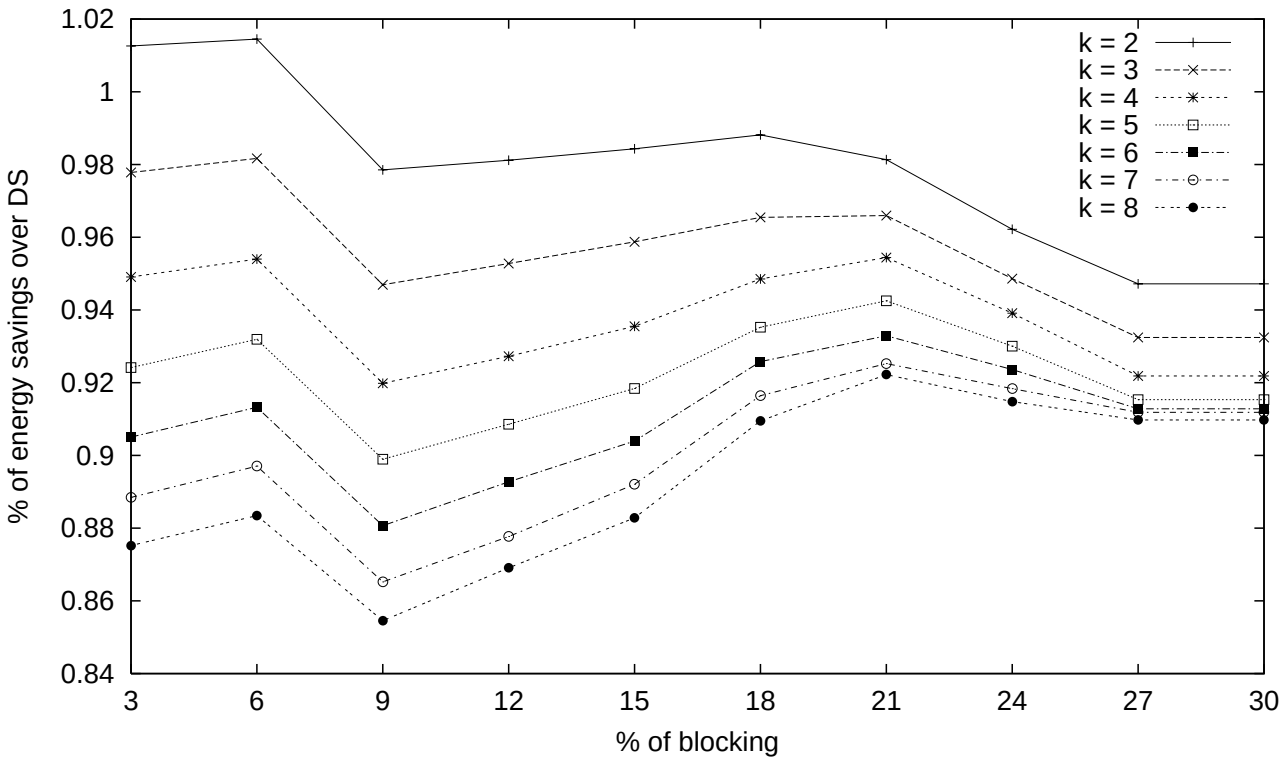

Figure 4. Average percentage gains of DMFI algorithm over DS algorithm for Bimodal distribution

points (e.g $k = 6$ and $CSperc = 18$) where DMFI performs slightly worse than the DS. This is also due to inheritance overhead as explained earlier.

6 Conclusions and Future Work

In this paper, we present slowdown algorithms in the presence of task synchronization. Our framework allows the tasks to have different slowdown factors depending on the task characteristics. Similar to priority inheritance, we introduce the notion of *frequency inheritance* to guarantee task deadlines. Each task has a precomputed *blocking slowdown factor* which is inherited by the critical sections of blocking tasks.

We present the Dual Mode Frequency Inheritance algorithm under the EDF scheduling policy. For the independent mode, slowdown factors are computed assuming the tasks are independent and for the synchronization mode the computation of slowdown factors take the blocking time into account. We prove that it is sufficient to execute in the synchronization mode for a shorter interval than that presented in the previous work [10] to further reduce the energy usuage. Our work builds upon the Stack Resource Protocol (SRP) and the blocking test under SRP is modified to avoid unnecessary transitions to the synchronization mode. We formulate the computation of slowdown factors for tasks with different power characteristics as an optimization problem. Experimental results show that the computed slowdown factors save on an average 10% energy over the known techniques. The techniques are energy efficient and can be easily implemented in an RTOS. This will have a great impact on the energy utilization of

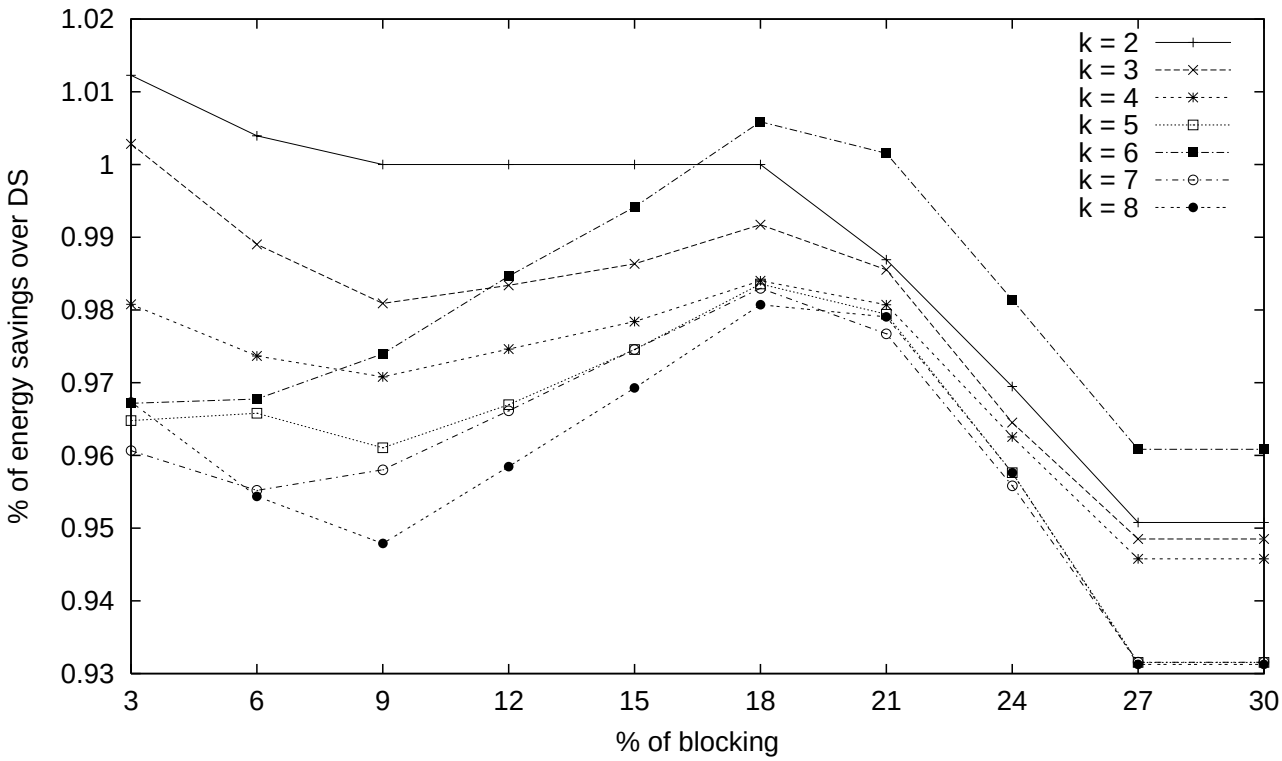

Figure 5. Average percentage gains of DMFI algorithm over DM algorithm for Uniform distribution

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.

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*, June 2001.

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

[3] R. Jejurikar and R. Gupta. Energy aware edf scheduling with task synchronization for embedded real time operating systems. In *COLP*, 2002.

[4] J. Luo and N. Jha. Power-conscious joint scheduling of periodic task graphs and a periodic tasks in distributed real-time embedded systems. In *ICCAD*, 2000.

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

[6] 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.

[7] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *ICCAD*, pages 365–368, 2000.

[8] 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.

[9] F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

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

[11] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC*, 2002.

A Appendix

A.1 Convex Minimization Problem

We prove that the in the minimization problem given by Equations 10, **??** and **??** 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 [5].
- The sum of two convex functions is convex [5].
- Given a convex function $f(x)$ and a positive constant c, $c \cdot f(x)$ is a convex [5].

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.

Table 1. Counter Example 1 for High Speed (Sync Mode) Interval

Arrival A_i	Period T_i	C_i	B_i	U_i	S_i
0.0001	5	2	3	0.4	1.0
3.0001	7	3	1	0.8285	0.971
0	60	3	0	0.8785	0.8785

Table 2. Counter Example 2 for High Speed (Sync Mode) Interval

Arrival A_i	Period T_i	C_i	B_i	U_i	S_i
0.0001	5	2	2.5	0.4	0.9
3.0001	7.5	2	2.5	0.6667	1.0
0	100	3	0	0.6967	0

B Counter Examples and intuition

B.1 Claim on High Speed Interval

Claim: If the current job J_c, blocks a higer priority job J_h, then we set the high speed interval upto the completion of the current job J_c.

ResultThe above claim is **FALSE**.

Consider a periodic task system as shown in Table 1. Figure 6 shows the schedule. It is seen that the tasks need to execute at a speed H upto the task T_b resumes execution outside the critical section. Example, but such B_i values are not NOT possible under SRP. So we present another example with the same B_i values for both.

B.2 Claim on Mode switching

Claim: You cannot blindly follow the *synchronization* mode and switch to *independent mode* when no further blocking.

The modes are well formed by themselves, however you cannot change between modes arbitrarily. We cannot blindly follow the modes without making sure that the synchronization mode is higher OR selecting the maximum of the two modes on blocking.

In the above example, the high priority tasks have a higher speed in independent mode and lower in the synchronization mode. This is really not a practical case, however we have to consider the worst case for a generalized framework. Once a task is blocked, you execute the tasks at a low speed and after blocking is over, you will continue task τ_3 at the low independent mode speed. This will result in a deadline miss for the task τ_3.

In the example τ_3 blocks for 0.1 time units and then the tasks τ_2 and τ_1 execute at the sync mode speed. When τ_3 resumes execution in the independent mode, it will miss its deadline.

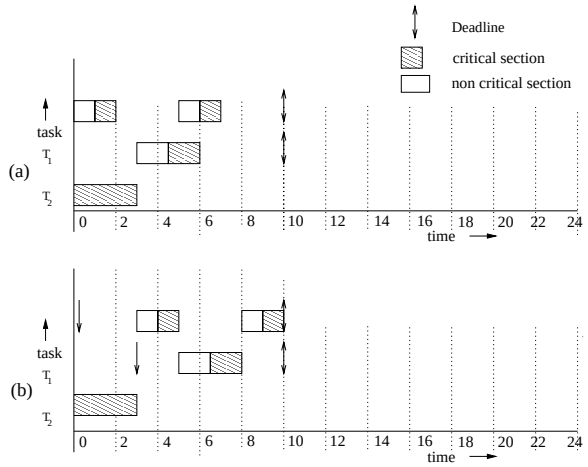

Figure 6. Counter Example for the Claim. (a) The tasks with their arrival times and deadlines (Period = Deadline). $H = 1.0$ and $L = 0.878$. (b) The tasks need to execute at a speed of $\eta = 1.0$ to meet the deadline. It can be easily seen that the tasks will miss their deadline at a speed of L is used for τ_2, though it is not blocked for any time.

Table 3. Counter Example for Switching Between Modes

Arrival A_i	Period T_i	C_i	B_i	η_i^I	η_i^S
0.0001	5	2	1	1.0	0.9
0.0001	8	2	1	1.0	0.675
0	11	1.1	0	$\frac{1}{3.5}$	0.54

Solution : Consider switching to the $max(\eta_i^I, \eta_i^S)$. Since it is the maximum the proof will follow from that by Zhang and Chanson.

Better Solution: Consider the additional constraint :

$$\overset{\forall k}{k = 1, ..., n} \quad \sum_{i=1}^{k} \frac{1}{\eta_i^I} \frac{C_i}{D_i} \geq \frac{1}{\eta_k^S} \frac{B_k}{D_k} + \sum_{i=1}^{k} \frac{1}{\eta_i^S} \frac{C_i}{D_i} \tag{14}$$

B.3 Claim on Feasiblility Test under Task Synchronization

The following test does not guarantee feasibility:

$$\overset{\forall i}{i = 1, ..., n} \quad \frac{1}{\eta_i^S} \frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{1}{\eta_k^S} \frac{C_k}{D_k} \leq 1 \tag{15}$$

A schedule when T_1 is blocked for 4 time units and then T_2 arrives at 4 (not blocked). Then it executes 4.5 time units and then second instance of T_1 proceeds, results in a deadline miss.

$$\frac{0.5}{0.125} + \frac{0.25}{0.125} + \frac{4.5}{1} + \frac{0.25}{0.125} = 12.5$$

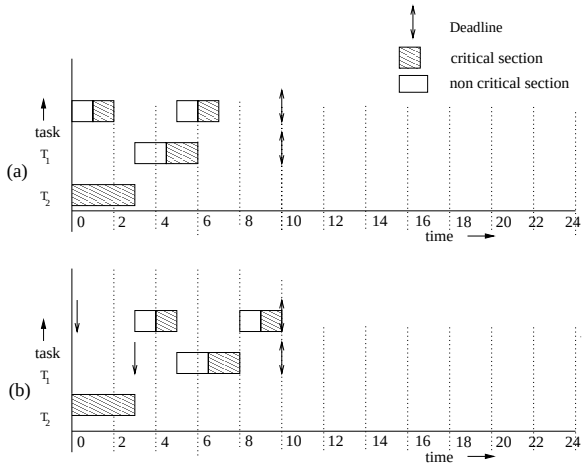

Figure 7. Counter Example for the Claim. (a) The tasks with their arrival times and deadlines (Period = Deadline). $H = 1.0$ and $L = 0.6967$. (b) It can be easily seen that the tasks will miss their deadline when executed at a speed of L when it is not blocked. τ_2, is exeuted at L and misses its deadline. (c) The tasks need to execute at a speed of $\eta = 1.0$ to meet the deadline. (H interval is needed till the blocking task resumes without blocking / finishes / get it precise)

Table 4. Counter Example for Feasiblility Test under Task Synchronization

Arrival A_i	Period T_i	C_i	B_i	η_i^I	η_i^S	Sync Feas Test
0.0001	6	0.25	0.5	0.125	0.125	1
4.0001	7.5	4.5	0.5	1.0	1.0	1
0	120	1.0	0	0.125	0.125	1

$$4 + 2 + 4.5 + 2 = 12.5$$

Thus second instance misses a deadline.

B.4 Blocking Slowdown is WRONG

A schedule when T_1 is blocked for 3 time units and then T_2 arrives at 4 (not blocked). Then it executes 3.25 time units and then second instance of T_1 proceeds, results in a deadline miss.

$$\frac{0.5}{0.166667} + 3/1 + 3.25/1 + 3/1 = 12.5$$

$$3 + 3 + 3.25 + 3 = 12.25$$

Thus second instance misses a deadline.

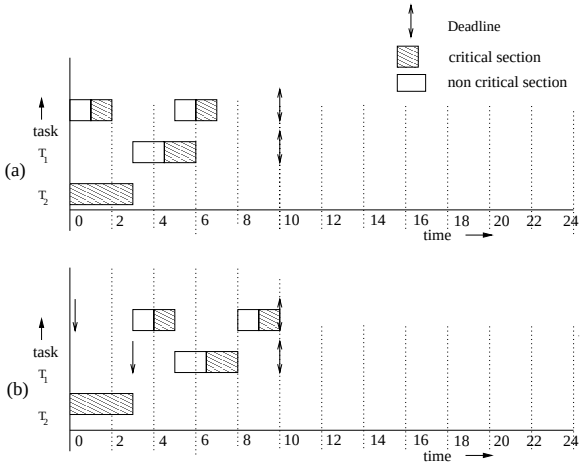

Figure 8. Counter Example for *Modes cannot be kept independent.* (a) The tasks with their arrival times and deadlines (Period = Deadline). $H = 1.0$ and $L = 0.6967$. (b) It can be easily seen that the tasks will miss their deadline when executed at a speed of L when it is not blocked. τ_2, is exeuted at L and misses its deadline. (c) The tasks need to execute at a speed of $\eta = 1.0$ to meet the deadline. (H interval is needed till the blocking task resumes without blocking / finishes / get it precise)

Table 5. Counter Example to show that Blocking Slowdown Factor is WRONG

Arrival A_i	Period T_i	C_i	B_i	η_i^S	η_i^B	Sync Feas Test
0.0001	6	3.0	0.5	1.0	$1/6$	0.583
4.0	7.5	3.25	0.5	1.0	1.0	1
0	90	1.0	0	1/6	X	1

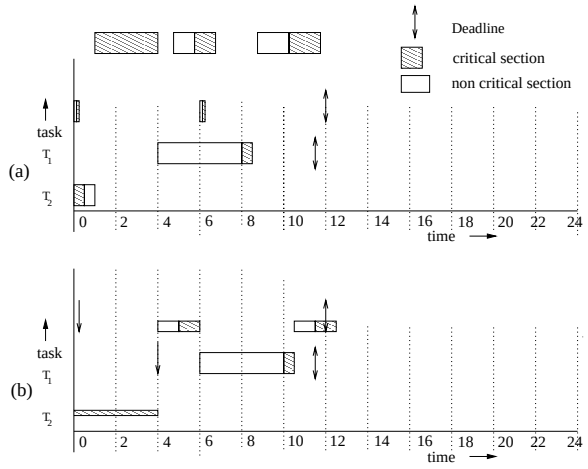

Figure 9. Counter Example for the Claim. (a) The tasks with their arrival times and deadlines (Period = Deadline) at full speed. (b) The tasks are executed at under frequency inheritance algorithm with arbitrary eta_i values satisfying the constraints. It can be easily seen that the task $\tau_{1,2}$ misses its deadline. Since the task is delayed a lot by the slowdown / blocking encountered by task τ_1. Feasibility of task τ_2 depends on the condition $B_2 \leq B_1$, when the blocking arises from a lower priority task e.g. τ_3. The considion is implicitly satisfied in SRP/PCP (at full speed).

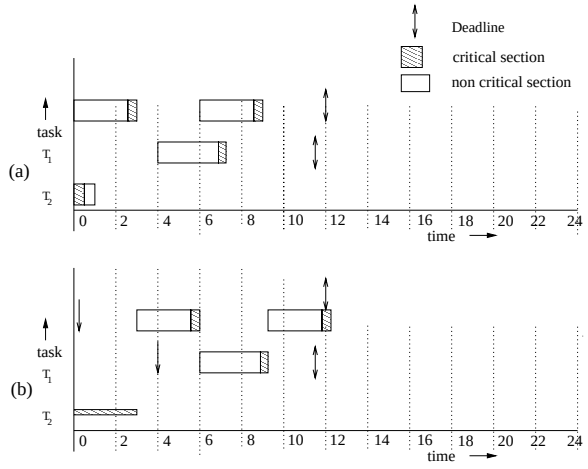

Figure 10. Counter Example for the Claim. (a) The tasks with their arrival times and deadlines (Period = Deadline) at full speed. (b) The blocking critical sections are executed at / inherit the blocking slowdown factor. It is seen that the task $\tau_{1,2}$ misses its deadline. Since the task is delayed a lot by the slowdown / blocking encountered by task τ_1, due to execution at the blocking slowdown factor. Feasibility of task τ_2 depends on the condition $B_2 \leq B_1$, when the blocking arises from a lower priority task e.g. τ_3. The considion is implicitly satisfied in SRP/PCP (at full speed).