2004

Dynamic Voltage Scaling for Sporadic And Periodic Tasks

Ala' Adel Qadi
University of Nebraska at Lincoln, aqadi@cse.unl.edu

Steve Goddard
University of Nebraska – Lincoln, goddard@cse.unl.edu

Shane Farritor
University of Nebraska - Lincoln, sfarritor@unl.edu

Dynamic Voltage Scaling for Sporadic And Periodic Tasks

Ala′ Qadi Steve Goddard
Computer Science & Engineering
University of Nebraska—Lincoln
Lincoln, NE 68588-0115
{*aqadi,goddard*}*@cse.unl.edu*

Shane Farritor
Mechanical Engineering
University of Nebraska - Lincoln
Lincoln, NE 68588-0656
{*sfarritor*}*@unl.edu*

Technical Report TR-UNL-CSE-2004-01
January 2004

Abstract

Dynamic voltage scaling (DVS) algorithms save energy by scaling down the processor frequency when the processor is not fully loaded. Many algorithms have been proposed for periodic and aperiodic task models but none support the periodic and sporadic task models when the deadlines are not equal to their periods. A DVS algorithm, called General Dynamic Voltage Scaling (GDVS), that can be used with sporadic or periodic tasks in conjunction with the preemptive EDF scheduling algorithm with no constraints on the deadlines is presented here. The algorithm is proven to guarantee each task meets its deadline while saving the maximum amount of energy possible with processor frequency scaling when tasks execute with their worst-case execution times. GDVS was implemented in the μC/OS-II real-time operating system for embedded systems. Though theoretically optimal, the actual power savings realized with GDVS depends on the type of the task set and the processor's DVS support. GDVS is tested and evaluated with both a real-time application and a simulated task set. A difference exists between the theoretical power savings and the actual power savings which is due to the limited number of frequency levels the Rabbit 2000 processor supports.

1 Introduction

Many embedded real-time systems consist of a battery operated microprocessor system with a limited battery life. Some of these systems use rechargeable batteries (like cellular phones and robots) while others use dry batteries. In both cases it is very important to maximize the battery life. Dynamic Voltage Scaling (DVS) aims at reducing the power consumption of the system by

operating the processor at a lower frequency and thus on a lower voltage.

In CMOS circuits the power consumed by a CMOS gate is proportional to the square of the voltage applied to the circuit, as shown by Equation (1) where C_L is the gate load capacitance (output capacitance), V_{DD} is the supply voltage and f is the clock frequency [32]. The circuit delay t_d is given by Equation (2) where k is a constant depending on the output gate size and the output capacitance and V_T is the threshold voltage [32]. The clock frequency is inversely proportional to the circuit delay; it is expressed using t_d and the logic depth of a critical path as in Equation (3) where L_d is the depth of the critical path [32].

$$P_{CMOS} = C_L V_{DD}^2 f \tag{1}$$

$$t_d = k \frac{V_{DD}}{(V_{DD} - V_T)^2} \tag{2}$$

$$f = \frac{1}{L_d \cdot t_d} \tag{3}$$

It is clear form Equation (1) that reducing the supply voltage will reduce the power consumption. However it also reduces the clock frequency, as shown by Equations (2) and (3), which slows down the processor, meaning that jobs will be executing at a slower rate. Thus, the challenge in applying DVS algorithms to real-time systems is to save maximum power while still meeting all temporal requirements of the system.

In recent years significant research has been done in the area of DVS (e.g.,[2, 7, 8, 10, 12, 13, 16, 3, 18, 19, 21, 22, 25, 27, 28, 29, 30, 31, 33, 34]). These efforts have resulted in a number of DVS algorithms supporting various task models for embedded and real-time systeys. Successful DVS imprementations in commercial processors include Intel's Xscale processor [9], Transmeta's Crusoe processor [5] and Rabbit Semiconductors' Rabbit processor [26].

DVS algorithms in [2, 7, 8, 10, 13, 16, 3, 18, 21, 22, 27, 28, 29, 30, 34] support variations of the Liu and Layland periodic task model [17] under RM scheduling or EDF scheduling. Algorithms presented in [18, 19, 25] considered task models that also support aperiodic requests with soft deadlines or non-periodic tasks with hard deadlines in which job release times were known a priori.

To date, however, no DVS algorithms support the periodic model when the deadlines are not equal to the periods or the canonical sporadic task model defined by Mok [20] in which tasks have a minimal inter-execution time rather than a fixed period.

Each task in a periodic task set $\mathrm{T} = \{T_1, T_2, \ldots, T_n\}$ has three associated parameters, *p*, *e*, and *d*:

p is the period;

e is the worst-case execution time of the job;

d is the relative deadline for the job.

Each task in a sporadic task set $\mathrm{T} = \{T_1, T_2, \ldots, T_n\}$ also has three associated parameters, *p*, *e*, and *d*:

p is the minimum separation period between the release of two consecutive jobs of a task;

e is the worst-case execution time of the job;

d is the relative deadline for the job.

In this work, a DVS algorithm called General Dynamic Voltage Scaling (GDVS) algorithm is presented and evaluated. GDVS supports periodic and sporadic task models executed under EDF scheduling with no constraints in deadlines. The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 presents our DVS algorithm. Section 4 proves the optimality of the algorithm in terms schedulability and theoretical power savings. Section 5 presents the implementation and evaluation of the algorithm in a stand-alone environment and in an embedded real-time system. We conclude with a discussion of results in Section 6.

2 Related Work

The algorithms in [2, 21, 28, 29] assume the periodic task model and rely on the principles of intra-task DVS. That is, they adjust the processor voltage level, and hence the processor speed, based on the execution path a task takes and commonly rely on compiler support rather than operating system support to conserve power.

The algorithms in [7, 8, 10, 12, 13, 16, 3, 22, 25, 29, 30, 34] also assume the periodic task model with deadlines equal to periods, but rely on an alternative approach to intra-task DVS, called inter-

task DVS. In general, inter-task DVS algorithms determine the processor voltage on a task-by-task basis. That is, they adjust the supply voltage at a task level such that idle time is removed from the schedule while guaranteeing that all tasks meet their respective deadlines.

The approach used in this work falls into the category of inter-task DVS. Of the published inter-task DVS work, the algorithm by Aydin et al. (presented as Proposition 3) in [1] and the Static Voltage Scaling algorithm by Pillai and Shin [22] are the most closely related to our DVS approach. The algorithms are essentially the same, albeit with very different presentations, and appear to be simultaneously discovered. For simplicity, we refer to this algorithm as the Static Voltage Scaling algorithm—the name provided by Pillai and Shin.

The Static Voltage Scaling algorithm is an off-line algorithm that scales the processor voltage by a factor equal to α where α is the minimum utilization required for the task to remain schedulable under EDF or RM scheduling. This technique is also used in our approach to remove deterministic idle time from the schedule, as computed using worst-case execution times (WCET) for each task, but in a slightly different way.

The other DVS algorithms in [1] first use Static Voltage Scaling to set the base processor frequency and then make additional on-line reductions in processor frequency (voltage) by (i) adapting to the actual execution times and (ii) speculating on the early completion of future jobs. The second on-line algorithm in [22] also conserves energy by first using Static Voltage Scaling to set the base processor frequency and then further reduces the voltage level when a job executes for less than its WCET. The third on-line algorithm in [22] saves additional energy by deferring task execution as much as possible. Our algorithm would give the same result as the Static Voltage Scaling algorithm if all the tasks execute periodically with deadlines equal to periods.

The algorithm presented by Shin and Choi in [29, 30] also sets the initial voltage level using Static Voltage Scaling. They then lower the voltage level further whenever a single task is eligible for execution. Lee et al. [3] developed their DVS algorithms using only two voltage levels and distributing the tasks into two sets, each corresponding to one of the voltage levels: *High* and *Low*. Their work was based on the results of Ishihara and Yasuura [10] who formulated the processor

energy optimization problem as a discrete optimization problem that could be solved using linear integer programming techniques.

Kawaguchi et al. [12] presented an approach to schedule a periodic task set by means of task slicing and queues for fixed priority preemptive scheduling, which mainly makes use of the fact that tasks often do not execute with their WCET.

Hong et al. [7, 8] proposed a synthesis technique for variable voltage core based systems containing a set of independent, asynchronous periodic tasks with arbitrary start times (phases) that were scheduled with non-preemptive fixed priority scheduling. Zhang and Chanson present three algorithms in [34] that apply DVS to a periodic task model with non-preemptable sections. This work assumes all tasks are independent and fully preemptive.

The algorithms in [18, 19, 25] consider variations of the periodic task model that support aperiodic requests with soft deadlines or non-periodic tasks with hard deadlines in which job release times are known a priori. Luo and Jha [18] presented an algorithm to schedule periodic tasks, soft aperiodic tasks and hard aperiodic tasks with precedence constraints using task graphs, cyclic scheduling and slack steeling.

This is the first work to support DVS for both periodic and sporadic tasks with no constraints on deadlines

3 A General DVS Algorithm

The General Dynamic Voltage Scaling (GDVS) algorithm presented here is classified as an inter-task DVS algorithm. That is, it adjusts the processor voltage on a job-by-job basis, where a job represents the release of a task. Recall from Equations (2) and (3) that the processor frequency is proportional to the voltage level. As with most DVS algorithms, GDVS is defined in terms of processor frequency, rather than voltage levels, since the relationship between the processor frequency and task execution times can be expressed directly.

The GDVS algoritm is similar to our DVVST algorithm presented in [23] in that it maintains a frequency-scaling factor, α, that represents the percent of the maximum processor frequency. Rather

than using the Static Voltage Scaling algorithm of [22] to set the initial frequency level, GDVS starts with a minimum possible frequency-scaling factor, which can be theoretically zero, and scales the processor frequency up and down depending when jobs are released. The scaling factor α is increased by an amount of $e_i/min(p_i, d_i)$ when task T_i is first released. Let r_i be the last release time of task T_i. GDVS reduces α at time $r_i + min(p_i, d_i)$ by the amount of $e_i/min(p_i, d_i)$ if the next job if task T_i was not yet released. When task T_i later releases the job, α is increased by the same amount. For periodic tasks with $d_i = p_i$, α does not change after the first release since the next job is released at the deadline of the current job. The algorithm is explained in detail after we introduce a few definitions.

Definition 1: *The frequency-scaling factor, α, is defined as the ratio between the new processor frequency and the maximum processor frequency:*

$$\alpha = \frac{f_{new}}{f_{\max}} \tag{4}$$

Corollary 1: $\alpha \leq 1$.

Proof: The maximum value that we can scale the frequency to is f_{max}. Therefore

$$\alpha \leq \alpha_{\max} = \frac{f_{\max}}{f_{\max}} = 1.$$

□

Definition 2: *The idle-state scaling factor, α_{idle}, is the minimum scaling factor possible that puts the processor in a sleep mode when there is no job to execute.*

Theoretically $\alpha_{idle} = 0$, but in many systems α_{idle} must be greater than zero to support platform requirements, or to interact with external devices that trigger the release of a sporadic task. In this section it is assumed that $\alpha_{idle} = 0$. This assumption is relaxed in the next section when we describe the implementation of the algorithm on a real system.

In [23] a task set called TD is defined as the task set holding all the tasks that did not release a job at their minimum separation period. The definition of TD is modified here to fit the general case of having deadlines not equal to periods for periodic or sporadic tasks.

Definition 3: *TD, is a subset of the task set* $T = \{T_1, T_2, \ldots, T_n\}$ *where for every task in TD the scaling factor* α *has been reduced by an amount equal to* $\frac{e_i}{min(d_i,p_i)}$.

The GDVS algorithm is shown in Figure 1. Initially, *TD* = T and the processor frequency is set to α_{idle}. When a task T_i releases a job, the algorithm immediately increases the scaling factor α by an amount equal to $\frac{e_i}{min(d_i,p_i)}$ and removes task T_i from the set *TD*. If the deadline of task is T_i is greater or equal to the period and T_i does not release a job at the end of its minimum separation period p_i the algorithm reduces the scaling factor α by an amount equal to$\frac{e_i}{min(d_i,p_i)}$ and task T_i is added to the set *TD*. If the deadline is less than the period, then the algorithm always reduces the scaling factor by $\frac{e_i}{min(d_i,p_i)}$ at $r_i + d_i$ where r_i is the latest job release time and d_i is the deadline of the job. If the algorithm detects that no job is currently executing, then it sets α to the minimum possible value $\alpha_{idle,}$or in other words, it sets the processor to the idle or sleep mode.

The value of α may depend on the previous value of α and since α changes with time, we use α_n to represent the n^{th} change to α at time t. Equation (5) shows how, if at all, α_n is changed at time t.

```
GDVS( ):
set α = α_idle and TD = T // set initial conditions
while(true) {
  sleep until (∃  T_i : (T_i releases a job and T_i ∈TD) or
                  (T_i ∉TD and current_time ≥ r_i + p_i))
                  or (no task is executing)
  if T_i released a job and T_i ∈TD then
                  // scale up the processor frequency
     set α = α + e_i/min(d_i,p_i) and TD = TD - {T_i}
  else if T_i ∉TD and current_time ≥ r_i + min(d_i, p_i)
                  // scale down the processor frequency
     set α = α − e_i/min(d_i,p_i) and TD = TD + {T_i}
  else            // set processor to idle mode
     set α = α_idle and TD = T }
```

Figure 1. The GDVS Algorithm.

$$\alpha_n = \begin{cases} \alpha_{idle}\ ,\ t = 0 \text{ or } no\ task\ is\ executing \\ \alpha_{n-1} - \frac{e_i}{min(d_i,p_i)}\ ,\ t \geq r_i + min(d_i, p_i) = 0 \text{ and } T_i \notin TD \\ \alpha_{n-1} + \frac{e_i}{min(d_i,p_i)}\ ,\ T_i\, is\, released\, at\ time\ t \text{ and } T_i \in TD \\ no\ change\ otherwise \end{cases} \tag{5}$$

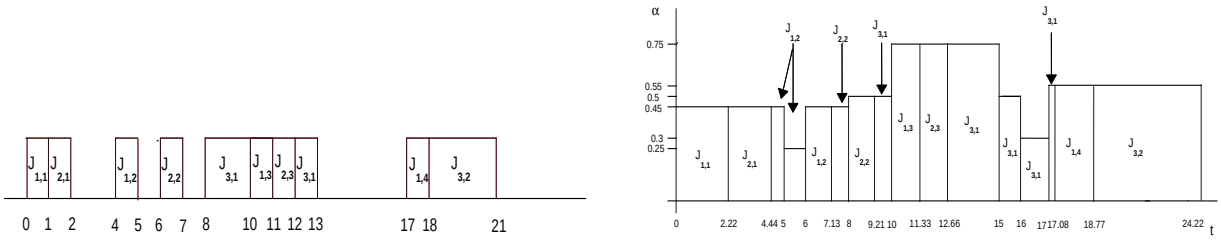
The following example illustrates how the GDVS algorithm scales the processor frequency (voltage) under EDF scheduling.

Job	$r_{i,j}$	$D_{i,j}$	EDF without GDVS			EDF with GDVS		
			Exec Interval	α	% of Job Executed	Execution Interval	α	% of Job Executed
$J_{1,1}$	0	4	[0,1)	1	100%	[0,2.22)	.45	100%
$J_{2,1}$	0	5	[1,2)	1	100%	[2.22,4.44)	.45	100%
$J_{1,2}$	4	8	[4,5)	1	100%	[4.44,5)	.45	24.3%
						[5,6)	.25	25%
						[6,7.13)	.45	50.7%
$J_{2,2}$	6	11	[6,7)	1	100%	[7.13,8)	.45	39.33%
				1		[8,9.21)	.5	60.67%
$J_{3,1}$	8	18	[8,10)	1	66.67%	[9.21,10)	.5	13.13%
			[12,13)	1	33.33%	[12.66,15)	.75	58.34%
						[15,16)	.5	16.66%
						[16,17)	.3	10%
						[17,17.08)	.75	1.87%
$J_{1,3}$	10	14	[10,11)	1	100%	[10,11.33)	.75	100%
$J_{2,3}$	11	16	[11,12)	1	100%	[11.33,12.66)	.75	100%
$J_{1,4}$	17	21	[17,18)	1	100%	[17.08,18.77)	.55	100%
$J_{3,2}$	18	28	[18,21)	1	100%	[18.77,24.22)	.55	100%

Table 1. Job attributes of the example task set when executed under EDF with and without GDVS. The columns labelled $r_{i,j}$ and $D_{i,j}$ represent the release time and absolute deadlines of job $J_{i,j}$. The scaling factor α is set at the start of each execution interval.

Example 1: Let us consider the sporadic task set T_1 = (1,4), T_2 = (1,5), T_3 = (3,10), note that in this example $d_i = p_i$ therefore we represented the task set with tuple (e, p). The (un-scaled) system utilization is U = 0.75. Let us consider scheduling the jobs that were released in the interval [0, 20) under preemptive EDF while using the GDVS algorithm to scale the processor frequency (voltage). Assume that the tasks released jobs as follows: T_1 at times 0, 4, 10, and 17; T_2 at times 0, 6, and 11; and T_3 at times 8 and 18. Let job $J_{i,j}$ represent the j^{th} release of task T_i. Figure 2(a) illustrates the execution of these jobs without GDVS, and Figure 2(b) illustrates the same jobs executed with GDVS. The specific job attributes for both executions are listed in Table 1.

Notice that in Figure 2(a) the processor is idle in the intervals [2,4), [5,6), and [13,17) under EDF scheduling without GDVS. For this set of release times, the GDVS algorithm resulted in an execution in which the processor was never idle during the observed period shown in Figure 2(b). However, no task missed its deadline—a fact proven in the next section for all feasible task sets.



(a) Executing the example task set under EDF without GDVS.

(b) Executing the example task set under EDF with GDVS.

Figure 2. Executing the example task set under EDF with GDVS. The x-axis in each figure represents time. In (b), the y-axis represents the frequency scaling factor α, which is set at the start of each execution interval.

4. Theoretical Validation

This section addresses the temporal correctness and energy savings possible when task sets are executed under EDF with GDVS. Section 4.1 presents the temporal correctness and optimality of EDF with GDVS. Section 4.2 quantifies the power savings possible when both the processor voltage and frequency can be scaled, as well as when only the processor frequency can be scaled. It is shown that GDVS is optimal with respect to power savings when only the frequency can be scaled and all tasks execute with their WCET.

4.1 Temporal Correctness

A voltage (frequency) scaling scheduling algorithm for real-time systems is correct if it guarantees that all jobs meet their deadlines under a specified scheduling algorithm. Scaling the processor frequency results in new task execution times that are proportional to the frequency-scaling factor. Theorem 1 gives an equation to calculate the processor time capacity. Theorem 2, states that schedulability under EDF is a necessary and sufficient feasibility condition for the task sets to be schedulable under GDVS. Before presenting these theorems, however, new definitions are required.

Definition 4: *Scaled-mode execution time, e_s, is the execution time needed to execute a job under a frequency-scaling factor.*

Over any time interval where the scaling factor α is constant, e_s can be calculated by Equation (6) where e_{si} is the scaled execution time of task T_i, e_i is the (normal) worst-case execution time of T_i, and α_c is the current scaling factor.

$$e_{si} = \frac{e_i}{\alpha_c} \tag{6}$$

Definition 5*: Scaling Factor Change Interval, $\tau_{Si,}$ is the time interval between two consecutive scaling factor changes α_i and α_{i+1}.*

Before we introduce the Theorem that states the necessary and sufficient condition for schedulability under GDVS we need to introduce a few definitions and theorems, some of them have already been introduced in [24].

Definition 6: *Job Inter-Release Time is the time interval between the release of any job of task T_i and the release of the next job of the same task. That is the time interval between the release of job $J_{i,j}$ and the release of job $J_{i,j+1}$.*

We use the notation $\delta_{i,j}$ to denote the inter-release time between jobs $J_{i,j}$ and $J_{i,j+1}$ where $r_{i,j}$ is the release time of job $J_{i,j}$:

$$\delta_{i,j} = r_{i,j+1} - r_{i,j} \tag{7}$$

Corollary 1: $\delta_{i,j} \geq p_i$

Proof: This corollary follows directly from the definition of the periodic and sporadic task in which tasks must have a minimum separation p between the release of jobs. Therefore the job inter-release time cannot be less than p_i. □

Corollary 2: *In any job inter-release time interval $\delta_{i,j} = [r_{i,j}, r_{i,j+1})$ the processor is scaled by an amount equal to $\frac{e_i}{min(d_i,p_i)}$ for a time amount equal to $min(d_i, p_i)$.*

Proof: At the instant $r_{i,j}$ either the algorithm scales up the frequency by $\frac{e_i}{min(d_i,p_i)}$ if the frequency had been scaled down for that task—otherwise frequency has already been scaled up for that task. The algorithm does not scale down processor frequency $\frac{e_i}{min(e_i,p_i)}$ until $r_{i,j} + min(d_i, p_i)$. Thus, since $r_{i,j} + min(d_i, p_i) \leq r_{i,j+1} - r_{i,j}$, the processor is scaled up by $\frac{e_i}{min(d_i,p_i)}$ for every $\delta_{i,j}$. □

Definition 7 *The Task Scaling Factor Active Time t_{up_i} is the total time for which the processor frequency was scaled up by a factor equal to $\frac{e_i}{min(d_i,p_i)}$ in any time interval*

The processor time capacity definition is introduced in [24] to provide a way of quantifying the available processor time when frequency is being scaled. The processor time capacity quantifies the available processor time by looking at the system as if execution times are the same but time itself is scaled. We state the definition here again to emphasize it since we will introduce more analysis based on it.

Definition 8: *Processor Time Capacity ρ is the amount of scaled time available in any real time interval when the processor is running in scaled mode.*

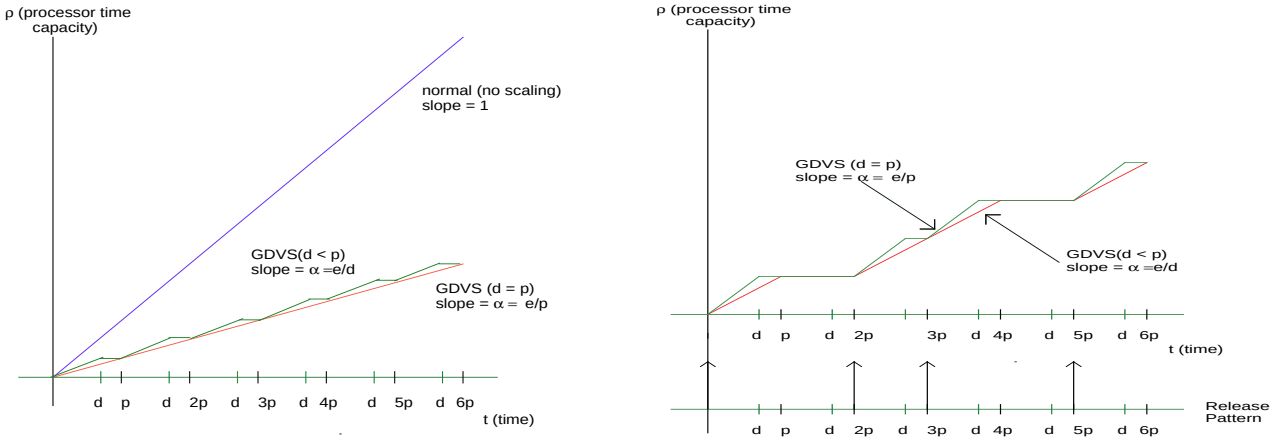
Over any time interval $[t_{1,}t_2)$ where α is constant, the processor time capacity ρ or the scaled time is the product of the length of the interval times the scaling factor as given in Equation (8).

$$\rho = \alpha(t_2 - t_1) \tag{8}$$

Figure 3(a) shows processor time capacity as a function of time for a periodic task set with one task, if there is no voltage scaling then the processor time capacity will be equal to the current time. Therefore the processor time capacity function is a straight line with a slope of one. However if the deadline for the task is equal to the period then the processor time capacity function is a straight line with a slope equal to $\alpha = \frac{e_i}{p_i}$. If the deadline is less than the period then processor time capacity function is straight line with a slope equal to $\alpha = \frac{e_i}{d_i}$ in the intervals between the job release time and the deadline of that job. Then it becomes a constant in the time interval from the deadline to the end of the period because α has been reduced by $\alpha = \frac{e_i}{d_i}$ and since this is the only task we have $\alpha = 0$. If we have more than one task then we can sum the processor time capacity resulting from each task to get the total processor time capacity.

Figure 3(b) shows processor time capacity as a function of time for a sporadic task set with one task, we can see the difference between the two cases where the deadlines are equal to the periods or less than the periods. We can see that the capacity flattens out if a task has released a job at its minimum separation period.

If we have more than one task then we can sum the processor time capacity resulting from each task to get the total processor time capacity.



(a) Processor Time Capacity Verses Time For A Single Periodic Task

(b) Processor Time Capacity Verses Time For A Single Sporadic Task

Figure 3. Processor Time Capacity Verses Time

Lemma 1: *The processor time capacity ρ_{T_i} for a task T_i in any time interval $\tau = [t_0, t_d)$, where t_0 is an idle instant can be calculated from Equation (9)*

$$\rho_{T_i} = \begin{cases} N_i \cdot e_i & t_d - r_{i,N_i} \geq min(d_i, p_i) \\ \left((N_i - 1) + \left(\frac{(t_d - r_{i,N_i})}{min(d_i,p_i)}\right)\right) \cdot e_i & t_d - r_{i,N_i} < min(d_i, p_i) \end{cases} \tag{9}$$

where N_i is the number of jobs released by Task T_i in $[t_0, t_d)$ and r_{i,N_i} is the release time of the last job in $[t_0, t_d)$.

Proof: In general for any task T_i in any time interval $[t_0, t_d)$

$$\rho_{T_i} = Postive\, Change\, in\, \alpha \times Time\, Interval\, For\, The\, Change \tag{10}$$

$$\rho_{T_i} = \alpha_i \times t_{up_i} \tag{11}$$

Because the algorithm only scales down the frequency at the end of $min(d_i, p_i)$, the task scaling factor active time t_{up_i} is going to be always a multiple of $min(d_i, p_i)$ unless the interval between t_d and the release time of the last job of T_i, r_{i,N_i}, is less than $min(d_i, p_i)$. This means that we have two cases to consider. To make a clear distinction between the two cases let us divide τ into two parts

$t_{integer} = [t_0, r_{i,N_i})$ and $t_{fraction} = [r_{i,N_i}, t_d)$ as shown in Figure 4. τ can be represented as the sum of these two parts.

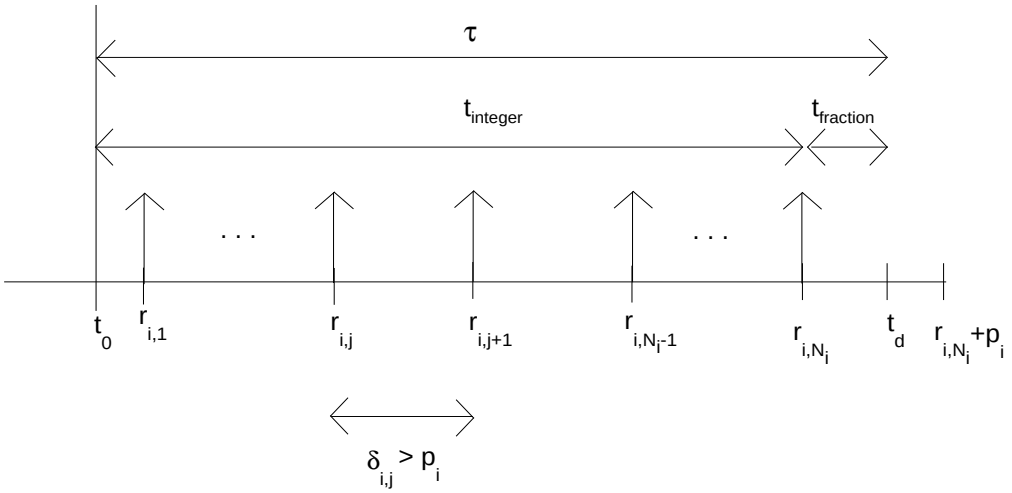


Figure 4. Division Of τ into $t_{integer}$ and $t_{fraction}$

Case 1: $t_d - r_{i,N_i} \geq min(d_i, p_i)$, in this case because $t_d - r_{i,N_i} \geq min(d_i, p_i)$ then the processor has been scaled up by $\frac{e_i}{min(d_i,p_i)}$ for $min(d_i, p_i)$ in $t_{fraction}$. Prior to r_{i,N_i}, $N_i - 1$ jobs have been released by T_i in $t_{integer}$, because the end of t_{integr} is r_{i,N_i}. Because $N_i - 1$ jobs released in $t_{integer}$ we will have $(N_i - 1)$ job inter-release times. By Corollary 2 if we have $(N_i - 1)$ job inter-release times in $t_{integer}$ the frequency will be scaled up by $\frac{e_i}{min(d_i,p_i)}$ for $(N_i - 1) \cdot min(d_i, p_i)$ time units.

Now we can calculate ρ_{T_i}

$$\begin{aligned}
\rho_{T_i} = \alpha_i \cdot t_{up_i} &= \alpha_i \cdot (t_{up_i}\, in\, t_{integer} + t_{up_i}\, in\, t_{fraction}) \\
&= \frac{e_i}{min(d_i, p_i)} \cdot ((N_i - 1) \cdot min(d_i, p_i) + min(d_i, p_i)) \\
&= \frac{e_i}{min(d_i, p_i)} \cdot (N_i \cdot min(d_i, p_i)) = N_i \cdot e_i
\end{aligned} \tag{12}$$

Case 2: $t_d - r_{i,N_i} < min(d_i, p_i)$ in this case because $t_d - r_{i,N_i} < min(d_i, p_i)$ then the processor has been scaled up by $\frac{e_i}{min(d_i,p_i)}$ for the whole interval of $t_d - r_{i,N_i}$. Prior to r_{i,N_i} the the frequency will be scaled up by $\frac{e_i}{min(d_i,p_i)}$ for $(N_i - 1) \cdot min(d_i, p_i)$ for time units the same reason as Case 1.

Now we can calculate ρ

$$\begin{aligned}\rho_{T_i} = \alpha_i \cdot t_{up_i} &= \alpha_i \cdot (t_{up_i}\, in\, t_{integer} + t_{up_i}\, in\, t_{fraction}) \\ &= \frac{e_i}{min(d_i, p_i)} \cdot ((N_i - 1) \cdot min(d_i, p_i) + (t_d - r_{i,N_i})) \\ &= \left((N_i - 1) + \left(\frac{(t_d - r_{i,N_i})}{min(d_i, p_i)}\right)\right) \cdot e_i\end{aligned} \tag{13}$$

□

Theorem 1: *The processor time capacity ρ for a task set $T = \{T_1, T_2..., T_n\}$ in any time interval $\tau = [t_0, t_d)$, where t_0 is an idle instant can be calculated from Equation (14).*

$$\rho = \sum_{T_i \in S_1} N_i \cdot e_i + \sum_{T_i \in S_2} \left((N_i - 1) + \left(\frac{(t_d - r_{i,N_i})}{min(d_i, p_i)}\right)\right) \cdot e_i \tag{14}$$

where $S_1 \subseteq T$, $S_2 \subseteq T$ and

$$\forall T_i \in S_1,\ t_d - r_{i,N_i} \geq min(d_i, p_i)$$

$$\forall T_i \in S_2,\ t_d - r_{i,N_i} < min(d_i, p_i)$$

Proof: Let us divide T into two task sets S_1 and S_2 where $T = S_1 \cup S_2$ and

$$\forall T_i \in S_1,\ t_d - r_{i,N_i} \geq min(d_i, p_i)$$

$$\forall T_i \in S_2,\ t_d - r_{i,N_i} < min(d_i, p_i)$$

Because the processor time capacity for a task is independent of any other task, the total processor time capacity for $T = \{T_1, T_2..., T_n\}$ is

$$\rho_T = \sum_{T_i \in T} \rho_{T_i} \tag{15}$$

For task set S_1, $\rho_{S_1} = \sum_{T_i \in S_1} \rho_{T_i}$, by Lemma 1, ρ for T_i can be calculated from Case 1 of Equation (9). Therefore

$$\rho_{S_1} = \sum_{T_i \in S_1} N_i \cdot e_i$$

For task set S_2, $\rho_{S_2} = \sum_{T_i \in S_2} \rho_{T_i}$, by Lemma 1, ρ for T_i can be calculated from Case 2 of Equation (9). Therefore

$$\rho_{S_2} = \sum_{T_i \in S_2} \left((N_i - 1) + \left(\frac{(t_d - r_{i,N_i})}{min(d_i, p_i)}\right)\right) \cdot e_i$$

For the whole task set T we have

$$\rho_T = \rho_{S_1} + \rho_{S_2}\ because\ T = S_1 \cup S_2$$
$$\rho_T = \sum_{T_i \in S_1} N_i \cdot e_i + \sum_{T_i \in S_2} \left((N_i - 1) + \left(\frac{(t_d - r_{i,N_i})}{min(d_i, p_i)} \right) \right) \cdot e_i$$

□

Corollary 3: *For a periodic task with $d_i \geq p_i$ the processor time capacity in any time interval $\tau = [t_0, t_d)$ where t_0 is an idle instant given by Equation (16).*

$$\rho = (t_d - t_0) \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} \tag{16}$$

Proof: In this case the capacity will always be given by Case 2 of Equation (14) unless $t_d - r_{i,N_i} = p_i$ because $t_d - r_{i,N_i}$ is going to be always $\leq min(d_i, p_i) = p_i$. The reason for this is that the task set is periodic, therefore we will not have any interval greater than p_i without the release of a job. The division of τ into $t_{integer}$ and $t_{fraction}$ is shown in Figure 5. $t_{integer}$ and $t_{fraction}$ can be calculated from Equations (17) and (18) respectively. Note that the capacity is zero in the interval $[t_0, r_{i,1})$ because the processor was idle at t_0 and will remain idle until $r_{i,1}$

$$t_{integer} = \left\lfloor \frac{(t_d - t_0)}{p_i} \right\rfloor \cdot p_i \tag{17}$$
$$t_{fraction} = t_d - r_{i,N_i} = \left((t_d - t_0) - \left\lfloor \frac{(t_d - t_0)}{p_i} \right\rfloor \cdot p_i \right) \tag{18}$$

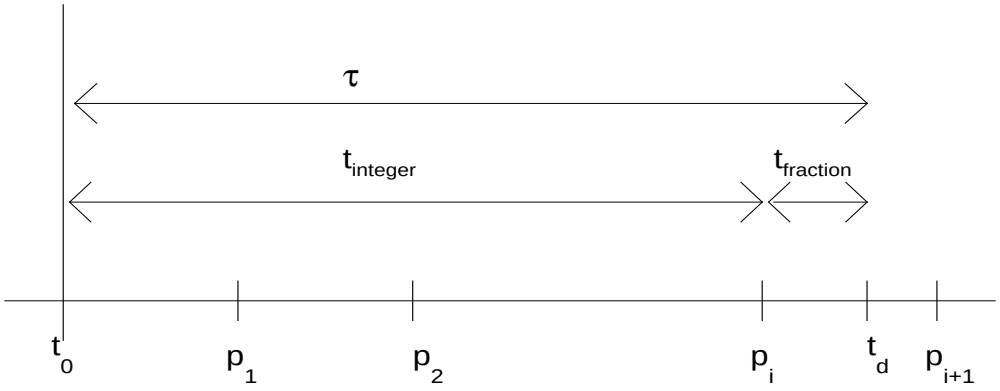


Figure 5. Division Of τ into $t_{integer}$ and $t_{fraction}$

The number of released jobs N_i by task T_i in $[t_d, t_0)$ is $\left\lceil \frac{t_d - t_0}{p_i} \right\rceil$. Therefore

$$N_i - 1 = \left\lceil \frac{t_d - t_0}{p_i} \right\rceil - 1 = \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \tag{19}$$

Substituting Equations (19) and (18) in Case 2 of Equation (14) we get

$$\rho_{T_i} = \left(\left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor + \left((t_d - t_0) - \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot p_i \right) \cdot \frac{1}{p_i} \right) \cdot e_i$$

$$\rho_{T_i} = \left(\left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor + \left(\frac{t_d - t_0}{p_i} - \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \right) \right) \cdot e_i = \frac{t_d - t_0}{p_i} \cdot e_i = (t_d - t_0) \cdot \frac{e_i}{p_i}$$

If $t_d - r_{i,N_i} = p_i$ then we need to substitute in the first case of Equation (14). The number of released jobs N_i by task T_i in $[t_d, t_0)$ is $\left\lceil \frac{t_d - t_0}{p_i} \right\rceil$. $[t_d, t_0)$ ia an integer multiple of p_i because $t_d - r_{i,N_i} = p_i$ therefore we can remove the the ceiling function to get

$$\rho_{T_i} = \frac{t_d - t_0}{p_i} \cdot e_i = (t_d - t_0) \cdot \frac{e_i}{p_i}$$

Using Equation (15) to calculate the capacity for the whole task set we get

$$\rho = \sum_{i=1}^{n} \rho_{T_i} = \sum_{i=1}^{n} (t_d - t_0) \cdot \frac{e_i}{p_i} = (t_d - t_0) \cdot \sum_{i=1}^{n} \frac{e_i}{p_i}$$

□

Corollary 4: *For a periodic task with $d_i < p_i$ the processor time capacity in any time interval $\tau = [t_0, t_d)$ where t_0 is an idle instant is given by Equation (20).*

$$\rho = \sum_{T_i \in S_1} \left\lceil \frac{\tau}{p_i} \right\rceil \cdot e_i + \sum_{T_i \in S_2} \left(\left\lfloor \frac{\tau}{p_i} \right\rfloor + \left(\tau - \left\lfloor \frac{\tau}{p_i} \right\rfloor \cdot p_i \right) \cdot \frac{1}{d_i} \right) \cdot e_i \tag{20}$$

where $S_1 \subseteq T$, $S_2 \subseteq T$ and

$$\forall T_i \in S_1,\ t_d - r_{i,N_i} \geq min(d_i, p_i)$$

$$\forall T_i \in S_2,\ t_d - r_{i,N_i} < min(d_i, p_i)$$

Proof: If T is periodic and $d_i < p_i$ then we have

$$N_i = \left\lceil \frac{t_d - t_0}{p_i} \right\rceil \tag{21}$$

$$N_i - 1 = \left\lceil \frac{t_d - t_0}{p_i} \right\rceil - 1 = \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \tag{22}$$

$$t_d - r_{i,N_i} = (t_d - t_0) - \left\lfloor \frac{(t_d - t_0)}{p_i} \right\rfloor \cdot p_i \tag{23}$$

$$min(d_i, p_i) = d_i \tag{24}$$

Substituting Equations (21), (22), (23) and (24) in (14) we get Equation (20) directly. □

A necessary and sufficient condition to schedule periodic and sporadic tasks under EDF is given in Theorem 4.2 in [11]. The theorem states that a task set T is schedulable if and only if

$$\forall L > 0, L \geq \sum_{i=1}^{n} f\left(\frac{L - d_i + p_i}{p_i}\right) \cdot e_i \tag{25}$$

where

$$f(a) = \begin{cases} \lfloor a \rfloor & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases} \tag{26}$$

and L is any time interval. The least upper bound on demand by any task can be calculated form Equation (27) presented in Lemma 4.1 of [11].

$$D_{T_i}[0, L) = f\left(\frac{L - d_i + p_i}{p_i}\right) \tag{27}$$

where $D_{T_i}[0, L)$ is the least upper bound on the demand in the interval $[0, L)$ by T_i and the function f is defined by Equation (27).

Corollary 5: *$D[0, L)$ by a task set with $d_i > p_i$ is less than or equal to $D[0, L)$ a task set with the same execution times and $d_i = p_i$ where $D[0, L)$ is the least upper bound on the demand in the interval $[0, L)$ by the task set.*

Proof: If $L \geq d_i - p_i$ then $D[0, L)$ for a task set with $d_i = p_i$ is

$$D[0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L - d_i + p_i}{p_i} \right\rfloor \cdot e_i = \sum_{i=1}^{n} \left\lfloor \frac{L - p_i + p_i}{p_i} \right\rfloor \cdot e_i = \sum_{i=1}^{n} \left\lfloor \frac{L}{p_i} \right\rfloor \cdot e_i \tag{28}$$

For a task set with the same execution times and $d_i > p_i$, if $L \geq d_i - p_i$, $D[0, L)$ is

$$D[0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L - d_i + p_i}{p_i} \right\rfloor \cdot e_i = \sum_{i=1}^{n} \left\lfloor \frac{L}{p_i} + \frac{p_i - d_i}{p_i} \right\rfloor \cdot e_i \tag{29}$$

$\frac{p_i - d_i}{p_i} < 0$ because $d_i > p_i$, therefore

$$\left\lfloor \frac{L}{p_i} + \frac{p_i - d_i}{p_i} \right\rfloor \leq \left\lfloor \frac{L}{p_i} \right\rfloor$$

$$\Rightarrow \quad \left\lfloor \frac{L}{p_i} + \frac{p_i - d_i}{p_i} \right\rfloor \cdot e_i \leq \left\lfloor \frac{L}{p_i} \right\rfloor \cdot e_i$$

$$\Rightarrow \quad \sum_{i=1}^{n} \left\lfloor \frac{L}{p_i} + \frac{p_i - d_i}{p_i} \right\rfloor \cdot e_i \leq \sum_{i=1}^{n} \left\lfloor \frac{L}{p_i} \right\rfloor \cdot e_i$$

$$\Rightarrow \quad D[0, L) \, by \, T \, with \, (d_i > p_i) \leq D[0, L) \, by \, T \, with \, (d_i = p_i)$$

If $L < d_i - p_i$ then $D[0, L)$ equals 0 for both T with $d_i > p_i$ and T with $d_i = p_i$. □

Theorem 2: *Let $T = \{T_1, T_2, \ldots, T_n\}$ be either a periodic or a sporadic task set with no constraints on the deadlines, then preemptive EDF with GDVS will succeed in scheduling T if and only if:*

$$\forall L > 0, L \geq \sum_{i=1}^{n} f\left(\frac{L - d_i + p_i}{p_i}\right) \cdot e_i$$

where f is defined in Equation (26) and L is any interval in time.

Proof: Establishing the contrapositive shows necessity, i.e., a negative result from the equation implies that T is not feasible. Let us assume a negative result for the equation, that is

$$\exists L > 0, L < \sum_{i=1}^{n} f\left(\frac{L - d_i + p_i}{p_i}\right) \cdot e_i \tag{30}$$

But we know that if Equation (30) holds then EDF will not find a feasible schedule, therefore GDVS combined with EDF will not find a feasible schedule.

To show the sufficiency of the theorem, we assume that Equation (25) holds, GDVS and EDF are used to schedule T, yet a job misses its deadline. Let job J_d be the first job to miss its deadline at time t_d, let t_0 denote the last processor idle instant. We note that at the worst case, we will at least have an idle instant at $t = 0$.

Let τ be the time interval $[t_0, t_d)$. If job J_d missed its deadline at t_d then the demand in $[t_0, t_d)$ must have been greater than the processor time capacity in $[t_0, t_d)$. That is

$$processor\, time\, capacity \;<\; demand \tag{31}$$

We will show that it is not possible for the demand to be greater than the processor time capacity for all the possible cases then prove that for any combination of those cases it will not be possible for the demand to be greater than the processor time capacity.

Case 1: Lets assume that $\forall i,\ 0 < i \leq n,\ d_i \leq p_i$ and $t_d - r_{i,N_i} < min(d_i, p_i)$. First we will calculate the demand in $[t_0, t_d)$. Because $d_i \leq p_i$, no job $J_{i,j}$ will demand any processor time beyond $r_{i,j} + d_i$, $\forall i, j,\ 0 < i \leq n, 0 < j \leq N_i$ where n is the number of Tasks, N_i is the number of released jobs by task i in $[t_0, t_d)$. Because $d_i \leq p_i$ there is no intersection between $[r_{i,j}, r_{i,j} + p_i)$

and $[r_{i,k}, r_{i,k} + p_i), \forall i, j, k,\ 0 < i \leq n,\ 0 < j, k \leq N_i$, therefore we have only to account for the demand in the intervals $[r_{i,j}, r_{i,j} + d_i),\ \forall i, j, 0 < i \leq n, 0 < j \leq N_i - 1$ and $[r_{i,N_i}, t_d)$. Therefore

$$\begin{aligned} Demand\, in\, [t_0, t_d) =& Demand\, in\, [r_{i,j}, r_{i,j} + d_i)\, Intervals + \ Demand\, in\, [r_{i,N_i}, t_d) \\ Demand\, in\, [t_0, t_d) =& \sum_{i=1}^{n} (N_i - 1) \cdot Demand\, in\, [r_{i,j}, r_{i,j} + d_i)\, by\, T_i \\ &+ \sum_{i=1}^{n} Demand\, in\, [r_{i,N_i}, t_d)\, by\, T_i \end{aligned}$$

Because we are only concerned with the maximum demand that can occur, we can use the upper bound on the demand. Therefore

$$D[t_0, t_d) = \sum_{i=1}^{n} (N_i - 1) \cdot D_{T_i}[r_{i,j}, r_{i,j} + d_i) + \sum_{i=1}^{n} D_{T_i}[r_{i,N_i}, t_d) \tag{32}$$

using Equation (25) to calculate $D_{T_i}[r_{i,j}, r_{i,j} + d_i)$ and $D_{T_i}[r_{i,N_i}, t_d)$ we get

$$D_{T_i}[r_{i,j}, r_{i,j} + d_i) = f\left(\frac{d_i - d_i + p_i}{p_i}\right) \cdot e_i = f\left(\frac{p_i}{p_i}\right) \cdot e_i = f(1) \cdot e_i = \lfloor 1 \rfloor = 1 \cdot e_i = e_i \tag{33}$$

$$D_{T_i}[r_{i,N_i}, t_d) = f\left(\frac{(t_d - r_{i,N_i}) - d_i + p_i}{p_i}\right) \cdot e_i = f\left(1 + \frac{(t_d - r_{i,N_i}) - d_i}{p_i}\right) \cdot e_i \tag{34}$$

$0 < t_d - r_{i,N_i} < min(d_i, p_i) = d_i$ by definition of this case therefore

$$\begin{aligned} & & 0 < t_d - r_{i,N_i} < d_i \\ \Rightarrow & & -d_i < t_d - r_{i,N_i} - d_i < 0 \\ \Rightarrow & & -\frac{d_i}{p_i} < \frac{t_d - r_{i,N_i} - d_i}{p_i} < 0 \\ \Rightarrow & & 1 - \frac{d_i}{p_i} < 1 + \frac{t_d - r_{i,N_i} - d_i}{p_i} < 1 \\ \Rightarrow & & 0 \leq 1 + \frac{t_d - r_{i,N_i} - d_i}{p_i} < 1\, because\, 0 < \frac{d_i}{p_i} \leq 1 \end{aligned} \tag{35}$$

Equation (35) shows that $0 \leq 1 + \frac{t_d - r_{i,N_i} - d_i}{p_i}$. Therefore we can use Equation (26) to get

$$f\left(1 + \frac{(t_d - r_{i,N_i}) - d_i}{p_i}\right) = \left\lfloor 1 + \frac{(t_d - r_{i,N_i}) - d_i}{p_i} \right\rfloor$$

From Equation (35).Therefore we know that $0 \leq 1 + \frac{t_d - r_{i,N_i} - d_i}{p_i} < 1$. Therefore

$$D[r_{i,N_i}, t_d) = \left\lfloor 1 + \frac{t_d - r_{i,N_i} - d_i}{p_i} \right\rfloor = 0 \cdot e_i = 0 \tag{36}$$

Substituting Equations (33) and (36) in Equation (32) we get

$$D[t_0, t_d) = \sum_{i=1}^{n} (N_i - 1) \cdot e_i \tag{37}$$

The processor time capacity is given by Equation (14) which reduces to Equation (38) because $\forall i,\ 0 < i \leq n,\ d_i \leq p_i$ and $t_d - r_{i,N_i} < min(d_i, p_i)$.

$$\rho = \sum_{i=1}^{n} \left((N_i - 1) + \left(\frac{t_d - r_{i,N_i}}{min(d_i, p_i)} \right) \right) \cdot e_i \tag{38}$$

Substituting the processor time capacity from Equation (38) and the upper bound on demand from Equation (37) in Equation (31) we get

$$\sum_{i=1}^{n} \left((N_i - 1) + \left(\frac{t_d - r_{i,N_i}}{min(d_i, p_i)} \right) \right) \cdot e_i < \sum_{i=1}^{n} (N_i - 1) \cdot e_i \tag{39}$$

$min(d_i, p_i) = d_i$ because $d_i \leq p_i$, $0 < t_d - r_{i,N_i} < d_i$ by the definition of this case, therefore $0 < \frac{t_d - r_{i,N_i}}{min(d_i, p_i)} < 1$. Therefore

$$\sum_{i=1}^{n} \left((N_i - 1) + \left(\frac{t_d - r_{i,N_i}}{min(d_i, p_i)} \right) \right) \cdot e_i > \sum_{i=1}^{n} (N_i - 1) \cdot e_i$$

Which contradicts Equation (39).

Case 2: Lets assume that $\forall i,\ 0 < i \leq n,\ d_i \leq p_i$ and $t_d - r_{i,N_i} \geq min(d_{i,N_i}, p_i)$. As in Case 1 because $d_i \leq p_i$, no job $J_{i,j}$ will demand any processor time beyond $r_{i,j} + p_i,\ \forall i, j,\ 0 < i \leq n, 0 < j \leq N_i$. Because $t_d - r_{i,N_i} \geq min(d_i, p_i)$ we have distinct $N_i \cdot p_i$ intervals inside $[t_0, t_d)$. Therefore

$$\begin{aligned} Demand\, in\, [t_0, t_d) &= Demand\, in\, [r_{i,j}, r_{i,j} + p_i)\, Intervals \\ &= \sum_{i=1}^{n} N_i \cdot Demand\, in\, [r_{i,j}, r_{i,j} + p_i)\, by\, T_i \end{aligned} \tag{40}$$

As in Case 1 because we are only concerned with the maximum demand that can occur, we can use the upper bound on the demand. Therefore

$$D[t_0, t_d) = \sum_{i=1}^{n} N_i \cdot D_{T_i}[r_{i,j}, r_{i,j} + p_i) \tag{41}$$

$D_{T_i}[r_{i,j}, r_{i,j} + p_i)$ is given by Equation (33), Substituting Equation (33) in Equation (41) we get

$$D[t_0, t_d) = \sum_{i=1}^{n} N_i \cdot e_i \tag{42}$$

The processor time capacity is given by Equation (14) which reduces to Equation (43) because $\forall i,\ 0 < i \leq n,\ d_i \leq p_i$ and $t_d - r_{i,N_i} \geq min(d_{i,N_i}, p_i)$.

$$\rho = \sum_{i=1}^{n} N_i \cdot e_i \tag{43}$$

Substituting the processor time capacity from Equation (43) and demand from Equation (42) in Equation (31) we get

$$\sum_{i=1}^{n} N_i \cdot e_i < \sum_{i=1}^{n} N_i \cdot e_i$$

A contradiction.

Case 3: $d_i \geq p_i$ In cases 1 and 2 of this proof we have proved that

$$\forall \tau > 0, \tau = [t_0, t_d),\ Processor\, time\, capacity \geq D[t_0, t_d)$$

for any task set with $d_i \leq p_i$. By corollary 5 we have proved that if a task set has $d_i > p_i$ with the same execution times as a task set with $d_i = p_i$ then

$$\forall \tau > 0, \tau = [t_0, t_d),\ D[t_0, t_d)\, by\, a\, task\, set\, with\, (d_i = p_i) \geq D[t_0, t_d)\, by\, a\, task\, set\, with\, (d_i > p_i)$$

Therefore

$$\forall \tau > 0, \tau = [t_0, t_d),\ Processor\, time\, capacity \geq D[t_0, t_d)\, by\, a\, task\, set\, with\, (d_i > p_i)$$

which contradicts Equation (31).

General Case: Suppose that we have a mix of tasks, at any time instant the task set can be divided into sub task sets corresponding to one the previous cases. Let us define the task set S_i as the task set corresponding to proof case i. Let τ be the time interval $[t_{0,}t_d)$. If job J_d missed its deadline at t_d then the demand in $[t_{0,}t_d)$ must have been greater than the processor time capacity in $[t_{0,}t_d)$. That is

$$\begin{aligned} processer\, time\, capacity &< demand \\ \sum_{i=1}^{3} \rho\, for\, S_i &< \sum_{i=1}^{3} demand\, by\, S_i \end{aligned} \tag{44}$$

We know from mathematics that if $\forall x_i, y_i,\ x_i > y_i$ then $\sum_{i=1}^{n} x_i > \sum_{i=1}^{n} y_i$. We have already proven in the proof cases (1–3) that

$$\rho\, for\, S_i > demand\, by\, S_i, \forall S_i, 0 < i < 3$$

Therefore

$$\sum_{i=1}^{3} \rho\, for\, S_i > \sum_{i=1}^{3} demand\, by\, S_i$$

which contradicts Equation (44). □

Because we always have two switches per job for every job in the case of $d < p$, we can account for switching by adding the time for switching up and switching down to the execution time of the job, therefore, the new execution time for each task is given in Equation (45).

$$e_{new_i} = e_i + S_U + S_D \tag{45}$$

Where e_{new_i} is the execution time including the switching overhead: S_U is the overhead to switch the frequency up, S_D is the overhead to switch the frequency down. One more consideration here is the need to check if the time interval between d_i and p_i is greater than the switching overhead. If not the algorithm should not scale the frequency down.

4.2 Power Savings

The amount of power that can be saved depends on whether both frequency and voltage are scaled or frequency alone is scaled. Some processors, such as the Crusoe processor [5], have a feed back loop to scale voltage when the frequency is scaled. Other processors, such as the Rabbit processor [26], can operate on multiple voltage levels but cannot scale the voltage with frequency changes.

Equation (1) shows that power is linearly proportional to the frequency and quadratically proportional to the voltage. If the processor automatically scales the voltage when the frequency is scaled, then there will be a voltage level corresponding to each frequency level. Let α be the frequency-scaling factor and β be the voltage-scaling factor corresponding to α. From Equation (2) it is clear that the frequency and voltage are related, but the relation between α and β depends on the gate

threshold voltage V_T and the voltage itself, V. Equation (46)[23] shows the relation between α and β.

$$\alpha = \frac{(\beta V - V_T)^2}{\beta (V - V_T)^2} \tag{46}$$

The normalized power savings will be given by

$$Power\ Savings = \frac{P_{\max} - P_{GDVS}}{P_{\max}} \tag{47}$$

where P_{max} is the average power consumed by the processor operating at frequency f_{max} and P_{GDVS} is the average power consumed by the processor operating under the GDVS algorithm. The equations representing the power savings for GDVS are the same as the ones for the DVSST algorithm [23], therefore Theorem 3 of [23] still applies here and GDVS will save the maximum amount of power when only the frequency is scaled.

Theorem 3: *If only the frequency can be scaled and the task set is feasibly scheduled, then the processor will save the maximum possible amount of power under GDVS when all tasks execute with the their WCET.*

Proof: See [23]. □

In this paper we present another way to compare the power savings. Consider a case when we will operate the processor at full frequency when there is a task to execute to finish it as fast as possible and shut the processor down completely when there is no task to execute. We will refer to this method as the On_Off algorithm. Theorem 4 shows that GDVS saves at least the same amount of power as the On_Off method. Another reason not to use the On_Off method is that the cost of switching to a complete sleep mode in some processors is very high. Our implementation platform— the Rabbit 2000 processor— has this high switching overhead penalty if we switch to the idle state using the low power oscillator as explained in Section 5.

Theorem 4: *The power consumed under GDVS is less than power consumed under the On-Off algorithm if both the frequency and voltage are scaled, and equal to it if only the frequency is scaled.*

Proof: Under the On-Off algorithm the processor is operated at either maximum frequency $f_{\max}$ or zero frequency when it is off. Let us assume that the processor executes a task T for a period of

time τ with utilization U_τ over τ were it is on for a period of τ_{on} and off for a period of τ_{idle}. Then the average power consumed in τ is

$$P_{on\_off} = \frac{1}{\tau}(P_{idle}\tau_{idle} + P_{on}\tau_{on})$$

but $P_{idle} = 0$ because the processor is operating at zero frequency when it is idle

$$P_{on\_off} = \frac{P_{on}\tau_{on}}{\tau}$$

but $P_{on} = Cf_{\max}V_{\max}^2$ because the processor is operating at maximum frequency when it is on

$$P_{on\_off} = \frac{Cf_{\max}V_{\max}^2\tau_{on}}{\tau}$$

but $\tau_{on} = U_\tau\tau$

$$P_{on\_off} = \frac{Cf_{\max}V_{\max}^2U_\tau\tau}{\tau} = Cf_{\max}V_{\max}^2U_\tau = P_{\max}U_\tau$$

From [23] the power consumed by the processor if we scale both frequency and voltage is given by

$$P_{GDVS} = \frac{1}{\tau}\sum_{i=0}^{n} C\alpha_i\beta_i^2 f_{\max}V_{\max}^2 \cdot \tau_i \tag{48}$$

where α_i, β_i and τ_i are the frequency scaling factor, voltage scaling factor and scaling factor change interval respectively.

but $P_{\max} = Cf_{\max}V_{\max}^2$ therefore $P_{GDVS} = \frac{P_{\max}}{\tau}\sum_{i=0}^{n}\alpha_i\beta_i^2 \cdot \tau_i$

but $\forall i, 1 \le i \le n, \beta_i \le 1$

$\Rightarrow \quad \beta_i^2 \le 1$

$\Rightarrow \quad \tau_i\alpha_i\beta_i^2 \le \tau_i\alpha_i$ because $\tau_i\alpha_i \ge 0$

$\Rightarrow \quad \sum_{i=0}^{n}\alpha_i\beta_i^2\tau_i \le \sum_{i=0}^{n}\alpha_i\tau_i$

$\Rightarrow \quad \frac{1}{\tau}\sum_{i=0}^{n}\alpha_i\beta_i^2\tau_i \le \frac{1}{\tau}\sum_{i=0}^{n}\alpha_i\tau_i$ because $\frac{1}{\tau} \ge 0$

$\Rightarrow \quad \frac{1}{\tau}\sum_{i=0}^{n}\alpha_i\beta_i^2\tau_i \le U_\tau$ because $U_\tau = \frac{1}{\tau}\sum_{i=0}^{n}\alpha_i\,\tau_i$

$\Rightarrow \quad P_{\max}\frac{1}{\tau}\sum_{i=0}^{n}\alpha_i\beta_i^2\tau_i \le U_\tau P_{\max}$

$\Rightarrow \quad P_{GDVS} \le P_{on\_off}$

This is the general case where we scale both voltage and frequency. If we only scale frequency then $\beta = 1$, substituting $\beta = 1$ we get $P_{GDVS} = P_{on\_off}$. If the voltage is ever scaled down, then we get $P_{GDVS} < P_{on\_off}$ □

Theorem 2 gives a necessary and sufficient condition for schedulability under EDF with GDVS. Thus, GDVS does not affect the optimality of EDF scheduling for periodic and sporadic task sets. Theorem 3 shows that, in theory, GDVS is optimal with respect to power savings when only the frequency can be scaled and all tasks execute with their WCET. Theorem 4 shows that GDVS saves more power than just operating the processor at maximum frequency and shutting it off when there is no task to execute. However, in practice, it is much harder to achieve optimal power savings due to algorithm overhead and limited frequency levels supported by many processors. The next section discusses these implementation issues.

5. Implementation And Evaluation

The GDVS algorithm was implemented in a modified version of Jean Labrosse's μC/OS-II (Micro C/ OS-II) real time operating system [14]. The original version of μC/OS-II uses the RM algorithm to preemptively schedule up to 64 tasks. The modified version used in this study also supports EDF scheduling of up to 64K tasks [15]. Algorithm overhead was measured using a stand-alone Rabbit 2000 test board [26]. The actual power savings realized with GDVS is a function of the task set and the processor. The GDVS power savings where evaluated by both simulation and a specific real-time application, the Robotic Highway Safety Marker.

Section 5.1 describes frequency scaling in the Rabbit 2000. Section 5.2 presents slight modifications to the GDVS algorithm required in practice since currently available embedded processors have a limited number of frequency scaling levels. The overhead created by GDVS under EDF scheduling on the Rabbit 2000 is reported in Section 5.3. Section 5.4 describes the Robotic Highway Safety Marker and power savings realized for that application. Section 5.5 presents results for a simulated periodic task set with deadlines less than periods.

5.1 Frequency Scaling in the Rabbit 2000

There are two crystal oscillators built into the Rabbit 2000. The main oscillator accepts crystals up to a frequency of 29.4912 MHz and is used to derive the clock for the processor and peripherals. The low power clock oscillator requires a 32.768 kHz crystal, and is used to clock the watchdog timer, a battery backed time/date clock, and a periodic interrupt. The main oscillator can be shut down in a special low-power mode of operation, and the 32.768 kHz oscillator is then used to clock all the things normally clocked by the main oscillator.

The main oscillator can be doubled in frequency and/or divided by 8. If both doubling and dividing are enabled, then there will be a net frequency division by 4. Our model of the Rabbit 2000 has an 18.532 MHz main oscillator. Thus, there are four frequency levels available from the main oscillator: 18.532MHz, 9.266MHz, 4.633MHz and 2.3165MHz—which correspond to 100%, 50%, 25% and 12.5% of the maximum frequency. Since the maximum frequency at which we can operate the processor is 18.532 MHz and the low power mode frequency is 32.768 kHz, the idle-state scaling factor used by GDVS is $\alpha_{idle} = \frac{32.768kHz}{18.532MHz} = .00176$. In practice, the value of α_{idle} can be close to zero but never zero as assumed in the theoretical presentation of GDVS.

The Rabbit 2000 processor can operate at different voltages but it does not change the voltage level dynamically when the frequency level is changed. Thus, only the processor frequency will be scaled dynamically, which will result in a linear savings in average power as explained in Section 4.2.

5.2 Modifying GDVS for the Rabbit Processor

There are four non-idle scaling levels available on the Rabbit 2000, rather than the infinite number of levels often assumed in theory. Fortunately, the algorithm can be modified slightly to allow scaling the frequency to a discrete number of levels by rounding the value of α to the next upper scaling level. For example, if we have a processor with scaling levels 0.25, 0.5, 0.75, and 1.0 and the value of α_n at some point in time t as calculated by GDVS is 0.58, then the next upper scaling level to which we set α_n is 0.75.

Another challenge in implementing GDVS on the Rabbit 2000 is that serial communication baud rates cannot be derived from the low-power oscillator. Thus, α_{idle} = 0.00176 cannot be used with any application that requires serial communication. Since the wireless transceiver used in the Robotic Highway Safety Marker uses a serial interface to the processor, we use $\alpha_{idle} = \alpha_{\min} = 0.125$ so that the application will not lose communication with the other robots.

5.3 Algorithm Overhead

There are two primary sources of overhead created by GDVS: changing frequency levels and detecting when the frequency can be scaled. Changing the processor frequency from one level to another is (approximately) constant, and was measured on the Rabbit 2000 processor to be 120 μs per frequency change with the main oscillator.

The second source of overhead is largely dependent on how the algorithm detects when it is possible to scale the processor frequency. When a task is released, a check is made to see if the frequency needs to be increased (i.e., if the task $\in$ *TD*). A timer list is used to detect when it is possible to scale down the processor frequency. A timer is set when the task is released and cancelled if the task is released again before the timer expires. The processor frequency is scaled down by $e_i/min(d_i, p_i)$ whenever a timer expires for task T_i.

The timer list is implemented as a sorted linked list with no effort made to optimize list insertion since most applications that use the Rabbit 2000 have very few tasks; our application has only six tasks and the version of μC/OS-II that comes with the board only supports 64 tasks. Thus, insertion into a list of size n has cost $\mathrm{O}(n)$. The worst case occurs when an entry needs to be inserted at the end of the list. The list insertion time was measured for up to 512 tasks with random deadlines. For each list length from one to 512, the test was repeated a number of times equal to the list length with random timer values to be inserted. The insertion time was measured for each insertion and the average time of these values for each list length was recorded. The graph shown in Figure 6 plots the average timer list insertion time verses the number of tasks from 20 such experiments. Time is measured in terms of periodic clock ticks on the Rabbit 2000, which occur at a rate of 2kHz or one

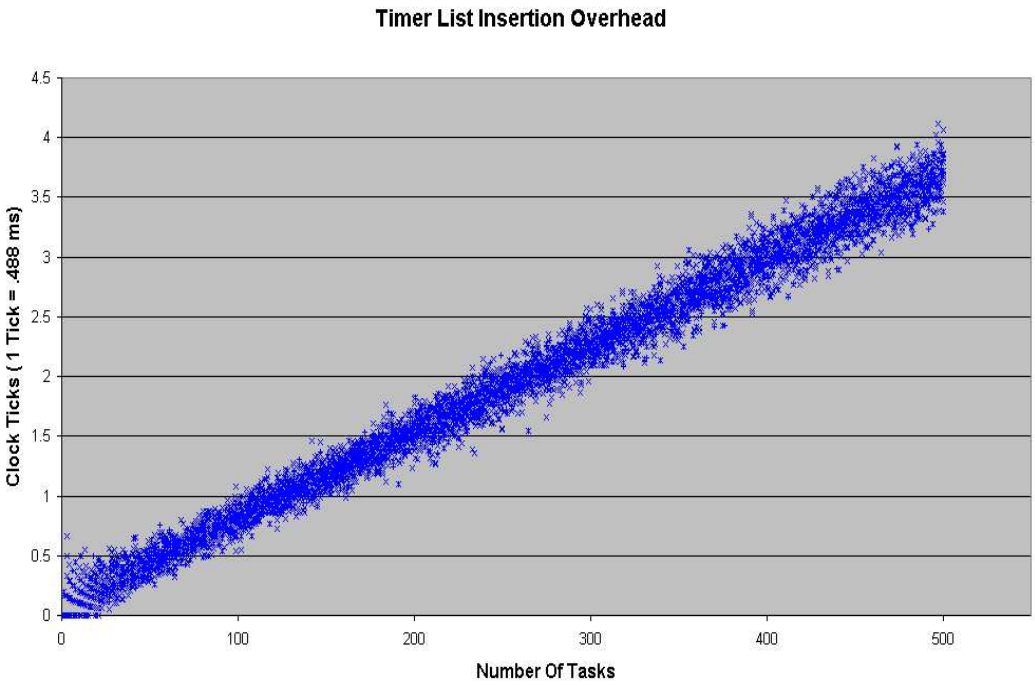


Figure 6. Timer list insertion overhead

clock tick every 488μS.

The average insertion time is less than 1 clock tick for a list with less than 125 entries, as shown in Figure 6. The insertion time is about 4 clock ticks (2 ms) for 512 entries. Clearly a more efficient implementation of the timer list should be used for large task sets.

5.4 Power Savings for a Robotic Highway Safety Marker

The Robotic Highway Safety Marker (RSM) is an automated safety devices designed to improve road construction work-zone design and safety. A RSM is a semi-autonomous mobile robot that carries a highway safety marker, commonly called a barrel. The RSMs operate in groups that consist of a single lead robot—called the foreman—and worker robots. To date, one foreman and six worker prototype RSMs have been developed. Each worker RSM has a Rabbit 2000 processor running our modified μC/OS-II. The prototype foreman is more sophisticated than the worker RSMs.

Control of the RSM group is hierarchical and broken into two levels—global and local control—to reduce the per-robot cost. The foreman robot performs global control. To move the robots, the foreman locates each RSM, plans its path, communicates destinations points (global waypoints),

and monitors performance. Local control is distributed to each individual RSM, which do not have knowledge of other robots and only performs local tasks.

The code for the RSM is implemented as a sporadic task set. The task set only executes after it receives a new waypoint from the foreman. A path from the initial position of the RSM to the new waypoint is computed as a parabola decomposed into multiple local waypoints. The number of local waypoints depends on the length of the path. The following six sporadic tasks comprise the RSM task set.

- **Serial Task:** reads commands from the foreman via a RF transceiver, converts the command to target destinations, and stores the destinations in a shared queue data structure.
- **Length Task:** calculates the path length, number of iterations, and other values for each target destination.
- **Waypoint Task:** calculates the desired wheel angles for each iteration of a PID control loop.
- **PID Task:** does the PID control for each iteration.
- **Encoder Task:** reads the current wheel angles.
- **Motor Task:** sends commands to each motor.

An abstract processing graph for this task set is shown in Figure 7. The precedence relations shown in Figure 7 represent the logical precedence constraints on the data processing and do not reflect actual release patterns. For example, to reduce latency in the processing graph, the last four nodes in the processing graph can be released simultaneously with deadline ties broken in favor of producer nodes, as described in [4]. The Serial task is released when data is available on the serial port. When data arrives, the Serial task converts it to a target destination, places it in a shared data structure and releases the Length task. Semaphores are not needed to synchronize access to the data structure, which results in a fully preemptable task set. The Length task calculates the first two local waypoints before the robot begins to move. As the robot moves to waypoint i, waypoint $i+2$ is computed. The design ensures that waypoint $i+2$ is computed before waypoint i is reached.

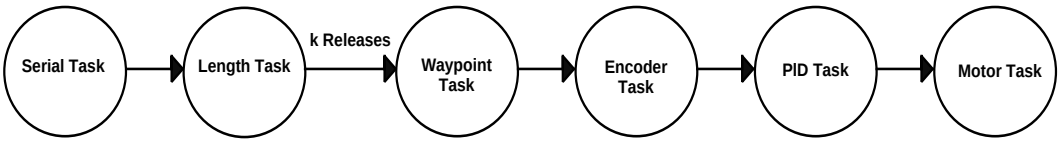


Figure 7. RSM processing graph.

This task set is modeled as a sporadic task set because the serial task receives commands with a minimum separation of 7.8125ms. The length task is executed the same number of times the serial task is executed. The number of times that Waypoint, PID, Encoder and Motor are executed depends on the number of local waypoints that need to be computed to reach the next global waypoint, which is dependent on the path length. Thus, for each execution of the serial task there may be a different number of executions for the Waypoint, PID, Encoder and Motor tasks. However, each task has a minimum separation period, as shown in Table 2.

The execution time for these tasks is very deterministic for two reasons. First the Rabbit 2000 has no cache memory, which eliminates memory-caching effects on execution time. Second the tasks repeat almost the same operation each time, with the exception of system initialization where some of the tasks execute a few more lines. Therefore the execution time of these tasks is usually very close to their WCET. The task execution times, shown in Table 2, were determined using an oscilloscope and free I/O pins on the processor.

Task	**Period**	**Execution Time**	$\boldsymbol{e_i}$ **/** $\boldsymbol{p_i}$
Serial	7.8125ms	100μs	.0128
Length	7.8125ms	1ms	.128
Way Point	3 $*$7.8125ms	2.5ms	.1066
Encoder	3 $*$7.8125ms	350μs	.0149
PID	3 $*$7.8125ms	1.06ms	.04522
Motor	3 $*$7.8125ms	250μs	.0106

Table 2. RSM sporadic task set parameters.

The maximum utilization for the task set is U= 0.31812, which occurs when all of the tasks execute in a periodic mode for an extended interval of time. If we have no idle periods over an extended interval of time, the lower bound on utilization is when we have only one execution of the

serial and length task followed by a very large number of executions of the other tasks. This will result in a processor utilization slightly greater than

$$\frac{e_{Waypoint}}{p_{Waypoint}} + \frac{e_{Encoder}}{p_{Encoder}} + \frac{e_{PID}}{p_{PID}} + \frac{e_{Motor}}{p_{Motor}} = .17732$$

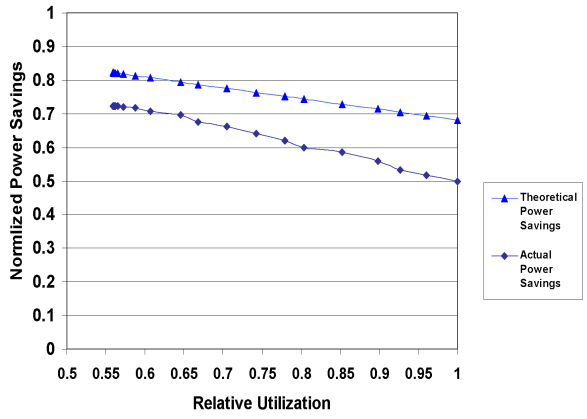
Depending on when commands arrive and the length of the path to be computed, a wide range of utilization values is possible. For any case, the theoretical maximum power savings will be $1 - U_\tau$ (as shown in Section 4.2), where U_τ is the utilization over the time interval τ. The actual power savings achieved is less because we cannot scale the frequency to the desired value; instead we scale it to the nearest upper level of frequency available on the Rabbit 2000, as described in Section 5.2.

As mentioned in Section 5.1, the Rabbit 2000 provides frequency scaling but does not directly adjust the voltage with the frequency. Thus, power savings can be linearly proportional to frequency scaling at best. However, since the Rabbit 2000 provides only a limited number of levels, rather than the unlimited number assumed in theory, there will be a difference between the actual savings and the theoretical power savings.

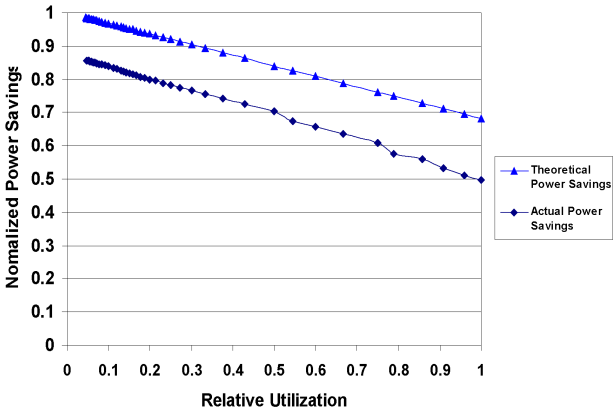
Figures 8(a) and 8(b) show the difference between the actual and the theoretical power savings. The normalized average power savings is plotted against relative utilization values, where the relative utilization is the ratio of a possible task utilization value to the maximum task utilization (0.31812). Figure 8(a) shows the normalized theoretical and actual power savings for the task set verses the relative utilization when there are no idle periods. That is, the robot is constantly moving but with destination commands of varying distance. In this case, the minimum relative utilization is 0.55739. Figure 8(b) shows the normalized theoretical and actual power savings when we have idle periods. That is, when the robot stops for intervals of time.

Note that the actual power savings deviate from a linear pattern even though only the processor frequency is scaled and the voltage remains constant. This is because when the frequency is scaled on the Rabbit 2000, it draws less current and the rate at which the current increases or decreases with each frequency level is not exactly linear.

The average ratio of the actual savings to the theoretical savings in both cases is about 83%. This means that GDVS achieved 83% of the theoretical power savings on the Rabbit 2000 for this



(a) Power savings with the robot constantly moving.



(b) Power savings with the robot not constantly moving.

Figure 8. Power Savings For The RSM

application.

If the task set were executed at a periodic rate, the GDVS would run the processor at a frequency equal to the task utilization, which is the same as the Static Voltage Scaling algorithm of [1, 22]. In this case GDVS will give the same power savings as the Static Voltage Scaling but with more overhead. Other DVS algorithms from the literature are unlikely to improve power savings much, even if the task set executes periodically, because they try to take advantage of the case when tasks do not execute with their WCET. In this application, however, task execution time is very deterministic and there is very little difference between average execution time and WCET.

5.5 Power Savings For Simulated Periodic Task Sets

The RSM application provided a good example of a sporadic task set with deadlines equal to periods, but since GDVS is a general algorithm, more evaluation for the power savings is needed. Two periodic task sets with deadlines less than periods were created. The task sets were simulated on the same implementation of GDVS used with the RSM application. The task sets consisted of five and ten tasks respectively. The task sets characteristics are summarized in the following points:

- Execution time e_i is the same for all tasks in the task set.

- $e_i = 10ms$ so that $e_i >> Switching\,to\,the\,low\,power\,oscillator\,Overhead$.
- p_i for tasks takes the values of 200 ms, 400ms and 800ms.
- The density of the task set is varied by changing the deadline of the tasks.

By changing the value of d_i we get different values of task set density ranging from .0975 to 1. Figure 9 shows normalized theoretical and actual power savings for the two task sets. The theoretical power savings for a periodic task set with deadlines less than or equal to periods is $1 - \sum_{i=0}^{n} \frac{e_i}{min(d_i,p_i)}$ when only frequency is scaled. Figure 9 shows that the actual power savings gets closer to the theoretical savings as the density increases. This is because the lower the density, the less the number of scaling levels we have; since we have only five levels on the Rabbit 2000 we have even fewer levels for lower densities. We also note that with a higher number of tasks in the task set (10 tasks) the power savings also decrease because the effect of the limited number of scaling factors becomes greater. It is clear from this result that implementing the algorithm on a processor that offers a larger number of frequency levels will improve the power savings.

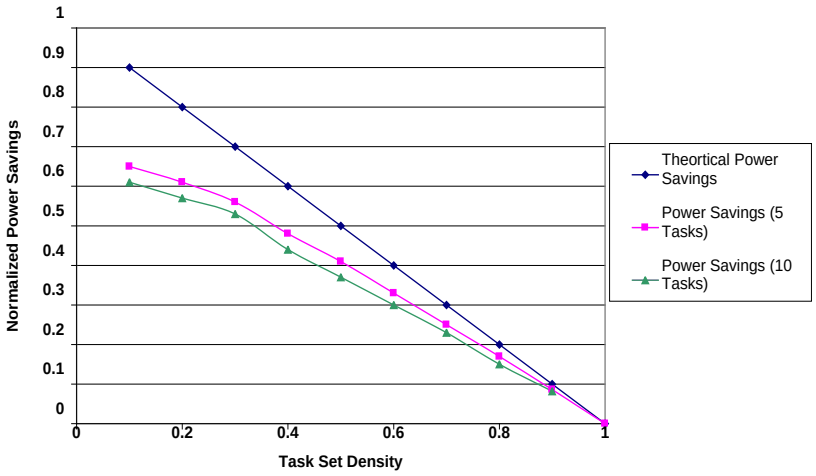


Figure 9. Power savings for a periodic task sets with $d_i \leq p_i$

6. Conclusion

A dynamic voltage-scaling algorithm called GDVS was presented for both periodic and sporadic task sets with no constraints on the deadlines executed under EDF scheduling. It was shown that schedulability under EDF is a necessary and sufficient schedulability condition for fully preemptive task sets to be scheduled under EDF with GDVS. GDVS is an inter-task DVS algorithm and the only attempt to save power when jobs execute for less than their WCET is to scale the processor to

a minimum frequency level whenever no jobs are pending. GDVS assumes that resources are not shared between tasks; DVS for resource-sharing sporadic tasks remains an open problem. GDVS is shown to be optimal when only the processor frequency is scaled and not the voltage. It is also shown that GDVS saves more energy than just switching the processor on and off. The optimality of GDVS when both the voltage and frequency are scaled remains a problem open for future research.

GDVS has been implemented in a modified version of μC/OS-II that supports EDF scheduling. GDVS was tested with a real-time application —The Robotic Highway Safety Marker— with a sporadic task set. GDVS was also tested with a simulated task set with deadlines less than periods. Both of these tests were run on the Rabbit 2000 processor. Results show differences between theoretical and actual savings are due to the limited number of frequency levels supported by the Rabbit 2000 processor.

References

[1] H. Aydin and R. Melhem and D. Mosse and P. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. *Proceedings of the 13th EuroMicro Conference on Real-Time Systems (ECRTS01)* June 2001.

[2] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt. A. Veidenbaum, and A. Nicolau. Profile-Based Dynamic Voltage Scheduling Using Program Checkpoints in the COPPER Framework. *Proceeding of Design, Automation and Test in Europe Conference (DATE)*, March 2002.

[3] Y. Doh and C. M. Krishna. EDF Scheduling Using Two-Mode Voltage-Clock-Scaling for Hard Real-time Systems. *Proc. of CASES 2001*, pp. 221-228, 2001.

[4] S. Farritor,M. Rentchler. Robotic Highway Safety Markers. *Proceedings of ASME International Mechanical Engineering Congress*, New Orleans, Louisiana, November 17-22, 2002.

[5] M. Fleischmann. Crusoe Processor Products and Technology, LongRun Power Management - Dynamic Power Management for Crusoe Processors. http://www.transmeta.com/pdf/white_papers/paper_mfleischmann_17jan01.pdf, Transmeta Inc., January 17, 2001.

[6] S. Goddard and K. Jeffay. Analyzing the Real-Time Properties of a Data flow Execution Paradigm using a Synthetic Aperture Radar Application. *Proc. 3^{rd} IEEE Real-Time Technology & Applications Symp.*, Montreal, Canada, pp. 60–71, June 1997.

[7] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power Optimization of Variable-Voltage Core-Based Systems. *IEEE Trans. Computer-Aided Design*, vol. 18, no. 12, pp. 1702-1714, Dec. 1999.

[8] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 178–187, December 1998.

[9] Intel XScale microarchitecture, http://developer.intel.com/design/intelxscale.

[10] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. *Proc. of ISLPED*, pp. 197–202, Aug. 1998.

[11] K. Jeffay. and S. Goddard A Theory Of Rate-Based Execution. *Proceedings of the 20th IEEE Real-Time Systems Symposium.*, Phoenix, AZ, pp. 304–314, December 1999.

[12] H. Kawaguchi, Y. Shin, and T. Sakurai. Experimental Evaluation of Cooperative Voltage Scaling (CVS): A Case Study. *Proceedings of IEEE Workshop on Power Management for Real-Time and Embedded Systems*, pp. 17-23, May 2001.

[13] W. Kim, J. Kim and S –L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. *Proceedings of Design Automation and Test in Europe (DATE'02)*, Paris, France, March 2002.
[14] J. Labrosse. *The Real Time Kernel MicroC/OS-II*, CMP Books, May 2002.
[15] C-M. Lee, Implementing Rate-Based Execution in MicroC/OS-II. Mater's Project, Dept. of CSE, University of Nebraska-Lincoln, November 27, 2002.
[16] Y.-H. Lee and C. M. Krishna. Voltage-Clock Scaling for Low Energy Consumption in Real-Time Embedded Systems. *Proceedings of the Sixth Int'l Conf. on Real Time Computing Systems and Applications*, pp. 272-279, 1999.
[17] C.L.Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, Vol.20, pp.46–61, 1973.
[18] J. Luo and N. K. Jha. Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems. *Proceedings of ICCAD*, pages 357–364, Nov 2000.
[19] A. Manzak and C. Chakrabarti. Variable Voltage Task Scheduling for Minimizing Energy or Minimizing Power. *Proceedings IEEE Int. Conf. on Acoustic, Speech, and Signal Processing (ICASSP'00)*, pp. 3239–3242, June 2000.
[20] A.K.-L. Mok. Fundamental Design Problems of Distributed Systems for the Hard Real Time Environment. Ph.D. Thesis,MIT, Dept. of EE and CS, MIT/LCS/TR-297, May 1983.
[21] D. Mosse, H. Aydin, B. Childers and R. Melhem, Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications. *Workshop on Compilers and Operating Systems for Low-Power (COLP'00)*, Philadelphia, PA, Oct. 2000.
[22] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. *Proc. of the 18th ACM Symp. on Operating Systems Principles*, 2001.
[23] A. Qadi, S. Goddard, and S. Farritor. Dynamic Voltage Scaling Algorithm for Sporadic Tasks, *Proceedings of the the 24rd IEEE Real-Time Systems Symposium*,Cancun, Mexico, December 2003., pp. 15-25.
[24] A. Qadi, S. Goddard, and S. Farritor. DVSST: A Dynamic Voltage Scaling Algorithm for Sporadic Tasks, University of Nebraska - Lincoln, Dept. of CSE, TR-CSE-UNL-2003-2, May 2003. Available via the Web: http://cse.unl.edu/∼goddard/Papers/TR-CSE-UNL-2003-2.pdf.
[25] G. Quan and X. Hu. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. *Proceedings of DAC'01: IEEE/ACM Design Automation Conference*, pp. 828-833, June 2001.
[26] Rabbit Semiconductors. Rabbit 2000 Microprocesser User's Manual, http://www.rabbitsemiconductor.com/documentation/docs/ manuals/ Rabbit2000/ UsersManual.
[27] D. Shin and J. Kim. A Profile-Based Energy-Efficient Intra-Task Voltage Scheduling Algorithm for Hard Real-Time Applications, *Proc. of ISLPED*, 2001.
[28] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design and Test Computers*, 18(2): 20–30, 2001.
[29] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. *Proceedings of the Design Automation Conference*, pp. 134–139, June 1999.
[30] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. *Proceedings of the International Conference on Computer-Aided Design*, pp. 365–368, November 2000.
[31] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 13–23, November 1994.
[32] W. Wolf. *Modern VLSI Design*, Prentice Hall Modern Semiconductor Design Series, Third Edition 2002.
[33] F. Yao, A. Demers, and S. Shenker. A Scheduling Model for Reduced CPU Energy. *IEEE Symposium on Foundations Computer Science*, pp. 374–382, Oct. 1995.
[34] F. Zhang and S. T. Chanson, Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptable Sections. *Proceedings of the 23rd IEEE International Real-Time Systems Symposium*, Austin, Texas, pp. 235–245, Dec. 2002.