Energy minimization techniques for real-time scheduling on multiprocessor platforms [*]

Shelby Funk[†] Joël Goossens[‡] Sanjoy Baruah[†]

October 1, 2001

Abstract

The scheduling of systems of periodic tasks upon multiprocessor systems with processors of differing speeds is considered. Computation speeds may differ because the system is comprised of different types of processors — *i.e.* it may be a uniform multiprocessor platform. Alternatively, the platform may be comprised of identical processors with speeds that differ because some processors may have their voltage levels reduced in order to minimize energy consumption. Given a periodic task system, a condition is derived for minimizing the computing capacity of the multiprocessor system while ensuring all deadlines of the task system are met when scheduled using the earliest deadline first (EDF) algorithm. A method of selecting the voltages of identical processors in order to meet this condition while minimizing the energy consumption is then presented. The resulting system is a platform of identical processors with different computation rates on which the task set can be scheduled using EDF.

Keywords: Multiprocessor; energy minimization; variable voltage; scheduling; earliest deadline first; periodic task system.

1 Introduction

Real-time problems may be divided into two basic categories: hard instances in which a job *must* execute completely between its designated arrival time and deadline, and soft instances in which the deadline may be missed under some conditions. Periodic tasks [8], sequences of jobs that execute repeatedly over regular time intervals, have proven to be extremely useful in describing real-time instances. The regularity of the periodic task instance allows for easy and concise analysis of the schedulability of a set of tasks. In this paper, we will be considering hard real-time instances consisting of both periodic tasks and (aperiodic) jobs.

Systems used to execute hard real-time instances must be designed to meet deadlines even when the requirements of the instance are most taxing. Often, this is achieved by over-provisioning system resources and much of the time the system is highly under-utilized. For

[*]Supported in part by the National Science Foundation (Grant Nos. CCR-9988327, and ITR-0082866).
[†]University of North Carolina, Chapel Hill, North Carolina.
[‡]Université Libre de Bruxelles, Brussels, Belgium.

this reason, analysts are often concerned with designing a system that will be guaranteed to meet all deadlines of a real-time instance while minimizing required system resources. The resources one chooses to minimize may vary depending on the focus of the problem. System computing capacity and power consumption are both elements of a system which are desirable to reduce as much as possible. System computing capacity is minimized to reduce the purchase cost and possibly to remain within current design limitations. Power consumption is minimized to reduce the energy usage (and therefore the operating cost) of a system, and possibly to increase portability or reduce size — decreasing the power requirement of a system may allow for a reduction in battery size or an increase in battery life [13].

Processor speed can be modified by altering the **clock frequency** (f_{clock} below) or by altering the **voltage** (V_{dd} below) [14]. The advantage of adjusting speed by adjusting voltage rather than clock frequency is that power consumption is proportional to the square of the voltage, whereas it is only linearly proportional to clock frequency. Furthermore, processor speed is proportional to f_{clock} and is nearly proportional to V_{dd}. For this reason, variable voltage processors are becoming more common and are used in many portable devices and other devices for which power consumption is a critical design issue [13].

In this paper, we introduce a method for minimizing the computing capacity of a system while ensuring that a real-time instance will meet all its deadlines. We then explore how to minimize the power consumption of a system with the speed constraints imposed by the schedulability problem. We allow the designer the flexibility of using a multiprocessor platform — a platform consisting of several processors all of which may execute any jobs of the real-time instance. We consider three types of multiprocessor platforms:

- In **identical multiprocessors** the rate of execution for a job is the same regardless of which processor is currently executing the job — *i.e.*, if a job runs for one time unit on one processor and then runs for one time unit on another processor, the same amount of work will be done in both cases. In this case, the speed of each processor is the same and is generally assigned the value 1.

- In **variable voltage identical multiprocessors** all processors have identical parameters, but their speeds may vary due to differences in voltage levels. If all processors are set to the same voltage level, they will all execute jobs at the same rate.

- In **uniform multiprocessors** the rate of execution for a job varies depending on which processor is currently executing the job — if a job runs for one time unit on one processor and then runs for one time unit on another processor, the amount of work done may differ. Uniform multiprocessors are represented by assigning a speed to each processor: if a job executes on a processor with speed s for t time units, then $s \cdot t$ units of work will be done.

In exploring the scheduling of hard real-time instances on multiprocessors, the following system requirements are assumed:

- **Job preemption is permitted.** If a job of the real-time instance is executing, it may be interrupted by another job. The job may then resume execution at the same execution point, without any time penalty, whenever the system becomes available again.

- **Job migration is allowed.** If a job is preempted while running on a processor, it may resume execution on another processor, also without penalty.
- **Job parallelism is not allowed.** While a job may execute on two separate processors at different points in time, it may not execute simultaneously on two separate processors.

This research. In this paper, we study the following problem:

Given a multiprocessor platform with m identical variable voltage processors and a collection of periodic tasks,

Determine the voltages that should be supplied to the processors such that

- all the jobs of the periodic tasks will meet their deadlines using the EDF scheduling algorithm, and
- energy consumption will be minimized.

Many real-time scheduling theorists have explored the problem of scheduling a real-time instance on identical multiprocessors [1, 2, 11, 12]. The problem of scheduling real-time instances on uniform multiprocessors is now beginning to be explored in detail [5].

2 Model

We wish to execute real-time job instances on uniform multiprocessors. A real-time instance is either a set of **jobs**, $J = \{j_1, \ldots, j_n\}$, or a set of **periodic tasks**, $\tau = \{\tau_1, \ldots, \tau_n\}$. Each job $j_i \in J$ is completely defined by the 3-tuple (a_i, c_i, d_i), where a_i is the job's arrival time, c_i is its execution requirement, and d_i its deadline. Periodic task sets are essentially sets of infinitely repeating real-time jobs. They are represented by 2-tuples (c_i, p_i), where c_i is the execution requirement of each job and p_i is its period: each job of this periodic task has an arrival time of $k \cdot p_i$, an execution requirement of c_i and a deadline of $(k+1) \cdot p_i$ for some $k = 0, 1, 2, \ldots$ The **utilization** of a task measures the proportion of a processor it will require. It is given by $u_i \stackrel{\text{def}}{=} \frac{c_i}{p_i}$. Similarly, the **total utilization** of a task set measures the proportion of a processor the entire set will require and is given by $U_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{c_i}{p_i}$.

We call J **feasible** if there exists some scheduling algorithm for which every job j_i receives c_i units of execution in the time interval $[a_i, d_i)$. A periodic task set is feasible if for every task τ_i and for every integer $k \geq 0$ the task τ_i receives c_i units of execution in the time interval $[k \cdot p_i, (k+1) \cdot p_i)$. In some cases, it may not be necessary to specify whether the real-time instance under discussion is a job set or a task set — it could be either. We refer to such instances by the symbol I, which is a (possibly infinite) set of jobs $I_j = (a_j, c_j, d_j)$. If an instance I is feasible on a multiprocessor platform π and A is an algorithm which schedules I to meet all its deadlines, then we call I A**-schedulable**.

This paper focusses on determining processor speeds for variable voltage identical multiprocessors that will ensure EDF-schedulability and minimize energy usage. The EDF, or "earliest deadline first," scheduling algorithm assigns jobs with earlier deadlines higher priority than those with later deadlines — if there are more jobs awaiting execution than there are processors, the jobs with the earlier deadlines will be assigned to processors and those with later deadlines will wait. Jobs with the same deadline are assigned priority arbitrarily, but consistently (perhaps using job identification to break such ties).

EDF is known to be an **optimal** scheduling algorithm in uniprocessor systems — a task set τ is feasible on a uniprocessor π if and only if it is EDF-schedulable on π. Unfortunately, EDF is *not* optimal on multiprocessors since there are tasks sets that are feasible on some multiprocessors but will miss deadlines if EDF is used [4]. There are nevertheless significant advantages to using EDF for scheduling on multiprocessor platforms. While it is beyond the scope of this paper to describe in detail these advantages, some important ones are listed below:

- Very efficient implementations of EDF have been designed (see, e.g., [10]).
- It can be shown that when a set of jobs is scheduled using EDF, then the total number of *preemptions* is bounded from above by the number of jobs in the set.
- It can be similarly be shown that the total number of *interprocessor migrations* that of individual jobs is bounded from above by the number of jobs.

On uniform multiprocessors, there are a variety of choices available regarding which processor a job is assigned to. One possibility is to assume that higher priority jobs are always assigned to faster processors. This property, called the **work–conserving** property, implies that if a processor is idling then all processors with a slower speed are also idling. In this paper, we assume that EDF employs the work–conserving property when scheduling real-time instances on uniform multiprocessors. This variation of EDF for uniform multiprocessors also has two other important properties:

- It is **non-idling**. A job that has not completed its execution requirement will never be forced to wait for a processor while there is some processor that is not executing any job.
- It is **deterministic**. The scheduling algorithm generates a unique schedule for any given instance I.

It should be noted that if the work–conserving property is not assumed, EDF will not necessary satisfy the deterministic property on uniform multiprocessors — if two jobs running on different speed processors were switched, the work–conserving property would be violated, but all other properties of EDF would be satisfied. EDF will satisfy the non-idling property even if it is not assumed to be work conserving.

3 Definitions

We begin this section with some basic notation used to describe a multiprocessor platform π.

Definition 1 **($m(\pi), s_i(\pi), S(\pi), \lambda(\pi)$)** Let π be an m-processor multiprocessor platform with processor speeds $s_1, \ldots, s_m$, where $s_i \geq s_{i+1}$ for $i = 1, \ldots, m-1$. The constants $m(\pi), s_i(\pi), S(\pi)$, and $\lambda(\pi)$ are defined as follows:

- $m(\pi)$ denotes the number of processors in π: $m(\pi) \stackrel{\text{def}}{=} m$.
- $s_i(\pi)$ denotes the speed of the i^{th} fastest processor of π: $s_i(\pi) \stackrel{\text{def}}{=} s_i$.
- $S(\pi)$ denotes π's cumulative processing power: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi)$.
- We define an additional parameter $\lambda(\pi)$ as follows: $\lambda(\pi) \stackrel{\text{def}}{=} \max_{k=1}^{m(\pi)} \frac{\sum_{i=k+1}^{m(\pi)} s_i(\pi)}{s_k(\pi)}$.

Intuitively, the parameter $\lambda(\pi)$ measures the degree of "identicalness" of π — the closer π is to being an identical system, the larger the value of $\lambda(\pi)$. This value has an upper bound of $(m-1)$, which occurs when π consists of m identical processors. Alternatively, $\lambda(\pi)$ can be arbitrarily small when the processors have extremely different speeds. For example, $\lambda(\pi) < \epsilon$ for the m-processor platform where $s_{i+1}(\pi) \leq \frac{\epsilon}{m} \cdot s_i(\pi)$ for $i = 1, \ldots, m-1$.

Note that the above definition implies that $s_i(\pi)$ is non-increasing for any multiprocessor platform π — *i.e.,* for all multiprocessor platforms π, $s_i(\pi) \geq s_{i+1}(\pi)$ for all $i = 1, \ldots, m(\pi) - 1$.

The following definition provides a notation for indicating when an algorithm assigns tasks of an instance to a specific processor.

Definition 2 **($\delta(A, s_i(\pi), I, t), \delta(A, s_i(\pi), I_k, t)$)** Let A be any deterministic scheduling algorithm, π be any multiprocessor, and I any real-time instance. Then at any time t and for any $i = 1, \ldots, m(\pi)$, we have

$$\delta(A, s_i(\pi), I, t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s_i(\pi) \text{ is idling at time } t \text{ when } A \text{ schedules } I \text{ on } \pi \\ 1 & \text{if some job is executing on } s_i(\pi) \text{ at time } t \text{ when } A \text{ schedules } I \text{ on } \pi. \end{cases}$$

Similarly, to indicate whether a specific job of I is executing on a processor at a given time, the following notation is used:

$$\delta(A, s_i(\pi), I_k, t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } I_k \text{ does not execute on } s_i(\pi) \text{ at time } t \text{ when } A \text{ schedules } I \text{ on } \pi \\ 1 & \text{if } I_k \text{ executes on } s_i(\pi) \text{ at time } t \text{ when } A \text{ schedules } I \text{ on } \pi. \end{cases}$$

In order to determine whether a real-time instance is feasible using a particular algorithm, the work done by the algorithm over time must be analyzed. The following notation is used to denote this concept.

Definition 3 $(W(A,\pi,I,t), W(A,\pi,I_k,t))$ Let π be any multiprocessor platform and let I be any real-time instance. The notation $W(A,\pi,I,t)$ denotes the work done by algorithm A on instance I using multiprocessor platform π in the time t. More formally:

$$W(A,\pi,I,t) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi) \int_0^t \delta(A, s_i(\pi), I, x)dx.$$

Similarly, the notation $W(A,\pi,I_k,t)$ denotes the work done by algorithm A on the k^{th} job of I:

$$W(A,\pi,I_k,t) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi) \int_0^t \delta(A, s_i(\pi), I_k, x)dx.$$

4 Uniform multiprocessor speeds for EDF-schedulability

Philips, Stein, Torng and Wein [12] explored in detail the idea of augmenting an identical multiprocessor platform in order to ensure EDF-schedulability. They proved that if a given real-time instance is feasible on a platform of m identical processors, then the same instance is EDF-schedulable on m identical processors that are $(2 - \frac{1}{m})$ times faster than the original system. In a previous paper, we extended this result to uniform multiprocessors: Given that a real-time instance is feasible on some uniform multiprocessor platform with m processors, the system may be augmented to ensure EDF-schedulability on another (faster) m-processor multiprocessor platform. We show below that the augmented platform need not be restricted to having the same number of processors as the original system.

We begin by introducing a lower bound on the amount work done by a work–conserving algorithm on an augmented system. This Lemma is a minor modification of Lemma 1 in our previous paper [5]. The modification accounts for the the possibility that ψ and π are not necessarily comprised of the same number of processors. The proof is essentially the same and is omitted from this paper.

Lemma 1 [5] Let I be any real-time instance. Let π and ψ be two multiprocessor platforms. Let A be any algorithm executing on π and let $\widetilde{A}$ be any work–conserving algorithm executing on ψ. Assume the speeds of π and ψ are related as follows:

$$S(\psi) \geq S(\pi) + \lambda(\psi) \cdot s_1(\pi). \tag{1}$$

Then at any time $t \geq 0$, the following condition holds:

$$W(\widetilde{A}, \psi, I, t) \geq W(A, \pi, I, t). \tag{2}$$

Lemma 1 relates the total work done by two platforms under specific conditions. Unfortunately, relating the total work does not necessarily provide enough information to guarantee that I does not miss deadlines on ψ using algorithm $\widetilde{A}$ even if I is feasible on π. The following theorem uses the previous result to conclude that when the conditions of Lemma 1 hold, the real-time instance I will be EDF-schedulable on ψ. While this theorem differs from Theorem 1 of [5] in that π and ψ do not necessarily have the same number of processors, the proof is exactly the same. The reader is encouraged to consult that paper for the proof.

Theorem 2 [5] Let I be a real-time instance that is feasible on multiprocessor platform π using algorithm A. Let ψ be another multiprocessor platform. If the following condition holds

$$S(\psi) \geq S(\pi) + \lambda(\psi) \cdot s_1(\pi) \tag{3}$$

then I is feasible on ψ using the EDF scheduling algorithm.

While Theorem 2 provides conditions under which instance I is EDF-schedulable, a platform π on which I is feasible must be provided in order for the theorem to be useful. The following corollary provides such a platform for any task set τ and therefore provides more general conditions under which τ is feasible on ψ.

Corollary 3 Let τ be a set of n real-time tasks with utilizations $u_1, \ldots, u_n$, where $u_1 = \max_{i=1}^{n} u_i$ and $U_n = \sum_{i=1}^{n} u_i$. Let ψ be a multiprocessor platform that satisfies the following property:

$$S(\psi) \geq U_n + \lambda(\psi) \cdot u_1. \tag{4}$$

Then τ is EDF-schedulable on ψ.

Proof. Let π be the n-processor multiprocessor platform with $s_i(\pi) = u_i$. Schedule τ on π by executing each task τ_i of τ on processor $s_i(\pi)$ of π. Using this algorithm, all deadlines of τ are met on π. Note that $s_1(\pi) = u_1$ and $S(\pi) = U_n$. Therefore, Inequality 4 holds and τ is feasible on ψ, the result follows from Theorem 2. ∎

Inequality 4 gives a sufficient condition to ensure EDF-schedulability, but not a necessary one: there are task sets which are EDF-schedulable on platforms ψ with cumulative speed smaller than $\lambda(\psi) \cdot u_1 + U_n$. In fact, if $\tau = \{(1,1),(1,1)\}$ it can be scheduled on a platform ψ consisting of 2 unit-speed processors. In this case $\lambda(\psi) = 1$. Therefore, $\lambda(\psi) \cdot u_1 + U_n = 1 \cdot 1 + 2 = 3$ so Inequality 4 is clearly violated. However, the bound is a tight one: in some cases violating this bound by even a small amount can result in a system on which I is not EDF-schedulable, as the following theorem shows.

Theorem 4 There exists a periodic task set $\tau = \{\tau_1, \ldots, \tau_n\}$ and a multiprocessor platform ψ and with cumulative speed given by

$$S(\psi) = \lambda(\psi) \cdot u_1 + U_n - \epsilon \tag{5}$$

(where ϵ is some arbitrarily small positive number) such that τ is not feasible on ψ.

Proof. Let $\tau = \{(1,1),(1,1)\}$ and let $\eta \overset{\text{def}}{=} \frac{\epsilon}{2}$. Consider the platform ψ consisting of two processors with speeds $\frac{2-\eta}{1+\eta}$ and $\frac{\eta\cdot(2-\eta)}{1+\eta}$. Since the cumulative speed of ψ is $(2-\eta)$, ψ will provide less capacity in the interval $[0,1)$ than the 2 units required by τ. Therefore at least one of the tasks of τ must miss its first deadline. Note that

$$\lambda(\psi) \cdot u_1 + U_n - \epsilon = \eta + 2 - \epsilon = \frac{\epsilon}{2} + 2 - \epsilon = 2 - \eta = S(\psi).$$

Therefore Equation 5 is satisfied. ∎

5 Energy minimization

The previous section provides a method for verifying whether ψ can be used to schedule a periodic task set using an EDF scheduler. In this section, we will determine the speed settings for ψ that will minimize energy usage $E(\psi)$. For each processor of ψ, the energy usage is the integral of the power consumption over time [14]:

$$E \overset{\text{def}}{=} \int P dt.$$

The power consumption of each processor is given[1] by the following equation [15]:

$$P = \alpha C_L V_{dd}^2 f_{clock}, \tag{6}$$

where α is the switching activity, C_L the load capacitance, V_{dd} the voltage and f_{clock} is the clock frequency. The processor speed is also a function of f_{clock}, V_{dd} and C_L:

$$S = \frac{f_{clock}}{kC_L} \cdot \frac{(V_{dd} - V_T)^2}{V_{dd}}, \tag{7}$$

where V_T is the threshold voltage, k is a constant. Note that if setting the processor speed is achieved by reducing the voltage, there would be a resulting quadratic gain in power. By contrast, if speed were set by adjusting the clock frequency, the power gain would be linear. In this section, we introduce a method for finding m voltage settings, $V_{dd_1}, \ldots, V_{dd_m}$, so that a given periodic task set τ will be EDF-feasible on m processors and the total power consumption will be minimized.

We assume that we are setting the voltage levels for m identical variable voltage processors — *i.e.*, the values of the switching activity (α), the load capacitance (C_L), the clock frequency (f_{clock}), the threshold voltage (V_T) and the constant k in Equation 7 are the same for all m processors of the platform. We vary processor speed by setting the voltages to different levels.

Notice that once the voltage levels are determined, they remain static — *i.e.* Equation 6 remains constant. Therefore, in this context minimizing energy usage is equivalent to minimizing the power consumption. From this point forward, we will discuss power minimization with the understanding that minimizing power consumption also minimizes energy usage.

We wish to minimize Equation 6, the power equation, subject to a number of constraints. Firstly, we must ensure that the speeds are non-increasing and not smaller than V_T, giving the following linear constraints:

$$V_{dd_1} \geq V_{dd_2} \geq \ldots \geq V_{dd_m} \geq V_T. \tag{8}$$

Furthermore, we need to apply a non-linear constraint on the total speed to ensure that $S(\psi) \geq \lambda(\psi) \cdot u_1 + U_n$. This constraint is not straightforward and requires more in-depth analysis. The complication arises primarily because of the $\lambda(\psi)$ term in the speed constraint.

[1] We restrict our study to a simple but popular "electrical" model, we believe that this model is realistic and that our method is not restricted to that model and can be adapted to more sophisticated one (if needed).

Recall that $\lambda(\psi) = \max_{k=1}^{m(\psi)-1} \frac{\sum_{j=k+1}^{m(\psi)} s_j(\psi)}{s_k(\psi)}$. In the case of two processors, the maximum must occur when $k = 1$ (since that is the only possible value for k), thus $\lambda(\psi) = \frac{s_2(\psi)}{s_1(\psi)}$. For a value of $m(\psi)$ larger than 2, the index k that is used to evaluate $\lambda(\psi)$ cannot be known unless the processor speeds are known. Since we are trying to solve for the speeds that optimize energy consumption, the solution to the optimization varies depends on the value of k that we use. Consequently, we solve this problem $(m(\psi) - 1)$ times — once for each possible value of k — and select the solution that gives the smallest power consumption.

The following equation is the speed constraint for $m(\psi)$ processors for a given value of k:

$$\sum_{j=k+1}^{m(\psi)} \left(\frac{(V_{dd_j} - V_T)^2}{V_{dd_j}}\right) \cdot \frac{V_{dd_k}}{(V_{dd_k} - V_T)^2} \cdot u_1 + U_n - k_S \cdot \left(\sum_{i=1}^{m} \frac{(V_{dd_i} - V_T)^2}{V_{dd_i}}\right) \leq 0, \tag{9}$$

where $k_S \stackrel{\text{def}}{=} \frac{f_{clock}}{kC_L}$. In addition, for the given value of k, we must ensure that $\lambda(\psi)$ is the maximum of all possible $\lambda(\psi)$ values. This adds another $m(\psi) - 2$ nonlinear constraints:

$$\sum_{j=i+1}^{m(\psi)} \left(\frac{(V_{dd_j} - V_T)^2}{V_{dd_j}}\right) \cdot \frac{V_{dd_i}}{(V_{dd_i} - V_T)^2} - \sum_{j=k+1}^{m} \left(\frac{(V_{dd_j} - V_T)^2}{V_{dd_j}}\right) \cdot \frac{V_{dd_k}}{(V_{dd_k} - V_T)^2} \leq 0 \tag{10}$$

for each $1 \leq i < m$ where $i \neq k$.

In summary, we wish to minimize Equation 6 subject to the non-linear constraints 9, 10 and the linear constraint 8. This is an optimization problem with a quadratic objective function and non-linear constraint functions. The operations research literature gives a large panel of solutions to this problem including the Sequential Quadratic Programming (SQP) method. We shall not give details here, it is not the subject of this paper (see for instance [6] for details). To resume, we solve $m(\psi) - 1$ optimization problems within a loop as follows:

```
$P_{sav} = \infty$
for $k = 1$ to $m(\psi) - 1$
    $[P, \vec{V}] = \mathsf{opt}(k)$
    if $P < P_{sav}$ then
        $[P_{sav}, \vec{V}_{sav}] = [P, \vec{V}]$
next
return $[P_{sav}, \vec{V}_{sav}]$
```

where $\mathsf{opt}(k)$ is an SQP function that minimizes Equation 6 subject to constraints 8, 9, and 10 for each index k. The minimal power consumption (P) and the associated voltage $(\vec{V})$ are returned by opt on each iteration. The overall minimum power consumption (P_{sav}) determines the voltage levels that are selected $(\vec{V}_{sav})$.

6 Evaluation

In this section, we introduce experimental results. We executed the optimization technique described above on randomly generated task sets. The optimizations were done for 4 processors and for 8 processors. In all cases, we assume the system parameters have the following

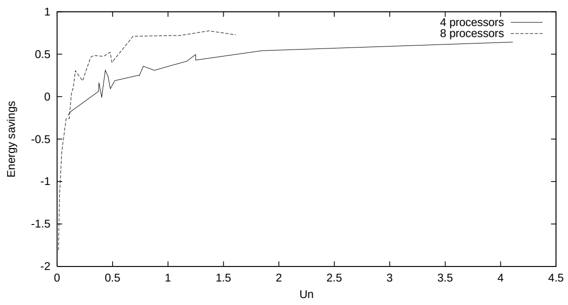

Figure 1: Energy savings for uniprocessors systems.

values (which are within normal ranges):

$$\alpha = 0.3,\ C_L = 1\ \mu\text{F},\ f_{clock} = 450\ \text{MHz},\ V_T = 0.5\ \text{V},\ k_S = 0.3667.$$

For this method, we only need to determine the utilization, u_1, of task τ_1 with the heaviest utilization, and the aggregate task set utilization U_n. The value u_1 is set to a random number in the range (0.0, 1.0). In determining the value of U_n, half of the values are initialized in the range $(u_1, m \cdot u_1)$ and half in the range $(m \cdot u_1, \infty)$. We compared the results to the power consumption of uniprocessors as well as identical multiprocessors.

The speed of each uniprocessor was set to U_n since any task set is EDF-schedulable if its utilization is at most 1 [8]. To find the identical multiprocessor speed settings we employ Corollary 3 of this paper. By this corollary, the randomly generated task set will be EDF-schedulable on any multiprocessor ψ if $S(\psi) \geq U_n + \lambda(\psi) \cdot u_1$. Since we want ψ to be an *identical* multiprocessor, the value of $\lambda(\psi)$ is clearly $m - 1$. Therefore the speed of each processor is set to $\frac{U_n+(m-1)\cdot u_1}{m}$. Once the speed is determined, the voltage levels can easily be found using Equation 7. These levels are then used to determine the power consumption of the constructed uniprocessor and identical multiprocessor systems. Power savings are calculated the finding the difference between the power usage of the constructed systems and the variable voltage system and then dividing this difference by power consumption of the constructed systems — *i.e.*, we calculate the savings of the variable voltage systems relative to the constructed systems. Recall that since the processor speeds are constant, the savings in power are equivalent to the energy savings.

We note that since the SQP method finds a *local* minimum, runs executed with different starting points may result in different power consumption levels. This is similar to the way that Newton's method may give different results when different starting points are provided. For these experiments, the same problem was submitted with various starting points and the result with the lowest power consumption was selected.

Figure 1 illustrates the energy savings of the variable voltage system compared to uniprocessor systems. When the value of U_n is small, the uniprocessors consume significantly less energy. This is because small levels of voltage consume a disproportionate amount of energy — in the most extreme case, when $V_{dd} = V_T$ the processor has a speed of 0 while it is still consuming energy. For task sets that have a very small total utilization, uniprocessors would be preferable to variable voltage multiprocessors from an energy conservation point of view.

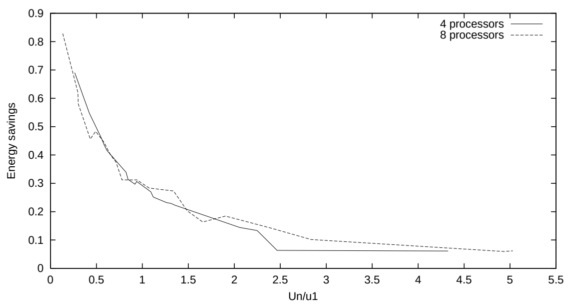

Figure 2: Energy savings for identical multiprocessors systems.

However, when the total utilization increases, the use of variable voltage multiprocessors can conserve a significant amount of energy. As the speeds increase, the voltage levels get large enough that Equation 7 is closer to linear. This allows the quadratic savings in the power equation to be more effectively exploited.

Figure 2 illustrates the energy savings of an m-processor variable voltage system as compared to an identical multiprocessor system, also comprised of m processors. In this case, the savings correlate with the ratio of the total utilization, U_n, to the heaviest utilization, u_1. We then normalized this ratio by the number of processors, m. When U_n is dominated by u_1, the variable voltage system consumes much less energy than the identical multiprocessor system. Note that the fastest processor must have a speed of at least u_1. Otherwise τ_1 will not be able to meet its deadline even if it executes on the fastest processor continually. Therefore, if the utilization of τ_1 is a very large proportion of the total utilization, the processor speeds may be highly variable. As the total utilization becomes significantly larger in relation to u_1, the power consumption approaches the consumption of the identical system. In this case, the power consumption can be reduced by evenly distributing the system speed. By Equation 6, power consumption is a constant times the sum of the squares of the voltages. Given that the sum of squares is minimized when values are identical, the power consumption of the system may be reduced when the voltages (and therefore the speeds) approach equality. Of course, in this case the value of λ is $m - 1$, its maximum possible value, and so using an identical system is only beneficial when u_1 is small relative to U_n.

These results demonstrate that, under certain circumstances, the variable voltage method described in this paper can result in significant energy savings.

7 Future Work

We are particularly interested in two specific areas of exploration based on the results of this paper. First, while we have minimized power consumption for static voltage settings, there could be more savings gained if we take advantage of dynamic-voltage processors. The problem of scheduling of periodic task sets on dynamic-voltage *uni*processors has already been considered [7, 9, 3]. We wish to extend those results to dynamic-voltage multiprocessors using techniques considered in this paper.

Second, we note that assuming EDF satisfies the work-conserving property on uniform multiprocessors could result in a large number of job migrations. We would like to find a similar result for an another version of EDF that does not assume the work-conserving property (perhaps considering a less stringent property instead) that may require fewer job migrations. Furthermore, it seems that a faster system would require fewer migrations. We would like to explore in more detail the relationship between system speed and job migrations.

8 Conclusion

Real-time task sets can often require a system to be over-provisioned in order to ensure that all deadlines are met. This is especially true for *hard* real-time systems, where missing a deadline may result in catastrophic results. In this case, systems must remain stable even when the task set is in its most demanding state. If the worst case occurs infrequently the system is often under-utilized and may consume more energy than necessary. Therefore, it is beneficial to consider the energy usage of a system in the design, while still ensuring that the task set will meet all deadlines.

The advent of variable voltage processors allows the designer to have more control over the energy consumption of the system. In this paper, we have introduced a method of employing variable voltage processors in system design to ensure EDF-schedulability of periodic task sets while reducing power consumption. We have shown that for certain task sets this method can conserve a considerable amount of energy as compared to uniprocessors and identical multiprocessors.

References

[1] Anderson, J., and Srinivasan, A. Early release fair scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Stockholm, Sweden, June 2000), IEEE Computer Society Press, pp. 35–43.

[2] Aydin, H., Mejia-Alvarez, P., Melhem, R., and Mosse, D. Optimal reward-based scheduling of periodic real-time tasks. In *Proceedings of the Real-Time Systems Symposium* (Phoenix, AZ, December 1999), IEEE Computer Society Press.

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

[4] Dhall, S. K., and Liu, C. L. On a real-time scheduling problem. *Operations Research 26* (1978), 127–140.

[5] Funk, S., Goossens, J., and Baruah, S. On-line scheduling on uniform multiprocessors. In *Proceedings of the* 22^{nd} *Real-Time Systems Symposium* (London, England, December 2001), IEEE Computer Society Press.

[6] Gill, P. E., Murray, W., and Wright, M. H. *Practical Optimization.* Academic Press, London, 1981.

[7] Hong, I., Qu, G., Potkonjak, M., and Srivastava, M. Synthesis techniques for low-power hard real-time systems on variable-voltage processors. In *Proceedings of the* 19^{th} *Real-Time Systems Symposium* (Madrid, Spain, December 1998), IEEE Computer Society Press, pp. 178–187.

[8] Liu, C., and Layland, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[9] Manzak, A., and Chakrabarti, C. Variable voltage task scheduling for minimizing energy or minimizing power. In *IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP'00)* (June 2000), pp. 3239 –3242.

[10] Mok, A. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems* (Washington D.C., May 1988), pp. 42–46.

[11] Mok, A. K. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment.* PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[12] Phillips, C. A., Stein, C., Torng, E., and Wein, J. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, 4–6 May 1997), pp. 140–149.

[13] Pouwelse, J., Langendoen, K., and Sips, H. Energy priority scheduling for variable voltage processors. In *Int. Symposium on Low Power Electronics and Design (ISLPED'01)* (Huntington Beach, CA, August 2001).

[14] Yao, F., Demers, A., and Shenker, S. A scheduling model for reduced CPU energy. *36th IEEE Symposium on Foundations of Computer Science* (Oct. 1995), 374–382.

[15] You, Y.-P., Lee, C., Lee, J.-K., and Shih, W.-K. Real-time task scheduling for dynamically variable voltage processors. In *IEEE Workshop on Power Management for Real-Time and Embedded Systems* (May 2001), pp. 5–10.