Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors *

Gang Quan Xiaobo (Sharon) Hu
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556
gquan@cse.nd.edu

ABSTRACT

Energy consumption has become an increasingly important consideration in designing many real-time embedded systems. Variable voltage processors, if used properly, can dramatically reduce such system energy consumption. In this paper, we present a technique to determine voltage settings for a variable voltage processor that utilizes a fixed priority assignment to schedule jobs. Our approach also produces the minimum constant voltage needed to feasibly schedule the entire job set. Our algorithms lead to significant energy saving compared with previously presented approaches.

1. INTRODUCTION

Energy consumption is one of the critical factors in designing battery-operated systems, such as portable personal computing and communication devices. To reduce system energy consumption, supply voltage reduction is the most powerful technique since power is a quadratic function of the voltage. Recent advances in power supply circuits [2, 9] have enabled systems to operate under dynamically varying supply voltages. In such an environment, the speed of the system can be dynamically controlled. Judicious exploitation of this feature can dramatically improve the energy consumption of a real-time system.

In this paper, we are interested in studying the following type of a real-time system implemented on a variable voltage processor. The real-time system consists of jobs with predefined release times, deadlines and required number of CPU cycles. Such jobs may either be aperiodic or be instances of periodic tasks, and are scheduled by a preemptive scheduler based on some static priorities, e.g., according to the rate-monotonic policy [7]. Such a fixed-priority assignment approach is used in most real-time scheduling algorithms [8]. If the jobs are executed by a variable voltage processor, the execution time of each job varies depending on the processor speed under different voltage levels. By setting the supply voltage to different values at different times, we can essentially build a voltage schedule. The challenge is to determine the voltage (or equivalently speed) schedule which lead to the minimal energy consumption.

A number of papers have been published on similar topics. Off-line scheduling algorithms for non-preemptive hard real-time tasks are discussed in [3, 6]. In [5, 6], more general variable voltage processor models are used assuming that processor voltage cannot change instantaneously or continuously. The more practical processor models make the problem much harder to solve. A heuristic approach is described in [5], and a linear programming formulation is introduced in [6]. Yao, Demers and Shenker [14] presented an $O(N^2)$ (or $O(N \log N)$ for a more sophisticated implementation) time off-line algorithm for finding the optimal voltage schedule , where N is the number of jobs to be scheduled. They assume that jobs are scheduled according to the earliest-deadline-first (EDF) scheduling policy [7]. Hong, Potkonjak and Srivastava described an on-line scheduling algorithm for real-time tasks on variable voltage processor, where it is assumed that the release times of jobs are not known *a priori* [4]. All of the above approaches employ the dynamic EDF priority assignment scheme for scheduling the jobs. Though the EDF policy is used in some real-time systems, fixed-priority assignments are adopted in most real-time scheduling algorithms of practical interest due to its low overhead and predictability [8].

Shin and Choi [12] presented a power conscious fixed priority scheduling scheme for hard real-time systems on a variable voltage processor. The approach makes use of a simple run-time checking mechanism: the processor can either be shut down (if there is no current active job) or adopt the speed such that the current active job finishes at its deadline or the release time of the next job. The advantage of the technique is its simplicity and hence can be readily incorporated into an operating system (OS) kernel. However, it cannot exploit the fact that the release times and deadlines of most real-time jobs are known off-line. Hence, it may not be able to fully utilize the benefit provided by a variable voltage processor. In [13], Shin, Choi, and Takayasu proposed an off-line algorithm to determine the lowest maximum processor speed to execute a periodic real-time task set on a variable speed processor. It is assumed that all the tasks start at the same time, so the first job of each task would have the longest response time [7]. The algorithm finds the

*This research was supported in part by the National Science Foundation under Grant MIP-9701416.

minimum processor speed that guarantees the schedulability of the first job for each task. Note that this approach can only be applied to the periodic tasks having the same starting time. Moreover, it is not difficult to see that the minimum processor speed can be further reduced after the completion of the first job. Therefore, it still fails to maximally explore the flexibility of a variable speed processor.

In this paper, we present a technique to determine voltage schedules that result in more energy saving. Similar to that in [14], our technique is based on the assumption that the timing parameters of each job is known off-line. Two algorithms are given in the paper. The first one takes $O(N^2)$ time (N is the number of jobs) to find the minimum constant speed needed to complete each job, since constant voltage tends to result in a lower power consumption. The second algorithm, with $O(N^3)$ time complexity, builds on the first one and gives two results. First, the minimum constant voltage (or speed) needed to complete a set of jobs is obtained. This is an important parameter for systems with no sophisticated power management hardware but only simple on/off modes or where peak power consumption is a concern. Secondly, a voltage schedule is produced. We prove that this voltage schedule always results in lower energy consumption compared to using the minimum constant voltage and shutting down the system when it is idle. We show through experiments that the energy saving achieved by applying our algorithm is quite significant. Our algorithm can also be readily combined with on-line scheduling techniques such as the one in [12] to further improve energy consumption during run-time.

The rest of the paper is organized as follows. Section 2 formulates the problem and gives some motivational examples. Two novel algorithms are presented in Section 3 and 4. Experimental results are discussed in Section 5 and Section 6 concludes the paper.

2. PRELIMINARIES

In this section, we first introduce the necessary notation and formulate the problem. Then, we review some known results and provide several motivational examples.

2.1 Problem formulation

The system we are studying consists of N independent jobs, $\mathcal{J} = \{J_1, J_2, \cdots, J_N\}$, arranged in the decreasing order of their statically assigned priorities. The following timing parameters are defined for each job J_n:

- R_n: the time at which job J_n is ready to be executed, referred to as *release time*.
- D_n: the time by which J_n must be completed, referred to as *deadline*.
- C_n: the maximum number of CPU cycles needed to complete job J_n without any interruption, referred to as *workload*.

It is not difficult to see that the above system model can be readily used to model task instances in periodic real-time systems, where R_m and R_n differ by some integer multiple of the task period if J_m and J_n belong to the same task.

A single processor is used to execute the jobs in the system, and the processor can work at different voltage levels which can be continuously varied in $[0, V_{\max}]$. When the supply voltage v changes, the processor speed (s) changes proportionally, and the power consumption (P) is a convex function of the processor speed. For simplicity, we will use processor speed and supply voltage interchangeably whenever applicable.

Given a set of real-time jobs and a variable voltage processor introduced, different voltage values can be set at different times. We refer to a set of voltage values during the entire time interval when the job set $\mathcal{J}$ being executed as a *voltage schedule*. Our problem is then to determine a voltage schedule with which the lowest amount of energy is consumed and the jobs are all completed at or before their deadlines.

Several observations are helpful in formulating our problem more formally. The authors of [14] presented a theorem regarding the best speed for a given set of jobs that must be completed within an interval. We restate the theorem in the following.

THEOREM 1. *Given a set of jobs starting at t_0 and to be completed by t_1, the voltage schedule that employs a constant voltage in $[t_0, t_1]$ is necessarily an optimal schedule in the sense that no other schedule consumes less energy to complete the jobs in time.*

Based on the above theorem, we can prove (see [11]) the following lemma which describes an important feature for any optimal voltage schedule.

LEMMA 1. *An optimal voltage schedule for a job set $\mathcal{J}$ is defined on a set of time intervals each of which must start and end at either the release times or deadlines of the jobs, and the processor maintains a constant speed in each of the intervals.*

According to Lemma 1, our voltage scheduling problem can be formally defined as follows:

DEFINITION 1. *Given a job set $\mathcal{J}$, find a set of intervals, $[t_s^k, t_f^k]$, and a set of speeds, $\mathcal{S} = \{\overline{S}(t_s^k, t_f^k), k = 1, 2, \cdots, K\}$, where t_s^k and t_f^k are among the job release times and deadlines, and $\overline{S}(t_s^k, t_f^k)$ is a constant speed, such that if the processor operates accordingly, all the jobs can be completed by their deadlines and no other voltage schedules can consume less energy.*

2.2 Motivational examples

Consider a simple real-time system with 3 jobs as follows:

$$\begin{array}{lll} J_1 : C_1 = 2 & R_1 = 2 & D_1 = 6 \\ J_2 : C_2 = 6 & R_2 = 0 & D_2 = 4 \\ J_3 : C_3 = 5 & R_3 = 3 & D_3 = 8 \end{array} \quad (1)$$

In (1), the static priority assignment for the jobs are different from the EDF since J_2 has a lower priority than J_1. Under the EDF policy[14], the best voltage schedule is $\overline{S}(0, 8) = 13/8$. With the given priority assignment, it is easy to check that this voltage schedule would result J_2 missing its deadline. Apparently, the voltage schedule obtained based on the EDF policy may not be applicable any more for job sets with fixed-priority assignment.

For the same example, the off-line approach in [13] cannot be readily applied because the jobs do not start at the same time. Using the on-line voltage scheduling algorithm in [12], we can obtain the voltage schedule as shown in Figure 1(a), where the processor speed is soly determined by the release

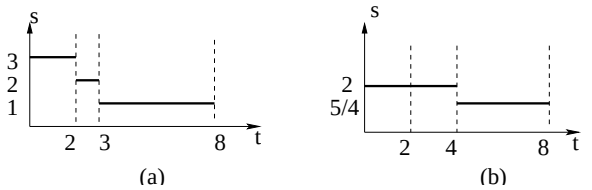

Figure 1: Two different voltage schedules for the simple real-time system in (1).

time of the next job. A better voltage schedule is given in Figure 1(b). It is not difficult to verify that the voltage schedule in Figure 1(b) consumes less energy than that in Figure 1(a). If we assume $P = s^2$, the energy in Figure 1(b) is $22.5/27 = 83\%$ of that in Figure 1(a).

From the above example, it is clear that existing approaches are not able to determine the optimal voltage schedule for a given job set when a fixed-priority assignment is used. However, we find the rationale behind the technique used in [14] is enlightening. The key to the algorithm in [14] is to find the minimum constant speed needed to finish certain subsets of all jobs. In the EDF priority assignment, this can be easily computed by

$$\overline{S}(R_m, D_n) = \frac{\sum_{J_i \in \mathcal{J}_j} C_i}{D_n - R_m} \tag{2}$$

where $D_n > R_m$, and $\mathcal{J}_j$ is the subset of jobs whose release times and deadline are *both* in $[R_m, D_n]$. When computing $\overline{S}(R_m, D_n)$, there is no need to include any jobs that are released in $[R_m, D_n]$, and have deadlines after D_n, since there *always* have lower priorities than J_n.

In the fixed-priority assignment case, (2) is no longer valid due to the fact that a job J_k released in $[R_m, D_n]$ with deadline larger than D_n may have a higher priority than J_n, and thus *may* preempt J_n in $[R_m, D_n]$ (depending on if J_n is finished before or after J_k's release time). In (1), J_2 and J_1 exhibit such a relationship. This uncertainty in the preemption relationship greatly increases the difficulty in finding the voltage schedules under the fixed-priority assignment scheme. In the following section, we present new observations and techniques to tackle such a problem.

3. DETERMINING THE MINIMUM CONSTANT SPEED FOR EACH JOB

In this section, we present our approach to find the minimum constant speed needed to complete each job by its deadline. Finding such speeds is beneficial in two aspects. First, it helps us to determine the minimum overall constant speed for the entire job set $\mathcal{J}$ such that if the supply voltage is set for this speed, it will result in the minimum energy consumption compared to any other constant speed for $\mathcal{J}$. Secondly, we can use it to derive a voltage schedule that leads to even lower energy consumption. Recall that the authors in [13] proposed a technique to compute the minimum constant speed to guarantee the schedulability for a periodic task system. Since all the tasks start at the same time, the problem reduces to finding the minimum constant speed for the first job of each task. This is a rather special case. We are dealing with a more general case where jobs can be released at any time.

Let us denote the minimum constant speed needed to complete job J_n by S_n. We first use another example as shown in (3) to illustrate several critical properties that S_n must possess.

$$\begin{array}{lllll} J_1 & : & C_1 = 1 & R_1 = 0 & D_1 = 9 \\ J_2 & : & C_2 = 4 & R_2 = 2 & D_2 = 8 \\ J_3 & : & C_3 = 5 & R_3 = 3 & D_3 = 10 \end{array} \tag{3}$$

Suppose we want to find S_3 for J_3 in (3). If we use interval $[0, 10]$ to compute S_3, then $S_3 = \overline{S}[0, 10] = 1$. Note that applying $\overline{S}[0, 10] = 1$ will cause an idle interval $[1, 2]$ and J_3 to miss its deadline. If we set $S_3 = \overline{S}[3, 10] = 5/7$, J_3 can be completed by D_3, but in order for J_3 to start at $t = 3$, the processor speed must be set to at least $\overline{S}[2, 3] = 4$. Otherwise, J_2 will prevent J_3 from finishing on time. Hence, $\overline{S}[3, 10]$ is not the valid minimum constant speed for J_3. For this example, $S_3 = \overline{S}[2, 10] = 9/8$. If we let the processor speed during $[0, 2]$ be $1/2$, and $P = s^2$, it is easy to verify that the power consumption for this schedule is $10.6/20.1 = 52.7\%$ of the one using $S_3 = \overline{S}[3, 10]$. To summarize, the minimum constant speed S_n is computed based on some intervals which must have the following properties: (*i*)there is no idle time within the interval that S_n corresponds to; (*ii*)applying S_n do not *force* other intervals to take higher speeds; and (*iii*) the interval must begin and end at the release times or deadlines of some jobs. These properties play a key role in determining the interval to compute S_n.

When deriving the minimum speed of J_n, only certain higher priority jobs whose execution may interfere with J_n's execution need to be considered. The following definitions help us to limit the number of jobs to be considered.

DEFINITION 2. *Time t is called a J_n-*scheduling point *if $t = R_i, 1 \le i \le n$ or $t = D_n$.*

For the rest of the paper, when we refer to a time t, we always mean a *scheduling point.*

DEFINITION 3. *A J_n-scheduling point t is called* the earliest scheduling point *of J_n and denoted as $T_E(n)$ if it is the largest J_n-scheduling point in $[0, R_n]$ and satisfies*

$$t \ge D_i \qquad \text{if } t > R_i, 1 \le i \le n.$$

The latest time of J_n by which J_n must be completed is called the latest scheduling point *of J_n and is denoted by $T_L(n)$.*

Based on the above definitions, $T_L(n)$ can initially be set to D_n, while $T_E(n)$ can be obtained by checking each J_n-scheduling point in the decreasing order starting from R_n. It is not difficult to see that any higher priority jobs released prior to $T_E(n)$ or after $T_L(n)$ have no impact on the speed needed to complete J_n provided that these jobs are finished by their deadlines. Thus, when computing S_n, we only need to focus on the jobs released within $[T_E(n), T_L(n)]$.

Since speed is closely related with average workload, we introduce the definition of J_n-intensity to capture the concept of average workload for job J_n.

DEFINITION 4. *Let t_a, t_b be two J_n-scheduling points, J_n-*intensity *in the interval $[t_a, t_b]$, denoted by $I_n(t_a, t_b)$, is defined to be*

$$I_n(t_a, t_b) = \frac{\sum_{i=1}^{n} \delta(J_i) * C_i}{t_b - t_a}, \tag{4}$$

where

$$\delta(J_i) = \begin{cases} 1 & t_a \le R_i < t_b \\ 0 & otherwise \end{cases} \quad (5)$$

Having no idle time is another key properties required for S_n, we give the following definition to precisely capture the idle time related concepts.

DEFINITION 5. *Interval* $[t_a, t_b]$ *is a* J_n-busy interval, *if*

- t_a, t_b *are* J_n *scheduling points, and* $T_E(n) \le t_a \le R_n < t_b \le T_L(n)$.
- *There is no idle time before the release time of any job within* $[t_a, t_b]$ *if the constant speed* $I_n(t_a, t_b)$ *is applied within the interval.*

For job J_n, it is not difficult to see that there may exist a number of J_n-busy intervals. The largest one among them is particularly interesting and we give a definition for it below.

DEFINITION 6. *A* J_n*-busy interval* $[t_s, t_f]$ *is called the* J_n-essential interval *if for any* J_n*-busy interval* $[t_a, t_b]$, *we have*

$$t_s \le t_a \quad and \quad t_b \le t_f. \quad (6)$$

The J_n-intensity corresponding to the J_n-essential interval possesses the properties of S_n stated earlier, which is summarized in the following important lemma (see [11]).

LEMMA 2. *The* J_n*-essential interval,* $[t_s, t_f]$, *and the corresponding* J_n*-intensity,* $I_n(t_s, t_f)$, *satisfy*

$$I_n(t, t_s) < I_n(t_s, t_f), \quad T_E(n) \le t < t_s, \quad (7)$$

$$I_n(t_s, t_f) < I_n(t_f, t), \quad t_f < t \le T_L(n), \quad (8)$$

and

$$I_n(t_s, t_f) = \min_{[t_a, t_b]} I_n(t_a, t_b), \quad (9)$$

where $[t_a, t_b]$ *is any* J_n*-busy interval. Furthermore, if* $I_n(t_s, t_f)$ *is adopted as the processor speed during* $[t_s, t_f]$, J_n *is completed by its deadline.*

According to Lemma 2, $I_n(t_s, t_f)$ is the valid minimum constant speed S_n. Thus, determining the minimum constant speed for each job now becomes determining the essential interval and corresponding intensity associated with each job. We present our algorithm, Algorithm 1, to search for J_n-essential interval and compute S_n.

Algorithm 1 follows the basic principle laid down in Lemma 2 but employs a little different search mechanism. It only searches a subset of busy intervals whose starting points are the scheduling points in $[T_E(n), R_n]$, and thus takes less time than a straightforward implementation of Lemma 2. The effectiveness of Algorithm 1 is guaranteed by the following theorem (see [11] for the proof).

THEOREM 2. *Algorithm 1 produces, in* $O(N^2)$ *time, the* J_n*-essential interval and the minimum constant speed to complete* J_n.

4. DETERMINING THE GLOBAL VOLTAGE SCHEDULE

Based on the algorithm searching the minimum constant speed for each job, we can find both the minimum constant speed needed to satisfy all job deadlines and a better voltage schedule to further improve the system energy consumption. In the following, we present the algorithm and several theorems, which tackle the two problems simultaneously. We first introduce the concept of *critical interval*.

Algorithm 1 Construct the essential interval for a job

```
Input: Job set J = {J_1, ..., J_N}, and T_E(n) and T_L(n) for job J_n
Output: J_n-essential interval [t_s, t_f] and S_n
t'_a = t_b = R_n;
t_a = T_E(n);
t'_b = T_L(n);
while t_a ≠ t'_a or t'_b ≠ t_b do
    t_a = t'_a;
    t'_b = t_b;
    for every J_n-scheduling point in [t_a, T_L(n)] do
        Find t_b such that I(t_a, t_b) is the minimum;
    end for
    for every J_n-scheduling point in [T_E(n), t_a] do
        Find t'_a such that I(t'_a, t_b) is the maximum;
    end for
end while
t_s = t_a, t_f = t_b, S_n = I(t_a, t_b);
```

DEFINITION 7. *The essential interval* $[t_s, t_f]$ *with the largest* J_n*-intensity is called the* critical interval *of job set* J.

It is easy to see that the minimum constant speed needed for J to be feasible must at least equal the J_n-intensity corresponding to the critical interval of $\mathcal{J}$. Applying $S_n = I_n(t_s, t_f)$ in $[t_s, t_f]$ can guarantee finishing J_n by its deadline, what happens to other jobs in this interval and the jobs elsewhere?

Suppose that the critical interval of $\mathcal{J}$ corresponds to J_n-essential interval $[t_s, t_f]$. We would like to investigate the impact of removing $[t_s, t_f]$ from the overall execution time interval of $\mathcal{J}$. By removing the critical interval of $\mathcal{J}$, we mean the following:

1. Remove from $\mathcal{J}$ the jobs *associated with* $[t_s, t_f]$, that is, job J_n and all other jobs having higher priorities than J_n and released within $[t_s, t_f)$.
2. "Shrink" the interval $[t_s, t_f]$ into a single time point, i.e., reduce every time instant greater than t_f by the amount of $(t_f - t_s)$. If R_i, $T_E(i)$ or $T_L(i)$ for any job J_i is inside $[t_s, t_f]$ before the reduction, it will be changed to the value of t_s.

For the remaining jobs, we can again find the critical interval. Repeatedly performing the above steps, we obtain a set of critical intervals and the corresponding speeds. We will show that these critical intervals form a valid, low-energy voltage schedule. We first summarize the above procedure in Algorithm 2 on the next page.

By applying Algorithm 2, we obtain a set of intervals $\mathcal{T}$ and their corresponding constant speeds $\mathcal{S}$. The following two theorems describe the important characteristics of $\mathcal{T}$ and $\mathcal{S}$(see [11] for the proofs).

THEOREM 3. *Given a job set* $\mathcal{J}$, *let* $[t_s^k, t_f^k]$ *and* $\overline{S}(t_s^k, t_f^k)$ *for* $1 \le k \le K$ *be the critical intervals and corresponding*

speeds output from Algorithm 2. Every job in $\mathcal{J}$ is guaranteed to be completed by its deadline if $\overline{S}(t_s^k, t_f^k)$ is used in the corresponding interval $[t_s^k, t_f^k]$.

THEOREM 4. *Given a job set $\mathcal{J}$, speeds $\overline{S}(t_s^k, t_f^k) \in \mathcal{S}$ obtained by Algorithm 2 satisfy the following: $\overline{S}(t_s^1, t_f^1) \geq \overline{S}(t_s^2, t_f^2) \geq \cdots \geq \overline{S}(t_s^K, t_f^K)$.*

Algorithm 2 Construct the set of critical intervals for $\mathcal{J}$.

1: **Input:** Job set $\mathcal{J} = \{J_1, ..., J_N\}$
2: **Output:** A set of critical intervals $\mathcal{T} = \{[t_s^1, t_f^1], \cdots, [t_s^K, t_f^K]\ \}$, and a set of speeds $\mathcal{S} = \{\overline{S}(t_s^k, t_f^k)\}$ for $1 \leq k \leq K$
3: **for** every job $J_i \in \mathcal{J}$ **do**
4: $\mathcal{A}, \mathcal{S}' = \emptyset$;
5: Find $T_E(i)$ and $T_L(i)$ according to Definition 3;
6: Determine J_i-essential interval $[t_s, t_f]$ and minimum speed S_i by Algorithm 1;
7: Add $[t_s, t_f]$ to $\mathcal{A}$ and S_i to S';
8: **end for**
9: k =1;
10: **while** $\mathcal{A}$ is not empty **do**
11: Select $S_j = S_i, i = 1, \cdots, n$ from $\mathcal{S}'$ and corresponding $[t_s, t_f]$ from $\mathcal{A}$;
12: Add $[t_s, t_f]$ as $[t_s^k, t_f^k]$ to $\mathcal{T}$ and S_j as $\overline{S}(t_s^k, t_f^k)$ to $\mathcal{S}$;
13: Remove $[t_s, t_f]$ from $\mathcal{A}$ and S_j from $\mathcal{S}'$;
14: **for** $J_m \in \mathcal{J}$ **do**
15: **if** $m \leq j$ and $t_s^k \leq R_m < t_f^k$ **then**
16: Remove J_m from $\mathcal{J}$;
17: Remove S_m from $\mathcal{S}$ and corresponding $[t_s, t_f]$ from $\mathcal{A}$;
18: **else**
19: **if** $t_s^k < R_m \leq t_f^k$; **then**
20: $R_m = t_s^k$;
21: **end if**
22: **if** $t_s^k < T_E(m) \leq t_f^k$; **then**
23: $T_E(m) = t_s^k$;
24: **end if**
25: **if** $t_s^k < T_L(m) \leq t_f^k$; **then**
26: $T_L(m) = t_s^k$;
27: **end if**
28: **end if**
29: **end for**
30: **for** every scheduling point $t > t_f^k$ **do**
31: $t = t - (t_f^k - t_s^k)$;
32: **end for**
33: **for** every $J_m \in \mathcal{J}$ whose $R_m, T_E(m)$ or $T_L(m)$ has changed **do**
34: Re-construct and modify J_m-essential interval $[t_s, t_f]$ in $\mathcal{A}$;
35: Re-compute and modify S_m in $\mathcal{S}'$;
36: **end for**
37: k++;
38: **end while**
39: Output $\mathcal{T}$ and $\mathcal{S}$;

From Theorem 3, we conclude that the set of critical intervals and their associated speeds obtained by Algorithm 2 form a valid voltage schedule. Also, based on Theorem 3 and 4, we have the the following corollaries(see [11]).

COROLLARY 1. *The first speed in the speed set produced by Algorithm 2 is the minimum constant speed that can be applied throughout the execution of all jobs such that no jobs violate their deadlines.*

COROLLARY 2. *The voltage schedule obtained by Algorithm 2 always saves more energy than the one that applies the minimum constant speed when the processor is busy while shuts down the processor when it is idle.*

Our approach to constructing a low-energy voltage schedule is a greedy approach since we strive to find the minimum constant speed during any critical interval. It guarantees to result the minimum peak power consumption. However, our algorithm may not always produce the minimum-energy voltage schedule. In the following experimental section, we will show that the energy saving achieved by applying our algorithm is quite significant.

5. EXPERIMENTAL RESULTS

In this section, we compare the performance of our research with the work proposed in [12] and [13]. For brevity, we use **VSLP**, **LPFS**, and **LPPS** to represent Algorithm 2, the algorithm in [12], and the one in [13], respectively. We assume that $P = S^2$. The approaches in [12] and [13] are intended to apply to periodic tasks, therefore, we also construct our job sets from periodic tasks to ensure a fair comparison.

Since different voltage scheduling approaches may benefit from the task timing parameters differently, a fair comparison needs to study a large spectrum of utilization factor values. In our experiments, we first randomly generate a set of periodic task sets, each of which contains five tasks. The period of each task is randomly selected from a uniform distribution between 10 to 50, the deadline of each task is assumed to equal its period, and the worst case execution time (WCET) is less than its period and is also randomly generated. Since the utilization bound for 5 tasks is approximately 0.74(see [7]), we partition the utilization factor values from 0.1 to 0.7 into intervals of length 0.1. To reduce statistical errors, the number of task sets with utilization values within each interval is no less than 20, and the average results are collected in Table 1. Furthermore, we apply our approach to two real-world applications: CNC (Computerized Numerical Control) machine controller [10] and INS (Inertial Navigation System) [1], and the results are shown in Table 2. We also incorporate in our approach the on-line voltage scheduling algorithm in [12], i.e., extending the execution of the current job till its deadline or the arrival of the next job when there is no job in the ready queue. Similar to that in [12], the execution time for a task is assumed to be normally distributed within its best and worst case execution time, and we assume that best case execution time (BCET) for each task is half of its WCET.

In Table 1 and Table 2, columns E_{VSLP}, E_{LPFS}, and E_{LPPS} represent the power consumption by algorithms **VSLP**, **LPFS**, and **LPPS**, respectively. To better present our results, in Table 1, we let E_{LPFS} be 100 and normalize the other two correspondingly. Column **Worst. Exec** represents the cases when task execution times equal their WCETs, and column **Norm. Exec** represents the cases when the task execution times are normally distributed.

From the statistical data shown in Table 1, one can readily conclude that our voltage scheduling strategy is more

Table 1: Experimental results comparing the three voltage scaling approaches for tasks with fixed priorities.

	Worst. Exec			Norm. Exec		
Utilization	E_{LPFS}	E_{LPPS}	E_{VSLP}	E_{LPFS}	E_{LPPS}	E_{VSLP}
0.1-0.2	100	45.02	41.46	100	44.31	41.12
0.2-0.3	100	50.44	47.63	100	48.07	46.35
0.3-0.4	100	68.30	66.31	100	52.46	50.83
0.4-0.5	100	59.89	55.86	100	59.41	54.90
0.5-0.6	100	73.90	69.77	100	78.55	69.62
0.6-0.7	100	89.43	84.96	100	88.97	83.36

Table 2: Experimental results for two real-world real-time systems

		Worst. Exec			Norm. Exec		
Systems	**Utilization**	E_{LPFS}	E_{LPPS}	E_{VSLP}	E_{LPFS}	E_{LPPS}	E_{VSLP}
CNC	0.4887	48173.9	31164.7	29867.2	32036.5	21380.1	19515.6
INS	0.7353	307670	272132.0	270890	193684	168993	168153

efficient in energy saving comparing with the other two approaches, with the job execution times equal to or less than their WCETs. The reason for this is that: when the ready queue is not empty, **LPFS** always uses the full speed to execute the jobs; **LPPS** is more efficient and uses the lowest maximum constant speed; in our approach, even lower speed is possible according to the voltage schedule obtained by Algorithm 2. Moreover, note that in Table 1, our algorithm can save more energy when the utilization of a task set is lower. The reason for this is that when the task utilization is low, our algorithm tends to find a constant speed which can be applied to relatively long intervals while still meet the deadline requirements for the jobs. When the task utilization is higher, the interval with constant speed becomes relatively shorter and saving becomes somewhat less. The results applying our approach to two real world systems, as shown in Table 2, also agree with our analysis above.

6. SUMMARY

In this paper, we study the problem of determining the optimal voltage schedule for a real-time system with fixed-priority jobs implemented on a variable voltage processor. Two algorithms are presented in the paper. The first one takes $O(N^2)$ time, where N is the number of jobs to be scheduled, and finds the minimum constant speed needed to complete each job. The second algorithm, with $O(N^3)$ time complexity, builds on the first one and produces the following: (i) the minimum constant voltage (or speed) needed to complete a set of jobs, and (ii) a voltage schedule which always results in lower energy consumption compared to using the minimum constant voltage and shutting down the system when it is idle. The experimental results obtained from both a number of randomly generated and real-world real-time systems have shown that our voltage schedule algorithm consistently leads to more energy saving than existing approaches. Our future work is to improve the algorithms presented and strive to find an optimal voltage schedule.

7. REFERENCES

[1] A.Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, 21:920–934, May 1995.

[2] V. Gutnik and A. Chandrakasan. An efficient controller for variable supply-voltage low power processing. *Symposium on VLSI Circuits*, pages 158–159, 1996.

[3] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power optimization of variable voltage core-based systems. *Proceedings of Design Automation Conference*, pages 176–181, 1998.

[4] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. *International Conference on Computer-Aided Design*, pages 653–656, 1998.

[5] 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 Real-Time Systems Symposium*, pages 178–187, 1998.

[6] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. *International Symposium on Low Power Electronics and Design*, pages 197–202, August 1998.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 17(2):46–61, 1973.

[8] J. Liu. *Real-Time Systems*. Prentice Hall, NJ, 2000.

[9] W. Namgoong, M. Yu, and T. Meng. A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator. *IEEE Internation Solid-State Circuits Conference*, pages 380–381, 1997.

[10] N.Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual assessment of a real-time system design: a case study on a cnc controller. *IEEE Real-Time Systems Sumposium*, Dec 1996.

[11] G. Quan and X. Hu. *Energy Efficient Scheduling for Real-Time Systems on Variable Voltage Processors.* Technical Report TR01-3, Dept. of Compu. Sci. & Eng., Univ. of Notre Dame, 2001.

[12] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. *Design Automation Conference*, pages 134–139, 1999.

[13] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. *International Conference on Computer-Aided Design*, pages 365–368, 2000.

[14] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.