Energy Efficient DVS Schedule For Fixed-Priority Real-Time Systems

GANG QUAN
University of South Carolina

and
XIAOBO SHARON HU
University of Notre Dame

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. By exploiting more efficiently the processor slack time, our approach can be more effective in reducing the execution speed for real-time tasks when necessary. Our approach also produces the minimum constant voltage needed to feasibly schedule the entire job set. With both randomly generated and practical examples, our heuristic approach can achieve the dynamic energy reduction very close to the theoretically optimal one (within 2%) with much less computation cost.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose And Application-Based Systems—*Real-Time and Embedded Systems*; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Dynamic voltage scaling, Fixed-priority scheduling, Low power, Real-time

1. INTRODUCTION

Energy consumption is becoming one of the critical factors in designing microprocessor based systems, because of not only the prevalence of battery-operated systems, such as portable personal computing and communication devices, but also the considerations of packaging cost and reliability issues. In a CMOS-based processor, power consumption consists of two components: dynamic and static power [Rabaey and Pedram 1996]. The dynamic component comes from charging and discharging capacitance and the short circuit current due to the non-zero rising and falling

Author's address: Gang Quan, Department of Computer Science & Engineering, University of South Carolina, Columbia, SC 29208, gquan@cse.sc.edu.
Xhaobo Sharon Hu, Department of Computer Science & Engineering, University of Notre Dame, Notre Dame, IN 46556, shu@cse.nd.edu.

times of signals. The static component mainly attributes to the leakage current sources, including both the subthreshold current and the junction reverse-bias current, flowing through the transistors. In many embedded systems today, dynamic energy consumption tends to be the dominate part in the overall energy consumption. With the scaling of the IC technology, however, static power consumption is increasing rapidly [ITRS snet].

Dynamic Voltage Scaling (DVS), which enables systems to operate under dynamically varied supply voltages, forms the basis for the total power consumption reduction. Since dynamic power is a quadratic function of the voltage, reducing the supply voltage and therefore the processor speed can effectively minimize the dynamic power consumption, which is the dominating part in the power consumption for most of today's processors and will still be a major component in overall power consumption in the future [Duarte et al. 2002]. In terms of reducing the overall energy consumption, many newly developed scheduling techniques, e.g. [Irani et al. 2003; Jejurikar et al. 2004; Niu and Quan 2004; Yan et al. 2003], are constructed based on the DVS schedule. For example, Yan *et. al.* [Yan et al. 2003] proposed to first reduce the processor speed such that no real-time task misses its deadline, and then adjust the voltage supply and body biasing voltage based on the processor speed to reduce the overall power consumption. For processor with no adaptive body biasing control, Iran *et. al.* [Irani et al. 2003] showed that the overall optimal voltage schedule can be constructed from the traditional DVS voltage schedule that optimizes the dynamic energy consumption. Therefore, judiciously exploiting the DVS feature of the variable voltage processor plays an important role in providing the just-in-time computation and achieving high *overall* energy efficiency for real-time embedded systems.

In this paper, we are interested in studying the DVS scheduling techniques for a fixed-priority (FP) real-time system. Specifically, the 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. These jobs are scheduled by a preemptive scheduler based on fixed priorities, e.g., according to the Rate-Monotonic Scheme (RMS) [Liu and Layland 1973]. FP scheme is adopted in most real-time systems of practical interest for its low overhead, ease of implementation, high predictability [Liu 2000]. While finding the optimal voltage schedule that consumes the least dynamic energy for a job set scheduled with Earliest Deadline First (EDF) scheme [Liu and Layland 1973] can be solved in polynomial time [Yao et al. 1995], the problem becomes NP-hard when the jobs are scheduled according to the FP scheme [Yun and Kim 2003].

In this paper, we present an efficient heuristic technique to find the voltage schedule of an FP real-time system, *i.e.,* different voltage (processor speed) settings at different time. Two algorithms are given 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 a job. 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 when designing systems with no sophisticated power management hardware but only simple on/off modes or where large voltage transition

overhead is a concern [Mochocki et al. 2002]. Secondly, a voltage schedule is produced. We prove that this voltage schedule always results in lower dynamic energy consumption compared to using the minimum constant voltage and shutting down the system when it is idle. The iterative technique introduced in this paper can further be exploited to handle discrete voltage levels and transition overhead.

We have tested our algorithm with both randomly generated job sets and practical applications, and compared with the results in previous research, e.g. [Shin and Choi 1999; Shin et al. 2000; Quan and Hu 2003]. The experimental results show that our approach can achieve dynamic energy savings very close to that by the optimal ones (with 2%) with much less computation complexity.

The rest of the paper is organized as follows. Section 2 introduces the related work and then formally formulates the problem. Two novel algorithms are presented in Section 3 and 4. Section 5 discusses further the characteristic of our approach and its extensions. Experimental results are discussed in Section 6 and Section 7 concludes the paper. This paper is the extended version of [Quan and Hu 2001].

2. PRELIMINARIES

In this section, we first introduce our system models and review some known results. Then we formally formulate the problem to be solved.

2.1 System model

The real-time 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*.

The real-time jobs are executed by a single processor based on the FP preemptive scheduling scheme. Without loss of generality, we assume no special relationships among release times and deadlines of different jobs. 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.

The system architecture under consideration includes a variable voltage processor, other necessary circuits for altering voltage and clock frequency, an operating system (OS), etc. Interesting readers can refer to [Burd and Brodersen 2000; Gutnik and Chandrakasan 1996; Namgoong et al. 1997; Nielsen et al. 1994; T. Pering 2000] for more implementation details of such an architecture. For ease of our discussions, we assume that supply voltage can be changed continuously. This constraint is removed in Section 4.2.

2.2 Related work

There has been substantial research conducted that exploits the DVS capability of modern processors in real-time scheduling for saving energy. To determine at what

voltage (speed) a system should operate at each time instant can be done either on-line or off-line. Weiser *et.al.* [Weiser et al. 1994] proposed an on-line approach to reduce the processor speed based on the assumption that the workload in the following time frame is similar to that in the previous one. Later on, Govil *et. al.* [Govil et al. 1995], Huang *et. al.* [Hwang and Wu 1997], and Jacob *et. al.* [Lorch and Smith 2001] improved this approach by proposing other better prediction policies for the future workload. However, as shown in an empirical work [Pering et al. 1998], the energy saving achieved by these approaches are likely to be severely limited due to the lack of knowledge for the system timing characteristics. Moreover, these approaches cannot be applied to many real-time systems with hard deadlines.

To meet the hard real-time requirements, many novel on-line techniques are also proposed, e.g. [Pillai and Shin ; Kim et al. 2002; Shin and Choi 1999; Aydin et al. 2001b; Kim et al. 2003]. While an on-line approach can efficiently exploit run-time variations, we believe that an energy efficient scheduling approach must incorporate more sophisticated off-line analysis results since many real-time embedded systems have highly deterministic timing specifications, and the energy consumption for such systems can be greatly reduced by aggressively taking advantage of these information when applying the DVS techniques. Furthermore, off-line approaches can afford to employ more advanced optimization algorithms without being constrained much by the time and energy consumption due to carrying out the approaches themselves. Some recent on-line techniques [Mochocki et al. 2005] also show that off-line results may be used effectively to guide on-line decisions.

There have been many off-line approach results reported in the literature, e.g. [Ishihara and Yasuura 1998; Hong et al. 1998; Yao et al. 1995; Aydin et al. 2001a; Jejurikar and Gupta 2002]. Yao, Demers and Shenker [Yao et al. 1995] presented an off-line algorithm to find the optimal voltage schedule for a real-time job set with arbitrary arrival times and hard deadlines. Aydin *et. al.* proposed an optimal voltage schedule for periodic task sets with different power consumption characteristics in [Aydin et al. 2001b]. In [Ishihara and Yasuura 1998], the lower power scheduling problem is formulated as an integer linear programming problem, and the system consists of a set of tasks with same arrive times and deadlines but different context switching activities. An off-line scheduling heuristic for non-preemptive hard real-time tasks is discussed in [Hong et al. 1998]. Most of the above work [Yao et al. 1995; Aydin et al. 2001b; Hong et al. 1998] assume that jobs are scheduled according to the EDF priority assignment [Liu and Layland 1973].

A number of papers have been published on FP DVS scheduling techniques. Shin and Choi [Shin and Choi 1999] presented an on-line FP scheduling scheme for hard real-time systems on a variable voltage processor. The advantage of the technique is its simplicity and hence can be readily incorporated into an operating system (OS) kernel. However, it determines the processor speed based solely on the release time of the closest "ready" job. It cannot exploit the fact that the release times and deadlines of most real-time jobs, particularly the jobs corresponding to the instances of periodic tasks, are known off-line. Hence, it often fails to fully utilize the benefit provided by a variable voltage processor. In [Makzak and Chakrabarti 2003], Manzak and Chakrabarti proposed to use the Lagrange multiplier method to determine the processor speed for a periodic task set scheduled with RMS. The

energy consumption is minimized under the constraint that the total utilization is a constant no bigger than the well-known utilization bound [Liu and Layland 1973]. Unfortunately, the energy saving with this approach are severely limited since using the utilization bound to predict the schedulability of the real time systems scheduled with RMS can lead to very pessimistic results [Liu and Layland 1973]. For example, according to this approach, a real-time system with 5 periodic tasks will have to limit the utilization of the processor to be within 0.74 which is rather pessimistic. In regard to this, Shin, Choi, and Takayasu [Shin et al. 2000] proposed to determine the lowest maximum processor speed based on the worst case response time analysis [Lehoczky et al. 1989]. Note that this approach is suitable for the periodic tasks having the same starting time or they can still be very pessimistic otherwise. Moreover, it is not difficult to see that the processor speeds can be further reduced for the jobs not having the worst case response time.

In [Yun and Kim 2003], Yun and Kim formally proved that the problem of finding the optimal voltage schedule for an FP real-time job set is NP-hard. Then they provided an approximation algorithm to find the near optimal solution with the complexity of $O(N^4/\epsilon^2)$ (where N is the number of jobs to be scheduled and ϵ $(0 < \epsilon \ll 1)$ measures the *closeness* of a desired solution to the optimal solution). Even though it is reported [Yun and Kim 2003] that this approach takes polynomial time in theory, the computation cost to get a good quality solution can be extremely high which makes it impractical to large real-time applications. In [Quan and Hu 2003], Quan and Hu proposed an approach to find the optimal voltage schedule for this problem. Unfortunately, this approach suffers prohibitively high computation cost, i.e., the computation complexity may increase exponentially with the number of jobs in the worst case, and therefore can only be applied to real-time systems with a small number of real-time jobs.

Given the NP nature of this problem, a computation efficient heuristic that can effectively save the energy is more desirable than the exhaustive search strategy. In what follows, we first define the problem formally and then present our heuristic techniques to solve the problem.

2.3 Problem formulation

Given a set of real-time jobs and a variable voltage processor as introduced, different voltage values (or processor speeds) can be set at different times. We refer to a set of voltage values or speeds 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 dynamic 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 [Yao et al. 1995] 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 2.1. *Assume that the dynamic power consumption of a variable voltage processor is a convex function of the processor speed. 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 dynamic energy to complete the jobs in time.

Based on the above theorem, we can easily prove the following lemma which describes an important feature for any optimal voltage schedule.

LEMMA 2.2. *An optimal voltage schedule for a job set $\mathcal{J}$ is defined on a set of time intervals in which the processor maintains a constant speed, and each of these intervals must start and end at either the release times or deadlines of the jobs.*

Proof: Suppose the curve $S(t)$ in Figure 1 is the optimal voltage schedule for a given job set, and $t_0, t_1, \cdots, t_m$ are the release times or deadlines of the jobs. Let $s_1, s_2, \cdots, s_m$ be the constant speeds such that

$$s_i \times (t_i - t_{i-1}) = \int_{t_{i-1}}^{t_i} S(t)dt,$$

i.e., the speeds that result in the same work load as $S(t)$ within each interval. It follows that no job will violate its deadline while the processor running at the speeds $s_1, s_2, \cdots s_m$. According to Theorem 2.1, the processor will consume less dynamic energy running at the speeds designated by $s_1, s_2, \cdots, s_m$ than that running at the speed defined by $S(t)$. Thus an optimal schedule can only change its speed either at the release times or deadlines of some jobs. □

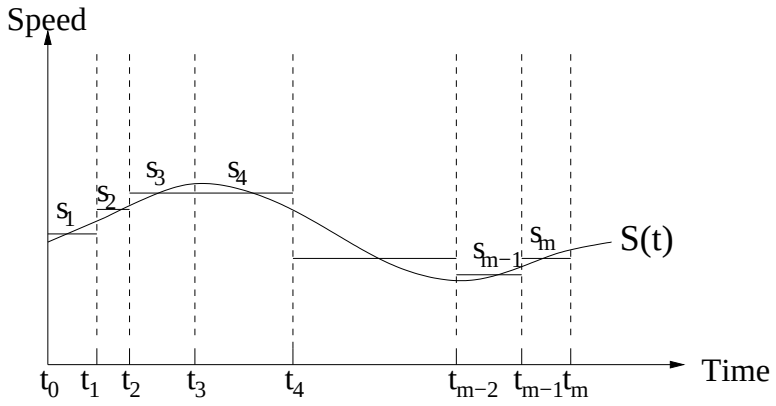

Fig. 1. Optimal dynamic energy-saving schedule for a FP task set.

According to Lemma 2.2, the processor needs only update its speed at a *scheduling point*, i.e., a release time or deadline of certain real-time job. With this lemma, our voltage scheduling problem can be formally defined as follows:

Problem 2.3. 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 jobs' scheduling points, 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.

While the only difference between Problem 2.3 and that studied in [Yao et al. 1995] is the way in which the priorities are assigned, it has been shown in [Quan and Hu 2001] that the approach in [Yao et al. 1995] is not able to determine the optimal voltage schedule for a given job set when the FP scheme is used. However, we find the rational behind the technique used in [Yao et al. 1995] is enlightening. The key to the algorithm in [Yao et al. 1995] 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

$$\overline{S}(R_m, D_n) = \frac{\sum_{J_i \in \mathcal{J}_j} C_i}{D_n - R_m} \tag{1}$$

where $D_n > R_m$, and $\mathcal{J}_j$ is the subset of jobs whose release times and deadline are *both* in $[R_m, D_n]$. In 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 they *always* have lower priorities than J_n. In the FP assignment case, (1) 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. If J_k does have a higher priority, it *may* preempt J_n in $[R_m, D_n]$ (depending on if J_n is finished before or after J_k's release time). This uncertainty in the preemption relationship greatly increases the difficulty in finding the voltage schedules under the FP assignment scheme. In the following section, we present new observations and techniques to tackle such a problem.

3. DETERMINING THE CONSTANT MINIMUM SPEED TO COMPLETE A JOB

In this section, we present our approach to find the minimum constant speed needed to complete a job by its deadline. Finding such speeds is beneficial in two aspects. First, it can help us 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 dynamic 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 dynamic energy consumption. In what follows, we first use an example to illustrate several critical observations to compute the constant minimum speed. Then we derive several key properties for the constant minimum speed and its associated interval, and propose an algorithm to find the constant minimum speed.

3.1 Properties related to the constant minimum speed

We use the example shown in Figure 2 to reveal several critical properties that the constant minimum speed must possess. Let us denote by S_n the minimum constant speed needed to complete a subset of jobs if J_n is the lowest priority job in this job set. Suppose we want to find S_3 for J_3 in Figure 2. If we use interval $[0, 14]$ to compute S_3, then, by (1), $S_3 = \overline{S}[0, 14] = 1/2$. However, applying $\overline{S}[0, 14] = 1/2$ will cause an idle interval $[2, 5]$ and J_3 to miss its deadline. On the other hand, if we set $S_3 = \overline{S}[6, 14] = 1/4$, J_3 can be completed by D_3. But in order for J_3 to start at $t = 6$, the processor speed must be set to at least $\overline{S}[5, 6] = 4$ which would not be possible if we assume the maximal speed is 1. With a lower speed in [5,6], J_1 will prevent J_3 from finishing on time. Hence, $\overline{S}[6, 14]$ is not the valid minimum constant speed for J_3. For this example, a valid minimum constant speed for J_3 is $S_3 = \overline{S}[5, 14] = 2/3$. Even if we allow $\overline{S}[5, 6] = 4$ for the case in Figure 2(b),

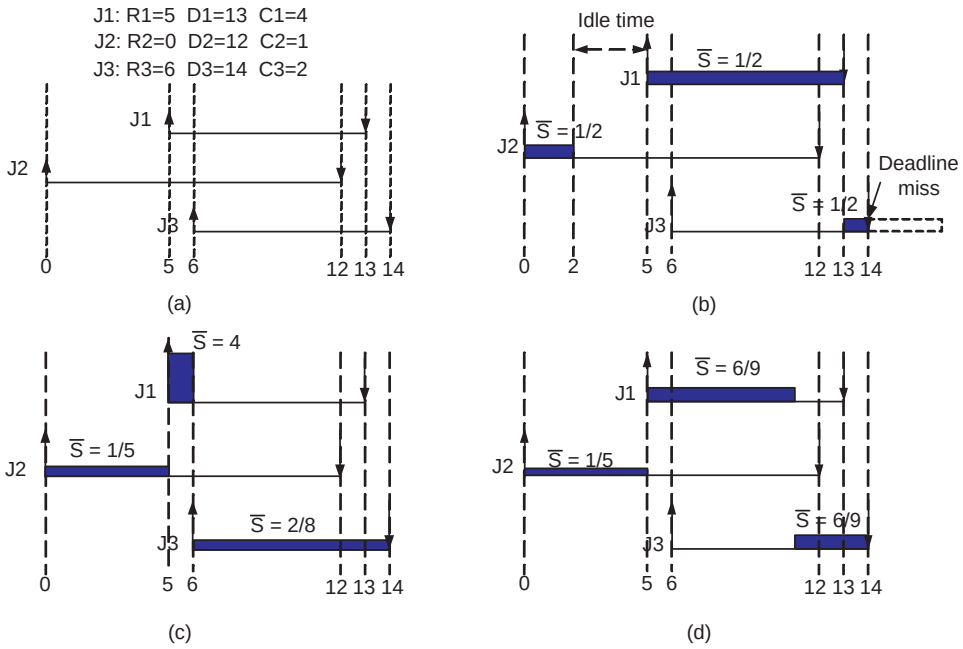

Fig. 2. Computing the minimum constant speed for J_3 and its associated intervals. The up and down arrows represent the jobs' release times and deadlines, respectively. (a) A real-time system with three jobs. (b) Using $S_3 = 1/2$ for interval $[0, 14]$ will cause an idle interval $[2, 5]$ and J_3 to miss its deadline. (c) Using $S_3 = 2/8$ for interval $[6, 14]$ will allow J_3 to complete by its deadline on the condition that $\overline{S}[5, 6] = 4$. (d) Using $S_3 = 6/9$ for interval $[5, 14]$ and every job can meet their deadlines.

note that we can let the processor speed during $[0, 5]$ be $1/5$, it is easy to verify that the dynamic power consumption ($P = s^3$) for Figure 2(d) is 4.68% of that for Figure 2(c).

To summarize, the minimum constant speed S_n is computed based on some intervals which must have the following properties:

—There is no idle time within the interval that S_n corresponds to.

—Applying S_n do not *force* other intervals to take higher speeds.

—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.

3.2 Determining the constant minimum speed for a single job

In what follows, we introduce how to compute the minimum constant speed needed to guarantee the deadline of one particular job. To make our explanation more concrete, we use the example as shown in Figure 3.

To compute the minimum speed of J_n, we need to consider the jobs in J_n that have *higher* priority than that of J_n. However, not all the higher priority jobs in $\mathcal{J}$ need to be considered to determine the minimum speed for J_n since some of them

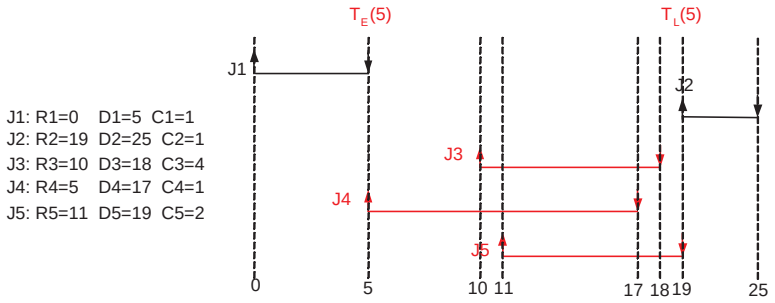

Fig. 3. An illustrative example with five jobs. $T_E(5)$ and $T_L(5)$ are the earliest and latest scheduling point of J_5, respectively.

have no possibility to interfere with the execution of J_n. The following definitions help us to limit the number of jobs to be considered.

Definition 3.1. **(Scheduling Point)** Time t is called a J_n*-scheduling point* if $t = R_i, 1 \leq i \leq n$ or $t = D_n$.

The J_n-scheduling points are simply the release time and deadline of J_n and the release times of the jobs with priorities higher than that of J_n. According to Lemma 2.2, if the processor speed varies between two consecutive scheduling points, we can always find a constant speed in the corresponding interval which is more energy efficient. **For the rest of the paper, when we refer to a time t, we always mean a *scheduling point*.**

Definition 3.2. **(Earliest and Latest Scheduling Points)** The largest J_n-scheduling point t that satisfies $0 \leq t \leq R_n$ and for all $i < n$,

$$t \leq R_i \qquad \text{if } \ t \leq D_i,$$

and

$$t \geq D_i \qquad \text{if } \ t \geq R_i,$$

is called *the earliest scheduling point* of J_n and is denoted by $T_E(n)$. 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 be simply 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. For the job set in Figure 3, one can readily verify that $T_E(5) = 5$ and $T_L(5) = 19$. It is not difficult to see that the higher priority jobs released prior to $T_E(n)$ (e.g. J_1 in Figure 3) or after $T_L(n)$ (e.g. J_2 in Figure 3) do not have any impact on the speed needed to complete J_n (e.g. J_5 in Figure 3) in a feasible schedule. 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 3.3. **(Job Intensity)** 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

$$I_n(t_a, t_b) = \frac{\sum_{i=1}^{n} \delta(J_i) * C_i}{t_b - t_a}, \tag{2}$$

where

$$\delta(J_i) = \begin{cases} 1 & t_a \le R_i < t_b \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In Figure 3, for instance, we have $J_5(5, 19) = \frac{4+1+2}{14} = 1/2$. J_n-*intensity* has a very useful property which is summarized below.

LEMMA 3.4. *Let t_a, t_b, and t_c be three J_n-scheduling points, and $t_a < t_b < t_c$. Then $I_n(t_a, t_b)$, $I_n(t_b, t_c)$, and $I_n(t_a, t_c)$ always satisfy one of the following two inequalities:*

$$I_n(t_a, t_b) \ge I_n(t_a, t_c) \ge I_n(t_b, t_c), \tag{4}$$

or

$$I_n(t_a, t_b) \le I_n(t_a, t_c) \le I_n(t_b, t_c). \tag{5}$$

Proof: Let the workload in $[t_a, t_c]$, $[t_a, t_b]$, and $[t_b, t_c]$ be W, W_1, and W_2, respectively. Then

$$W = W_1 + W_2$$

$$I_n(t_a, t_c) = \frac{W_1 + W_2}{t_a - t_c}$$

$$I_n(t_a, t_b) = \frac{W_1}{t_b - t_a}$$

$$I_n(t_b, t_c) = \frac{W_2}{t_c - t_b}.$$

Without loss of generality, let $I_n(t_a, t_b) \ge I_n(t_a, t_c)$, i.e.,

$$\frac{W_1}{t_b - t_a} \ge \frac{W_1 + W_2}{t_c - t_a}.$$

After some simple transformations, we have

$$\frac{W_2}{t_c - t_b} \le \frac{W_1 + W_2}{t_c - t_a}.$$

Thus (4) is true. Similarly, (5) can be proved to be correct. □

Since having no idle time is one of the key properties required for S_n, we give the following definition to precisely capture the idle time related concepts.

Definition 3.5. **(Job Busy Interval)** Interval $[t_a, t_b]$ is a J_n-*busy interval*, if the following conditions are satisfied:

—t_a, t_b are J_n scheduling points.
—$T_E(n) \le t_a \le R_n < t_b \le T_L(n)$.

—If speed $I_n(t_a, t_b)$ is applied within $[t_a, t_b]$, the processor is kept busy in $[t_a, t_b]$ by executing jobs with priorities higher or equal to that of J_n.

Note that, in Figure 3, intervals such as [11, 17] and [10,19] are J_5-busy intervals, but interval [5, 19] is not a J_5-busy interval since the processor will be idle from 7 to 10 if the speed $I_5[5, 19] = 1/2$ is applied within this interval. For a J_n-busy interval, we have the following lemma.

LEMMA 3.6. *An interval $[t_a, t_b]$ is a J_n-busy interval* if and only if

$$I_n(t_a, t) \geq I_n(t_a, t_b) \tag{6}$$

for every J_n-scheduling point $t \in (t_a, t_b]$.

Proof: Let $t \in (t_a, t_b]$. Since $I_n(t_a, t) \geq I_n(t_a, t_b)$, and $I_n(t_a, t) \times (t - t_a) \geq I_n(t_a, t_b) \times (t - t_a)$, the workload in $[t_a, t]$ is either not finished or finished at t if the processor speed is $I_n(t_a, t_b)$. Since this is true for any $t \in (t_a, t_b]$, the processor is not idle nor is executing any lower priority jobs within interval $[t_a, t_b]$. On the other hand, let the processor with constant speed $I_n(t_a, t_b)$ be always busy during $[t_a, t], t \in (t_a, t_b]$. Then, according to Definition 3.3, we have $I_n(t_a, t) \times (t - t_a) \geq I_n(t_a, t_b) \times (t - t_a)$, and thus $I_n(t_a, t) \geq I_n(t_a, t_b)$. □

For job J_n, there may exist a number of J_n-busy intervals as shown before. The largest one among them is particularly interesting and we give a definition for it below.

Definition 3.7. **(Job Essential Interval)** 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

$$t_s \leq t_a \quad \textit{and} \quad t_b \leq t_f. \tag{7}$$

The J_n-intensity corresponding to the J_n-essential interval possesses the properties of S_n stated earlier in this section. This is summarized in the following important lemma.

LEMMA 3.8. *The J_n-essential interval, $[t_s, t_f]$, and the corresponding J_n-intensity, $I_n(t_s, t_f)$, satisfy*

$$I_n(t, t_s) < I_n(t_s, t_f), \quad T_E(n) \leq t < t_s, \tag{8}$$

and

$$I_n(t_f, t) > I_n(t_s, t_f), \quad t_f < t \leq T_L(n), \tag{9}$$

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 assuming all higher priority jobs arriving before t_s are finished.

Proof: We prove (8) by contradiction. Assume there exists $t_0 \in [T_E(n), t_s)$, such that $I_n(t_0, t_s) \geq I_n(t_s, t_f)$. Let t_m be such that

$$I_n(t_m, t_s) = \max_i I_n(t_i, t_s), \quad t_m, t_i \in [T_E(n), t_s). \tag{10}$$

Then,

$$I_n(t_m, t_s) \geq I_n(t_s, t_f). \tag{11}$$

We want to show that under such assumption (i.e., (11)) $[t_m, t_f]$ is a J_n-busy interval. We consider two cases separately.

—For any t such that $t_m < t \leq t_s$, since $I_n(t_m, t_s) \geq I_n(t, t_s)$ (according to (10)), from Lemma 3.4, we have $I_n(t_m, t) \geq I_n(t_m, t_s)$. From Lemma 3.4 and (11) we have $I_n(t_m, t) \geq I_n(t_m, t_f)$.

—For any t such that $t_s < t \leq t_f$, from Lemma 3.6, we have $I_n(t_s, t) \geq I_n(t_s, t_f)$. Incorporating (11), we get $I_n(t_m, t) \geq I_n(t_s, t_f)$. From Lemma 3.4, we have $I_n(t_s, t_f) \geq I_n(t, t_f)$. Hence, $I_n(t_m, t) \geq I_n(t_s, t_f) \geq I_n(t, t_f)$. Again, from Lemma 3.4, we have $I_n(t_m, t) \geq I_n(t_m, t_f)$.

Thus for any t such that $t_m < t \leq t_f$, we have

$$I_n(t_m, t) \geq I_n(t_m, t_f),$$

and by Lemma 3.6, $[t_m, t_f]$ is a J_n-busy interval. Since $t_m < t_s$, this violates Definition 3.7.

We can prove inequality (9) similarly. Assume there exists $t_0 \in (t_f, T_L(n)]$, such that $I_n(t_f, t_0) \leq I_n(t_s, t_f)$. Let t_m be such that

$$I_n(t_f, t_m) = \min_i I_n(t_f, t_i), \quad t_m, t_i \in (t_f, T_L(n)]. \tag{12}$$

Then,

$$I_n(t_f, t_m) \leq I_n(t_s, t_f). \tag{13}$$

We want to show that under such assumption (i.e., equation (13)) $[t_s, t_m]$ is a J_n-busy interval. We consider two cases: (1) $t_s < t \leq t_f$; and (2) $t_f < t \leq t_m$. Following the similar procedure as above, we can prove that $[t_s, t_m]$ is a J_n-busy interval and violates Definition 3.7.

To show that J_n is completed by its deadline, we only need to note that, assuming all higher priority jobs arriving before t_s are finished, J_n can be finished by t_f if $I_n(t_s, t_f)$ is used as the processor speed within $[t_s, t_f]$, and $t_f \leq D_n$. □

According to Lemma 3.8, $I_n(t_s, t_f)$ is the valid minimum constant speed S_n. Thus, determining the minimum constant speed for a job now becomes determining the essential interval and corresponding intensity associated with the job. Such an algorithm can be simply implemented based on Definition 3.7. In the following, we present another algorithm, Algorithm 1 (see Figure 4), which also follows the basic principle laid down in Definition 3.7 but employs a little different search mechanism, and on average takes less time than a straightforward implementation of Definition 3.7.

For ease of understanding, we use the example in Figure 3 and compute the essential interval for J_5 with Algorithm 1. An interval is initiated to be $[R_5, R_5]$ and will grow increasingly in length with the iteration of the *while* loop in the algorithm. Algorithms 1 first searches for the scheduling point t_1 such that for any other scheduling point t located on the right hand side of the interval we have $I_5(R_5, t_1) \leq I_5(R_5, t)$ (If a tie occurs, take the right most one.) In this example, $t_1 = 19$. Then the algorithm searches for scheduling point t_2 from those scheduling points on the left hand side of the interval such that $I_5(t_2, t_1) \geq I_5(t, t_1)$ (If a tie occurs, take the left most one.) We have $t_2 = 10$ in this example. The algorithm continues in this fashion until the interval stops growing any longer, and the resultant interval, i.e., [10,19], is the essential interval for J_5.

Input: Job set $\mathcal{J} = \{J_1, ..., J_N\}$, and $T_E(n)$ and $T_L(n)$ for job J_n
Output: J_n-essential interval $[t_s, t_f]$ and S_n
$t'_a = t'_b = R_n$, $t_a = T_E(n)$, $t_b = T_L(n)$;
$I_{\max} = \infty$;
while ($t_a \neq t'_a$ or $t_b \neq t'_b$)
  $t_a = t'_a$, $t_b = t'_b$;
  $t = t_b$;
  $I_{\min} = I_{\max}$;
  for every J_n-scheduling point $t' \in (t_b, T_L(n)]$ in increasing order
    Compute $I(t_a, t')$ incrementally based on $I(t_a, t)$;
    if $I(t_a, t') < I_{\min}$
      $t'_b = t'$;
      $I_{\min} = I(t_a, t')$;
    end if
    $t = t'$;
  end for
  $I_{\max} = I_{\min}$;
  $t = t_a$;
  for every J_n-scheduling point $t' \in [T_E(n), t_a)$ in decreasing order
    Compute $I(t', t'_b)$ incrementally based on $I(t, t'_b)$;
    if $I(t', t'_b) > I_{\max}$
      $t'_a = t'$;
      $I_{\max} = I(t', t'_b)$;
    end if
    $t = t'$;
  end for
end while
$t_s = t_a, t_f = t_b, S_n = I(t_a, t_b)$;

Fig. 4. Algorithm 1: the algorithm for constructing the essential interval for a single job

To show that Algorithm 1 indeed produces the correct J_n-essential interval, we only need to prove that t_s and t_f obtained from Algorithm 1 satisfies (7). The following lemmas and theorem are sufficient for this purpose.

LEMMA 3.9. *At the end of each iteration of the* **while** *loop in Algorithm 1,* $[t_a, t_b]$ *is a* J_n-*busy interval.*

Proof Outline: Figure 5 describes the execution steps of Algorithm 1. At the end of each iteration of the *while* loop in Algorithm 1, we get interval $[t_2, t_1]$, $[t_4, t_3]$, and so on. Consider $[t_2, t_1]$. For $t \in (t_2, R_n]$, since $I_n(t_2, t_1) \geq I_n(t, t_1)$, by Lemma 3.4, we have $I_n(t_2, t) \geq I_n(t_2, t_1)$. Similarly, for $t \in (R_n, t_1]$, since $I_n(t, t_1) \leq I_n(R_n, t_1) \leq I_n(t_2, t_1)$, so $I_n(t_2, t_1) \geq I_n(t_2, t_1)$. According to Lemma 3.6, $[t_2, t_1]$ is a busy interval. Other intervals can be proved similarly.

LEMMA 3.10. *The final interval produced by Algorithm 1 satisfies*

$$I_n(t, t_s) < I_n(t_s, t_f), \quad T_E(n) \leq t < t_s, \tag{14}$$

and

$$I_n(t_f, t) > I_n(t_s, t_f), \quad t_f < t \leq T_L(n). \tag{15}$$

Proof: Let $T_E(n) \leq t < t_s$ as shown in Figure 5. If $I_n(t, t_s) \geq I_n(t_s, t_f)$, by Lemma 3.4, we have $I_n(t, t_f) \geq I_n(t_s, t_f)$. This cannot be true since in Algorithm

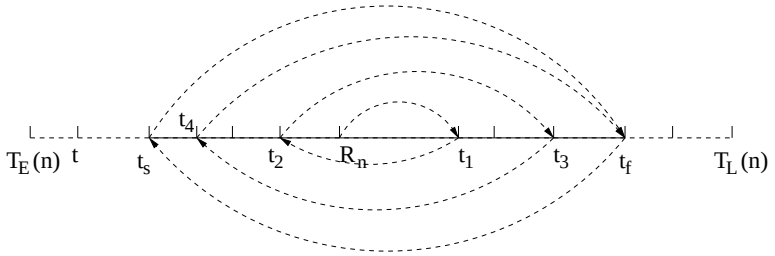

Fig. 5. This illustration shows the sequence of steps taken to construct J_n-essential interval according to Algorithm 1.

1, $I_n(t_s, t_f)$ is the maximum. Similarly, we can prove that

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

is not possible either. □

THEOREM 3.11. *Algorithm 1 produces, in $O(N^2)$ time, the J_n-essential interval.*

Proof: According to Lemma 3.8 and Lemma 3.9, we only need to prove that after the completion of Algorithm 1, there is no J_n-busy interval $[t_a, t_b]$ such that

$$t_a < t_s \quad or \quad t_b > t_f.$$

We use contradiction to prove the theorem. There are three cases to be considered.

—Case 1: $t_a < t_s \leq R_n < t_b \leq t_f$. Since $[t_a, t_b]$ and $[t_s, t_f]$ are both busy intervals, according to Lemma 3.6 and Lemma 3.4, we have

$$I_n(t_a, t_s) \geq I_n(t_a, t_b) \geq I_n(t_s, t_b) \geq I_n(t_s, t_f),$$

which contradicts (14).

—Case 2: $t_a < t_s \leq R_n < t_f < t_b$. Since $[t_a, t_b]$ and $[t_s, t_f]$ are both busy intervals, by Lemma 3.6 and Lemma 3.4,we have

$$I_n(t_a, t_s) \geq I_n(t_a, t_b) \geq I_n(t_s, t_b) \geq I_n(t_f, t_b),$$

and according to Lemma 3.10, we have

$$I_n(t_f, t_b) > I_n(t_s, t_f).$$

So,

$$I_n(t_a, t_s) > I_n(t_s, t_f),$$

which violates (14).

—Case 3: $t_s \leq t_a \leq R_n < t_f < t_b$. Since $[t_a, t_b]$ and $[t_s, t_f]$ are both busy intervals, according to Lemma 3.6 and Lemma 3.4, we have

$$I_n(t_s, t_f) \geq I_n(t_a, t_f) \geq I_n(t_f, t_b),$$

which contradicts (15).

To get the complexity of the algorithm, it is easy to see that the "while" loop takes $O(N)$ time since the maximum number of scheduling points for J_n is no

more than $O(N)$. Furthermore, by incrementally computing the intensity of each subsequent scheduling-point interval, each of the "for" loop also takes only $O(N)$ times. Therefore, the complexity of Algorithm 1 is bounded by $O(N^2)$. □

4. DETERMINING THE GLOBAL VOLTAGE SCHEDULE

In this section, we make use of the observations and Algorithm 1 discussed in Section 3 to derive the algorithm for solving the problem stated in Section 2. We will also elaborate the limitations and extensions of the new algorithm.

4.1 The overall algorithm

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

Definition 4.1. **(Critical Interval)** The essential interval $[t_s, t_f]$ with the largest J_n-intensity is called the *critical interval* of job set J.

The critical interval of $\mathcal{J}$ indicates the minimum constant speed, $S = \max_n S_n$, needed for $\mathcal{J}$ to be feasible. If $S_n = I_n(t_s, t_f)$ is used in $[t_s, t_f]$, which guarantees to finish 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 on the rest of the jobs. 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 that have higher priorities than J_n and are 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 as well as 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 (see Figure 6).

For the example shown in Figure 3, one can readily identify the first critical interval to be [10,19] with intensity as 5/9. Along with the removal of this interval, (*i*) job J_3 and J_5 are removed, (*ii*) the deadline of J_4 is changed to 10, and (*iii*) J_2 is "shifted" left correspondingly, i.e., with its updated release time and deadline as 11 and 16, respectively. Algorithm 2 continues until all the jobs are removed and we obtain a set of intervals $\mathcal{T}$ and their corresponding constant speeds $\mathcal{S}$. In the following, we present two theorems describing the important characteristics of $\mathcal{T}$ and $\mathcal{S}$.

Input: Job set $\mathcal{J} = \{J_1, ..., J_n\}$
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 $1 \leq k \leq K$
k =1;
while ($\mathcal{J}$ is not empty)
for every $J_i \in \mathcal{J}$
$\mathcal{A}, \mathcal{S}' = \emptyset$;
Determine J_i-essential interval $[t_s, t_f]$ and minimum speed S_i by Algorithm 1;
Add $[t_s, t_f]$ to $\mathcal{A}$ and S_i to S';
end for
Find j such that $S_j = \max S_i, i = 1, \cdots, n$;
Select the interval $[t_s, t_f]$ associated with S_j from $\mathcal{A}$;
Add S_j as $\overline{S}(t_s^k, t_f^k)$ to $\mathcal{S}$ and add the $[t_s, t_f]$ as $[t_s^k, t_f^k]$ to $\mathcal{T}$;
forevery $J_i \in \mathcal{J}$
if ($t_s \leq R_i \leq t_f$ and $i < j$)
Remove J_i from $\mathcal{J}$;
end if
if($t_s \leq R_i \leq t_f$)
$R_i = t_s$;
elsif ($R_i > t_f$)
$R_i = R_i - (t_f - t_s)$;
end if
if($t_s \leq D_i \leq t_f$)
$D_i = t_s$;
elsif($D_i > t_f$)
$D_i = D_i - (t_f - t_s)$;
end if
end for
k++;
end while

Fig. 6. Algorithm 2: the algorithm for constructing the set of critical intervals for $\mathcal{J}$.

THEOREM 4.2. *Given a job set $\mathcal{J}$, let $[t_s^k, t_f^k]$ and $\overline{S}(t_s^k, t_f^k)$ for $1 \leq k \leq K$ 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 $[t_s^k, t_f^k]$.*

The proof of Theorem 4.2 is rather long and given in the Appendix. From Theorem 4.2, we conclude that the set of critical intervals and their associated speeds obtained by Algorithm 2 form a valid voltage schedule. Furthermore, the speed for each critical interval is the lowest constant speed possible for that interval. However, we still do not know if any of the speeds from Algorithm 2 can be used as the overall minimum constant speed for the entire job set $\mathcal{J}$. The following theorem helps answer this question.

THEOREM 4.3. *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: $\overline{S}(t_s^1, t_f^1) \geq \overline{S}(t_s^2, t_f^2) \geq \cdots \geq \overline{S}(t_s^K, t_f^K)$.*

The proof for this theorem is also given in the Appendix. Based on Theorems 4.2 and 4.3, we have the the following corollaries.

COROLLARY 4.4. *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.

Proof: According to Theorem 4.3, the first speed is the highest necessary processor speed among all the critical intervals. Note that each job is associated with one critical interval and is schedulable as long as the processor speed is no less than the minimum speed for the critical interval, so all the jobs are schedulable if the first speed in the speed set produced by Algorithm 2 is applied throughout the entire execution period. □

COROLLARY 4.5. *The voltage schedule obtained by Algorithm 2 always saves more dynamic energy than the one that applies the minimum constant speed when the processor is busy while shuts down the processor when it is idle.*

Proof: Let the minimum constant processor speed be S_H, and the speed when it is shut down be S_L. From Lemma 4.3, the processor speed by Algorithm 2 can be represented as

$$S(t) = \alpha(t)S_L + (1-\alpha(t))S_H, \quad 0 \le \alpha(t) \le 1. \tag{16}$$

Also, let the total execution time of the job set be T, the dynamic power consumption when processor runs at speed $S(t)$ be $g(S(t))$, and the the total time when processor running at the minimum constant speed be t_1. We want to prove that

$$\begin{aligned} \int_0^T g(S(t))dt &\le \int_0^{t_1} g(S_H)dt + \int_{t_1}^T g(S_L)dt \\ &= g(S_H)T + (g(S_L) - g(S_H))(T - t_1). \end{aligned} \tag{17}$$

Since $g(x)$ is a convex function, from (16), we have

$$g(S(t)) \le \alpha(t)g(S_L) + (1-\alpha(t))g(S_H),$$

and thus,

$$\begin{aligned} \int_0^T g(S(t)) &\le \int_0^T \alpha(t)g(S_L)dt + \int_0^T (1-\alpha(t))g(S_H)dt \\ &= g(S_H)T + (g(S_L) - g(S_H)) \int_0^T \alpha(t)dt. \end{aligned} \tag{18}$$

Moreover, since

$$\int_0^T S(t)dt = \int_0^{t_1} S_H dt + \int_{t_1}^T S_L dt, \tag{19}$$

by (16) and (19), after several transformations, we have

$$\int_0^T \alpha(t) = T - t_1. \tag{20}$$

With (18) and (20), it is easy to see that (17) is correct. □

4.2 Discussions

In this section, we discuss the limitations of our approach and extensions of our approach to address more practical issues. In particular, we consider the fact that

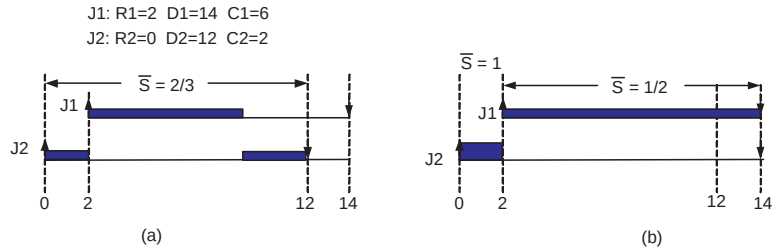

Fig. 7. An illustrative example with two jobs. (a) The voltage schedule computed according to Algorithm 2 dynamic energy consumption 3.56. (b) A better voltage schedule with dynamic energy consumption 3.5.

most commercial processors only have discrete supply voltage levels and suffer from transition overhead. Furthermore, we examine the challenges of reducing both dynamic and leakage power consumption for a real-time embedded system.

Our approach to constructing a low-energy voltage schedule guarantees to result the minimum peak dynamic power consumption. However, our algorithm may not always produce the minimum dynamic energy voltage schedule. Such as example is shown in Figure 7.

Figure 7(a) shows the voltage schedule computed from our algorithms. According to Algorithm 1, the J_1 essential interval is [2,14] with intensity of $\frac{6}{12} = 1/2$, and the J_2 essential interval is [0,12] with intensity of $\frac{8}{12} = 2/3$. Therefore, the first critical interval which is also the only critical interval for this job set is [0,12] with speed 2/3. Using this voltage schedule, we can easily compute the corresponding dynamic energy consumption as 3.56 (assuming $P = S^3$). With this schedule, both J_1 and J_2 finish at $t = 12$. However, if we allow J_2 to finish at $t = 2$ and thus J_1 can finish at $t = 14$ as shown in Figure 7(b), the dynamic energy consumption is only 3.5. This is because that our approach is a greedy approach and always strive to find the minimum constant speed during any critical interval. Note that computing the optimal solution (e.g. [Quan and Hu 2003]) takes exponential time, while our approach takes only polynomial time. In addition, as we will show in the following experimental section that energy savings achieved by applying our algorithm is very close to that by the optimal approach in general.

Up to now, we have assumed that the voltage (or the speed) of the processor can be varied continuously. However, practical commercial processors support only several discrete level supply voltages. To deal with discrete processor speeds, one approach is simply to round up the processor speed computed with Algorithm 2 to its immediate higher level. As higher processor speeds are used, all jobs can still meet their deadlines. This approach has the advantage that it does not increase the number of speed transitions and thus the associated timing and energy overhead. The disadvantage of such approach is that it uses "higher-than-necessary" processor speeds to execute real-time jobs and is not most energy efficient.

An alternative approach for handling discrete voltage levels is to incorporate the method introduced in [Ishihara and Yasuura 1998; Kwon and Kim 2005], i.e., using two available neighboring processor speeds immediate above or below the desired

processor speed computed from Algorithm 2. More specifically, let the processor speed computed from Algorithm 2 be S_n for job J_n[1], and let available processor speeds immediate above and below S_n be S_u and S_d, respectively. If we run J_n with S_u for t_u time units and with S_d for t_d time units, where

$$t_u = \frac{S_n - S_d}{S_u - S_d} \times \frac{C_n}{S_n}, \tag{21}$$

and

$$t_d = \frac{C_n}{S_n} - t_u. \tag{22}$$

Then J_n can meet its deadline and the energy cost for executing J_n is minimized. For more detailed proof of this conclusion, readers can refer to Theorem 3.1 in [Kwon and Kim 2005]. While this approach can further reduce the energy consumption with discrete processor speeds, the price to pay is the increased number of transitions, i.e., extra one for each job when its desired processor speed is not available. Note that the work in [Kwon and Kim 2005] assumes that a valid voltage schedule has already been found. Therefore, the approach cannot guarantee to produce a minimum energy schedule since the voltage schedule was determined without consideration of discrete levels.

Another limitation of our algorithm is that it assumes transition overhead is negligible. This assumption is acceptable if job timing parameters , particularly deadlines, are much longer than the time which it takes to transition between different voltage levels. Whether the above is true depends on the actual system under consideration. For systems where the transition time of the DVS processor is in tens or even hundreds of microseconds, the above assumption tends to hold.

When transition time is much longer, Algorithm 2 may produce invalid schedules (i.e., schedules that either violate job deadlines or requires voltage values exceeding the maximum allowed). To deal with this problem, the authors in [Mochocki et al. 2004] proposed a technique for systems scheduled by the EDF policy. The technique is based on the iterative algorithm of finding voltage schedules in [Yao et al. 1995] and guarantees to produce valid voltage schedules for EDF systems regardless of the magnitude of the transition overhead. Observe that Algorithm 2 has an overall structure similar to that of the algorithm in [Yao et al. 1995] except that the former is for FP scheduling while the latter is for EDF scheduling. By *overall structure*, we mean that both algorithms proceed by finding the critical interval of a job set, removing the interval and the associated jobs, then repeating the process. The algorithms in [Mochocki et al. 2004] essentially exploits this iterative nature of the algorithm in [Yao et al. 1995]. Therefore, to handle transition overhead for FP systems, the general approach in [Mochocki et al. 2004] can be used in combination with Algorithm 2 in a manner similar to what was one for EDF systems. We are working out the details of this approach [Mochocki et al.].

Since neither the optimal approach introduced in [Quan and Hu 2003] nor the heuristic method described in [Yun and Kim 2003] is based on the concept of critical intervals, it is not clear how they can be readily exploited by the technique proposed in [Mochocki et al. 2004]. In fact, the heuristic in [Yun and Kim 2003] adopts a

[1] S_n is the speed for the critical interval that contains J_n in Algorithm 2.

Input: $\mathcal{J}$, s_{th}, and T_{min} (s_{th} is the processor speed limiter.)
Compute the dynamic voltage schedule and s_n, $n = 1, 2, ..., N$;
// s_n is the minimal speed for J_n to meet its deadline
Let $s_n = s_{th}$ if $s_n \leq s_{th}$, $n = 1, 2, ..., N$;
if (processor is not idle)
  Run job J_i in the ready queue according to the FP or EDF DVS schedule;
else
  Compute the latest starting time, i.e., $LST(\mathcal{J}_n)$, for future jobs;
  if$(LST(\mathcal{J}_n) - t_{cur} > T_{min})$ // t_{cur} is the current time
    Shut down the processor and set up the wake up timer to be $LST(\mathcal{J}_n) - t_{cur}$;
  end if
end if

Fig. 8. Algorithm 3: the algorithm to reduce both dynamic and leakage power consumption for real-time systems.

similar approach as in [Quan and Hu 2003] except that the former reduces the search space by using certain observations. We are not aware of any results to overcome the limitations faced by the techniques introduced in [Quan and Hu 2003; Yun and Kim 2003].

Finally, our approach reduces the energy consumption by lowering the processor supply voltage as small as possible. While reducing processor supply voltage is efficient in reducing the dynamic energy, it becomes less effective with the leakage power consumption increasing dramatically as IC circuits continue to scale. To reduce the overall energy, the dynamic and leakage energy consumption need to be reduced collaboratively. In recent work, Irani et. al. [Irani et al. 2003] and Jejurikar et. al. [Irani et al. 2003; Jejurikar et al. 2004] proposed to set a limiter for the processor speed. The limiter is determined as follows. Consider a job with workload w. Let the total power of a processor during its active mode be $P_{act}(s)$. Then the total energy, i.e., $E_{act}(s)$, consumed to finish this job with speed s, can be represented as

$$E_{act}(s) = P_{act}(s) \times \frac{w}{s}. \tag{23}$$

The energy-minimal speed can be determined by setting $\frac{dE_{act}(s)}{ds} = 0$, i.e.,

$$P_{act}(s) = P'_{act}(s)s. \tag{24}$$

Equation (24) computes the most energy efficient speed (s_{th}) to finish one job. To increase or decrease the processor speed above or below s_{th} will increase either the dynamic or leakage power, and thus the total active power consumption for executing the job. This immediate leads to the algorithm (see Figure 8) for reducing the total energy.

As shown in Algorithm 3, the unconstrained DVS schedule produced by our algorithm can be readily incorporated into this approach (line 3). Since Algorithm 3 may use processor speeds "higher than necessary" to execute the real-time jobs, which may result in a large number of short and scattered idle intervals, some research efforts (e.g., [Jejurikar et al. 2004; Quan et al. 2004]) have been made to merge these idle interval so that the processor can be shut down to save the leakage power consumption. However, the detailed algorithm for doing so is out

of the scope of this paper. Further, finding a voltage schedule that minimizes the overall energy when shutdown overhead is not negligible is still an open problem.

5. EXPERIMENTAL RESULTS

In this section, we present experiment results for comparing the dynamic energy savings and computational efficiency achieved by ours and some previous work, i.e. [Quan and Hu 2003; Shin and Choi 1999][2], for FP real-time systems. All these experiments are conduct on Sun Blade-100. For brevity, we use **VSLP**,**LPFS**, and **OPT** in the following to represent Algorithm 2, the algorithm in [Shin and Choi 1999], and the optimal one in [Quan and Hu 2003], respectively.

We use three processor models in our experiments: a theoretical model P_1 and two practical models, i.e., P_2 and P_3. In P_1, based on the work in [Burd 2001], we assume that the processor speed is proportional to the supply voltage and the processor power consumption is a cubic function of the processor speed. Making these assumptions helps to collect concrete experimental data for the comparison purpose. Also, we ignore the discrete voltage levels and transition overhead in this model. Processor models P_2 and P_3 are derived from the commercial processor Transmeta TM5400 [Transmeta-Corporation 2000] and StrongARM SA-1100 [Intel], respectively. In these two models, the voltage can only vary in a limited range, and only a number of levels of working frequencies/voltages are available. Table I (adopted from [Pouwelse et al. 2001] and [Sinha and Chandrakasan 2001]) summarizes the relation between frequency, voltage, and power consumption for Transmeta TM5400 and StrongARM SA-1100. When the desired processor speed is not available, we adopt the approach in [Transmeta-Corporation 2000], i.e., use two neighboring processor speeds immediately above and below the non-existing speed to optimize the energy consumption. According to [Transmeta-Corporation 2000], the transition overhead is less than $20\mu s$. This overhead is not significant for most real-time systems where task timing parameters are on the order of milliseconds and is therefore ignored in our experiments.

We use two groups of real-time job sets in our experiment to evaluate the performance of our approach, one is randomly generated, and the other one is drawn from practical applications. In our first set of experiments, we use 10 groups of randomly generated real-time systems with the number of jobs being 2, 4, $\cdots$, 20. The arrival times and deadlines of these jobs are chosen to be uniformly distributed within $[0, 50]ms$, $[20, 100]ms$, respectively. These data are randomly chosen without special considerations. The execution time of each job is randomly generated from 1 to half of its deadline to make the job sets easier to schedule under the maximum processor speed. Only the job sets that are schedulable under the maximum processor speed are used in our experiment, and each group contains up to 100 such schedulable job sets. To reduce statistical errors, we collected the average energy

[2]While our algorithm's complexity is proven to be lower than the algorithm in [Yun and Kim 2003] discussed in the introduction, it would be desirable to compare the two through experiments as well. However, through our experiments, we have found some inconsistencies in the experiment results reported in [Yun and Kim 2003]. Furthermore, since the experimental data presented in [Yun and Kim 2003] are not compared with the optimal results, and the implementation details and initial test data are not available to us, we are not able to make a fair comparison.

Processors	Frequency (Mhz)	Voltage (V)	Relative Power (%)
P2	700	1.65	100
	600	1.60	80.59
	500	1.50	59.03
	400	1.40	41.14
	300	1.25	24.60
	200	1.10	12.70
P3	206	1.50	100
	195	1.42	78.9
	180	1.30	63.2
	165	1.20	50.0
	150	1.15	39.9
	135	1.10	33.6
	120	1.08	33.0
	105	0.95	19.8
	90	0.90	15.0
	75	0.82	11.8
	60	0.80	9.44

Table I. Clock frequency *vs.* supply voltage *vs.* relative power for Transmate Crusoe processor and StrongARM SA1100

consumption for each group, and normalized them using the optimal results, *i.e.*, the ones computed with **OPT**. Figure 9 shows the average dynamic energy savings for each generated job sets with the theoretical processor model, i.e., processor P_1, and Figure 10 shows the results with one of the practical processor models, i.e., processor P_2. [3] Moreover, to compare the computational cost, we also gather the average CPU times of **LPFS**, **VSLP** and **OPT**, and depict them in Figure 11.

The test cases in our second set of experiments contain three real-world applications: video phone [Shin et al. 2001], CNC [N.Kim et al. 1996] and INS [A.Burns et al. 1995]. We tested these application examples with all three processor models. In this set of experiments, we collected the energy consumption by each approach, normalized it with the energy consumption using processor without DVS capability, and filled in Table II. The CPU times for **VSLP** and **LPFS** were also collected and filled Table II. Note that the test data of using **OPT** on CNC and INS are unavailable because of its excessive requirement of computation time.

From the experimental results shown in Figure 9 and Figure 11, one can readily conclude that our voltage scheduling strategy is very effective in terms of both energy savings and computation cost. Note that, in Figure 9, the energy consumption by **VSLP** is very close to that of **OPT** (within 2% for our randomly generated examples) and much better than that of **LPFS**. The energy efficiency of our approach comes from the fact that **LPFS** always uses the full speed to execute the jobs when the ready queue is not empty, but in our approach many of the jobs can be in fact executed with much lower speed according to the voltage schedule obtained by Algorithm 2. Therefore, our approach can exploit the processor slack time more efficiently and achieve better energy saving performance than **LPFS**. On the other hand, the CPU time for **VSLP**, comparable with that of **LPFS**, is much less than

[3] Processor model P_3 yielded very similar results to that with P_2 for the randomly generated job sets and thus omitted.

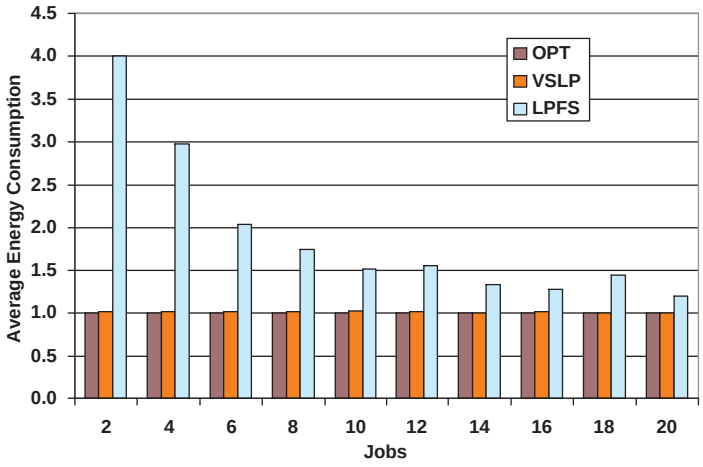

Fig. 9. Average dynamic energy saving performance comparison for **OPT**, **VSLP**, and **LPFS** using randomly generated job sets and the theoretical processor model, i.e., processor P_1.

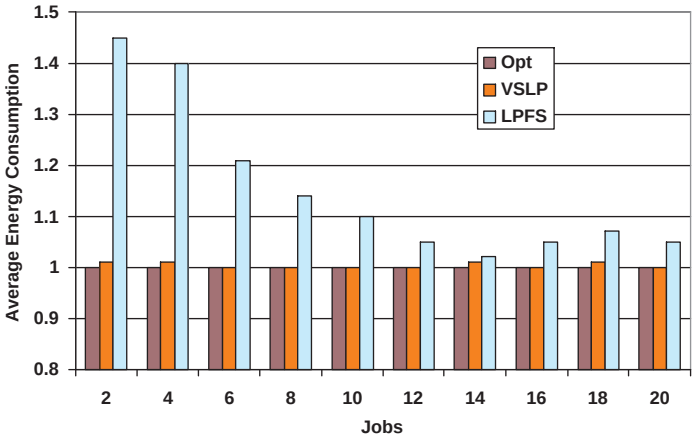

Fig. 10. Average dynamic energy saving performance comparison for **OPT**, **VSLP**, and **LPFS** using randomly generated job sets and the practical processor model, i.e., P_2.

that for the **OPT**. As shown in Figure 11, it can be very costly, if not impractical, to apply **OPT** in real applications that have a large number of jobs. Moreover, through our experiments, we can see that the factors such as the discrete voltage levels do impact the energy saving potential of our approaches as demonstrated in Figure 10. However, even with the considerations of these practical factors, our approach can still outperform previous research significantly (up to 45% as shown

Fig. 11. Average computation cost for **OPT**, **VSLP**, and **LPFS**.

Table II. Experimental results for three practical applications.

Systems			video phone	CNC	INS
Job Numbers			16	289	2147
Energy Consumption	P1	OPT	0.952	*N/A**	*N/A**
		VSLP	0.952	0.24	0.54
		LPFS	0.955	0.63	0.76
	P2	OPT	0.99	*N/A**	*N/A**
		VSLP	0.99	0.65	0.85
		LPFS	0.99	0.81	0.88
	P3	OPT	0.953	*N/A**	*N/A**
		VSLP	0.953	0.40	0.58
		LPFS	0.956	0.58	0.81
CPU Time(s)		VLSP	0.00	0.23	68.31
		LPFS	0.00	0.01	0.55

* These results are unavailable due to the excessive computation time requirement by **OPT**.

in Figure 10).

The experimental results for the practical applications shown in Table II also conform with our analysis above. **OPT** can only be applied to real-time systems with small numbers of jobs, such as the video phone [Shin et al. 2001], it becomes ineffective for more complicated real-time systems such as CNC and INS. While **VSLP** consumes more CPU time than that by **LPFS**, it produces much better voltage schedules for real-time systems in terms of energy saving. It is interesting to note that, with the theoretical processor model (i.e., processor P_1), our approach finds the theoretically optimal voltage schedule for the video phone application even though it is heuristic in nature. It is also interesting to note that **VSLP** is more energy efficient on processor P_2 than on processor model $P3$ for all three applications. This is because P_3 provides more voltage levels (according to [Sinha and Chandrakasan 2001]) than processor model P_2 (according to [Pouwelse et al.

2001]) does. Therefore the discrete voltage levels has less impacts on the energy saving performance for **VSLP**. While the practical factors of the processor has limited the energy potential of our approach, from Table II, our approach can still reduce 16% energy consumption on processor model P_2 and 32% on processor model $P3$ for CNC application.

6. SUMMARY AND FUTURE WORK

DVS scheduling techniques are critical energy-saving techniques not only for the current embedded systems in which dynamic energy consumption predominates in the total energy consumption, but also for the future generation of such systems when static grows significantly with the continuous evolvement of IC technology. In this paper, we present an off-line, heuristic DVS scheduling technique for an FP real-time embedded system. 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 a single job. The second algorithm, with $O(N^3)$ time complexity, builds on the first one and can produce the minimum constant voltage (or speed) needed to complete a set of jobs, as well as a voltage schedule which always results in lower dynamic energy consumption compared to using the minimum constant voltage and shutting down the system when it is idle.

The minimum constant voltage is an important parameter when designing systems with no sophisticated power management hardware but only simple on/off modes or where the voltage transition overhead is a concern. The experimental results obtained from both randomly generated examples and the real-world applications have shown the effectiveness of applying our voltage schedules to save energy, even with the considerations of the limitations in the practical processors. Moreover, our approach has much less computation cost than the theoretical optimal one and therefore can be applied to more complicated real-time applications.

The proposed heuristic provides a powerful means to *iteratively* construct the voltage schedule of a job set based on critical intervals. This unique approach can be readily exploited by other heuristics [Mochocki et al. 2004; Quan et al. 2004] for handling transition overhead and leakage power. We are working on the details of such combined techniques [Mochocki et al. ; Quan et al.]. Finding optimal voltage schedules in the presence of transition overhead and non-negligible leakage power still eludes researchers. We would like to examine these problems in our future work. Another challenge is to study the impact of different switching capacitances of jobs on the voltage schedule. In [Kwon and Kim 2005], an integer linear programming approach is used to find a voltage schedule that takes into account discrete voltage levels and different switching capacitance. However, the approach assumes that the start and end time of a job has already be found. This assumption simplifies the problem and do not guarantee to find the optimal voltage schedule. We plan to investigate this problem in our future work.

7. ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the reviewers for their effort in reviewing this paper. Their valuable comments have improved the paper greatly. This research is supported in part by NSF under grant number CNS-

0545913, CCR02-08992, CNS-0410771, and in part by the University of South Carolina Research Program under the Research Scholarship Award.

APPENDIX

We present the detailed proofs for Theorem 4.2 and 4.3 in this section.

A.1 Proof for Theorem 4.2

The following lemma is needed in proving Theorem 4.2.

LEMMA A.1. *Let $T_E(n) \leq T'_E(n)$ and $T_L(n) = T'_L(n)$ in Algorithm 1, and let the two resultant J_n-essential intervals be $[t_s, t_f]$ and $[t'_s, t'_f]$, respectively. Then*

$$I_n(t_s, t_f) \geq I_n(t'_s, t'_f),$$

and $t_s \leq t'_s < t'_f \leq t_f$.

Proof: Since $T_E(n) \leq T'_E(n)$, and $T_L(n) = T'_L(n)$, from Algorithm 1, we have

$$t_s \leq t'_s < t'_f \leq t_f.$$

If $t_s \leq t'_s < t'_f = t_f$, by Lemma 3.4 and Lemma 3.6, we have

$$I_n(t_s, t'_s) \geq I_n(t_s, t_f) \geq I_n(t'_s, t'_f).$$

If $t_s \leq t'_s < t_f < t'_f$, according to Lemma 3.10, we have $I_n(t_f, t'_f) > I_n(t'_s, t'_f)$. By Lemma 3.4 and Lemma 3.6, we have

$$I_n(t'_s, t'_f) < I_n(t_f, t'_f) \leq I_n(t_s, t_f).$$

□

To prove Theorem4.2, we concentrate on demonstrating that every job removed while removing the critical interval is fully executed within the interval and meets its deadline. Without loss of generality, consider a critical interval associated with job J_n, i.e., $[t^k_s, t^k_f] = [t_s, t_f]$. From Theorem 3.11, we know that J_n is schedulable. Consider the rest of the jobs that are removed when removing $[t_s, t_f]$. Note that only the jobs with priorities higher than that of J_n and release time in $[t_s, t_f)$ are removed. For a removed job J_i other than J_n, if $t_s \leq T_E(i) < T_L(i) \leq t_f$, the J_i-essential interval must be within $[t_s, t_f]$ and have a lower speed requirement than $I_n(t_s, t_f)$, so job J_i must be schedulable. Otherwise, we only need to prove that if the original earliest and/or latest scheduling points $T_E(i)$ and $T_L(i)$ are moved to t_s and/or t_f, respectively, the constant speed for the resultant new J_i-essential interval is still less than $I_n(t_s, t_f)$.

Several cases need to be considered. For each case, we let $[t_s, t_f]$ be the critical interval, $[t_a, t_b]$ be the original essential interval for J_i, and $[t'_a, t'_b]$ be the new essential interval for J_i after the change of $T_E(i)$ and/or $T_L(i)$.

—$t_a < t_s < t_b \leq t_f$. Since $[t_s, t_f]$ is the critical interval, and earliest starting point of job J_i have to be moved to t_s. Note that moving $T_L(i)$ to t_b will not change the original essential interval $[t_a, t_b]$. Therefore, according to Lemma A.1, we have

$$I_i(t'_a, t'_b) \leq I_i(t_a, t_b) \leq I_n(t_s, t_f),$$

and thus J_i is schedulable.

—$t_s \le t_a < t_b \le t_f$. Note that moving $T_E(i)$ to t_s if $T_E(i) < t_s$ and moving $T_L(i)$ to t_f if $T_L(i) > t_f$ do not change the initial J_k-essential interval. Hence, J_i is schedulable since $I_i(t_a, t_b) = I_i(t'_a, t'_b) \le I_n(t_s, t_f)$.

—$t_a \le t_s < t_f \le t_b$. In this case, $T_E(i)$ is moved to t_s and $T_L(i)$ is moved to t_f. Suppose $I_i(t'_a, t'_b) > I_n(t_s, t_f)$. According to Lemma 3.10, $I_i(t'_b, t_f) > I_i(t'_a, t'_b)$. According to Definition 3.3, since $i < n$, we have

$$I_n(t'_b, t_f) \ge I_i(t'_b, t_f) > I_i(t'_a, t'_b) > I_n(t_s, t_f)$$

which contradicts $I_n(t_s, t_f) \ge I_n(t'_b, t_f)$. Therefore, $I_i(t'_a, t'_b) \le I_n(t_s, t_f)$, and J_i is schedulable.

—$t_s \le t_a < t_f < t_b$ In this case, $T_L(i)$ is moved to t_f and $T_E(i)$ is moved to t_s if $T_E(i) < t_s$. With a similar proof as the previous case, we can show that $I_i(t'_a, t'_b) \le I_n(t_s, t_f)$, and J_i is schedulable.

□

A.2 Proof for Theorem 4.3

To prove Theorem 4.3, we only need to show that after removing a critical interval, the minimum speeds for jobs whose essential intervals have changed are never bigger than that of the removed critical interval.

Let the critical interval $[t_s, t_f]$ be a J_n-essential interval, and J_i be a job that R_i, $T_E(i)$ or $T_L(i)$ need to be changed due to this removal. The corresponding J_i-essential interval will be re-constructed. The priority of J_i can be either higher or lower than that of J_n. We deal with them separately. In our proofs, we let $[t_s, t_f]$ be the critical interval, $[t'_a, t'_b]$ and $[t''_a, t''_b]$ represent possible locations of the J_i-essential before removing $[t_s, t_f]$, $[t_a, t_b]$ be the J_i-essential interval after removing $[t_s, t_f]$.

Note that after removing the critical interval, t_s and t_f become a single point. For the ease of reference, we still keep them apart. We will denote the intensity of an interval before (resp., after) removing the critical interval by I' (resp. I). For the critical interval, I and I' are always the same.

Case 1: Job J_i has a higher priority than J_n ($i < n$).
In this case, R_i must be outside the critical interval and either $T_E(i)$ or $T_L(i)$ is changed after removing $[t_s, t_f]$ (see Algorithm 2). If $R_i < t_s$, $T_L(i)$ is moved to t_s. Similarly if $R_i > t_f$, $T_L(i)$ is moved to t_f. In either case, according to Definition 3.7, $[t_a, t_b]$ is a busy interval contained in the original J_i-essential interval $[t'_a, t'_b]$ (or $([t''_a, t''_b])$. According to Lemma 3.8, $I_i(t_a, t_b) = I'_i(t_a, t_b) \le I'_i(t'_a, t'_b)$. Since $I_n(t_s, t_f)$ is the critical interval, we conclude $I_i(t_a, t_b) \le I_n(t_s, t_f)$.

Case 2: Job J_i has a lower priority than J_n ($i > n$).
Let us first analyze the possible locations of the original J_i-essential interval with respect to the critical interval. First, we show that $[t'_a, t'_b]$ can neither be totally contained in $[t_s, t_f]$ nor partially overlap with $[t_s, t_f]$.

Let $t_s \le t'_a < t'_b \le t_f$. According to Definition 3.3 and Lemma 3.8, we have $I'_i(t_s, t'_a) \ge I_n(t_s, t'_a) \ge I_n(t_s, t_f) \ge I'_i(t'_a, t'_b)$, this contradicts Lemma 3.8. Let $t'_a < t_s < t'_b < t_f$. Similarly, $I'_i(t'_a, t'_b) \ge I'_i(t_s, t'_b) \ge I_n(t_s, t'_b) \ge I_n(t_s, t_f)$, this contradicts $I_n(t_s, t_f) > I'_i(t'_a, t'_b)$. Therefore, to prove the theorem, we only need to consider the scenario where $[t'_a, t'_b]$ contains the critical interval.

Depending on the location of the J_i-essential interval after the removal of the critical interval, the following cases exist.

—$t_a < t'_a$: According to Lemma 3.4 and Lemma 3.6, $I_i(t_a, t_b) \leq I_i(t_a, t'_a) = I'_i(t_a, t'_a)$. From Lemma 3.8, $I'_i(t_a, t'_a) < I'_i(t'_a, t'_b) \leq I_n(t_s, t_f)$. Thus, we have $I_i(t_a, t_b) < I_n(t_s, t_f)$.

—$t_b < t'_b$: Similar to the above proof, we have $I_i(t_a, t_b) < I_n(t_s, t_f)$.

—$t'_a \leq t_f \leq t_a$ and $t_f \leq t'_b \leq t_b$: (Note that t_s and t_f become the same after removing the critical interval.) By Lemma 3.4 and Lemma 3.6, we have $I'_i(t'_a, t'_b) \geq I'_i(t_a, t'_b) = I_i(t_a, t_b) \geq I_i(t_a, t_b)$. Since $I_n(t_s, t_f) \geq I'_i(t'_a, t'_b)$, therefore, we have $I_i(t_a, t_b) \leq I_n(t_s, t_f)$.

—$t'_a \leq t_a \leq t_s$ and $t'_b \leq t_b$: Let w_1, w_2, w_3, w_4, w_5 be the workloads in $[t'_a, t_a], [t_a, t_s], [t_s, t_f], [t_f, t'_b]$, and $[t'_b, t_b]$, respectively. Furthermore, let the workloads of those jobs that are removed when removing critical interval $[t_s, t_f]$ be w'_2. Apparently $w_2 \geq w'_2$, and

$$I'_i(t_a, t'_b) = \frac{w_1 + w_2 + w_3}{t'_b - t_a} \leq I'_i(t'_a, t'_b) \leq I_n(t_s, t_f) = \frac{w'_2}{t_f - t_s}, \tag{25}$$

Suppose $I_i(t_a, t_b) > I(t_s, t_f)$. Then, by Lemma 3.6,

$$\begin{aligned} I_i(t_a, t'_b) &= \frac{w_1 + w_2 - w'_2 + w_3}{t'_b - t_a - (t_f - t_s)} > I_i(t_a, t_b) \\ &> I_n(t_s, t_f) = \frac{w'_2}{t_f(n) - t_s(n)}. \end{aligned} \tag{26}$$

Moreover, by Lemma 3.6 and Lemma 3.4, we have

$$I'_i(t'_a, t'_b) \geq I'_i(t_a, t'_b) > I_i(t_a, t'_b). \tag{27}$$

It can be readily shown that (25), (26), and (27) cannot be true simultaneously. Therefore, the intensity of the new J_i-essential interval must be less than or equal to that of the critical interval being removed.

□

REFERENCES

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

Aydin, H., Melhem, R., Mosse, D., and Alvarez, P. 2001a. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. *ECRTS*, 225–232.

Aydin, H., Melhem, R., Mosse, D., and Alvarez, P. 2001b. Dynamic and aggressive scheduling techniques for power aware real-time systems. *RTSS*, 95–105.

Burd, T. 2001. *Energy-Efficient Processor System Design*. Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, Uiversity of California, Berkeley.

Burd, T. D. and Brodersen, R. W. 2000. Design issues for dynamic voltage scaling. *ISLPED*, 9–14.

Duarte, D., Vijaykrishnan, N., Irvin, M., Kim, H., and McFarland, G. 2002. Impact of scaling on the effectiveness of dynamic power reduction schemes. *ICCD*, 382–387.

Govil, K., Chan, E., and Wasserman, H. 1995. Comparing algorithms for dynamic speed-setting of a low-power cpu. *International Conference on Mobile Computing and Networking*, 13–25.

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

Hong, I., Kirovski, D., Qu, G., Potkonjak, M., and Srivastava, M. B. 1998. Power optimization of variable voltage core-based systems. *Proceedings of DAC*, 176–181.

Hwang, C. and Wu, A. 1997. A predictive system shutdown method for energy saving of event-driven computation. *Proceedings of International Conference on Compter Aided Design*, 28–32.

Intel. Strongarm processors. *http://developer.intel.com/design/strong/sa1100.htm*.

Irani, S., Shukla, S., and Gupta, R. 2003. Algorithms for power savings. *SODA*, 37–46.

Ishihara, T. and Yasuura, H. 1998. Voltage scheduling problem for dynamically variable voltage processors. *ISLPED*, 197–202.

ITRS. http://public.itrs.net/. *International Technology Roadmap for Semiconductors*. International SEMATECH, Austin, TX.

Jejurikar, R. and Gupta, R. 2002. Energy aware edf scheduling with task synchronization for embedded real time operating systems. *COLP*, 71–76.

Jejurikar, R., Pereira, C., and Gupta, R. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. *DAC*, 275 – 280.

Kim, W., Kim, J., and Min, S. L. 2003. Dynamic voltage scaling algorithm for dynamic priority hard real-time systems using work-demand analysis. *ISLPED*, 396–401.

Kim, W., Kim, J., and S.L.Min. 2002. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack analysis. *DATE*, 788–794.

Kwon, W. and Kim, T. 2005. Optimal voltage allocation techniques for dynamicaly variable voltage processors. *ACM Transactions on Embedded Computing Systems 4,* 1, 211–230.

Lehoczky, J., Sha, L., and Ding, Y. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. *RTSS*, 166–171.

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

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

Lorch, J. R. and Smith, A. J. 2001. Improving dynamic voltage scaling algorithms with PACE. In *SIGMETRICS/Performance*. 50–61.

Makzak, A. and Chakrabarti, C. 2003. Variable voltage task scheduling algorithms for minimizing energy/power. *IEEE Transactions on VLSI 11,* 2 (April), 270–276.

Mochocki, B., Hu, X., and Quan, G. On-line and off-line dvs scheduling of fixed-priority real-time systems on practical processors. *to be submitted to IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems*.

Mochocki, B., Hu, X., and Quan, G. 2002. A realistic variable voltage scheduling model for real-time applications. *ICCAD*, 726–731.

Mochocki, B., Hu, X., and Quan, G. 2004. A unified approach to variable voltage scheduling for nonideal dvs processors. *IEEE Trans. on Computer-Aided Design for Integrated Circuits and Systems 23,* 9, 1370– 1377.

Mochocki, B., Hu, X., and Quan, G. 2005. Practical on-line dvs scheduling for a fixed-priority real-time system. *RTAS*, 224–233.

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

Nielsen, L., Niessen, C., Sparso, J., and Berkel, K. 1994. Low-power operation using self-timing circuits and adaptive scaling of supply voltage. *IEEE Transactions on VLSI and Systems 2*, 425–435.

Niu, L. and Quan, G. 2004. Reducing both the dynamic and leakage energy consumption for hard real-time systems. *CASES*, 140–148.

N.Kim, Ryu, M., Hong, S., Saksena, M., Choi, C., and Shin, H. 1996. Visual assessment of a real-time system design: a case study on a cnc controller. *RTSS*, 300–310.

Pering, T., Burd, T., and Brodersen, R. 1998. The simulation and evaluation of dynamic voltage scaling algorithms. *ISLPED*, 76–81.

Pillai, P. and Shin, K. G. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*. 89–102.

Pouwelse, J., Langendoen, K., and Sips, H. 2001. Dynamic voltage scaling on a low power microprocessor. *SIGMOBILE*, 251–259.

Quan, G. and Hu, X. 2003. Minimum energy fixed-priority scheduling for variable voltage processors. *IEEE Transactions on ICCAD 22,* 8 (August), 1062–1971.

Quan, G. and Hu, X. S. 2001. Energy efficient fixed-priority scheduling for real-time systems on voltage variable processors. *DAC*, 828–833.

Quan, G., Niu, L., Hu, X., and Mochocki, B. 2004. Fixed priority scheduling for reducing overall energy on variable voltage processors. *RTSS*, 309–318.

Quan, G., Niu, L., Mochocki, B., Hu, X., and Quan, G. Real-time scheduling for reducing the overall energy for variable voltage processors. *under review by IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems*.

Rabaey, J. and Pedram, M. 1996. *Low Power Design Methodologies*. Kluwer.

Shin, D., Kim, J., and Lee, S. 2001. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers 18,* 2 (March-April), 20–30.

Shin, Y. and Choi, K. 1999. Power conscious fixed priority scheduling for hard real-time systems. *DAC*, 134–139.

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

Sinha, A. and Chandrakasan, A. P. 2001. Jouletrack- a web based tool for software energy profiling. *DAC*, 220–225.

T. Pering, T. Burd, R. B. 2000. Voltage scheduling in the lparm microprocessor system. *ISLPED*, 96–101.

Transmeta-Corporation. January,2000. *TM5400 processor specifications*. http://www.transmeta.com/crusoe/download/pdf /TMS5400_ProductBrief_5-23-00.pdf.

Weiser, M., Welch, B., Demers, A., and Shenker, S. 1994. Scheduling for reduced cpu energy. *Proceedings of USENIX Symposium on Operating System Design and Implementation*, 13–23.

Yan, L., Luo, J., and Jha, N. 2003. Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. *ICCAD*, 30–37.

Yao, F., Demers, A., and Shenker, S. 1995. A scheduling model for reduced cpu energy. *IEEE Annual Foundations of Comp. Sci.*, 374–382.

Yun, H.-S. and Kim, J. 2003. On energy optimal voltage scheduling for fixed-prioirty hard real-time systems. *ACM Transactions on Embedded Computing Systems 2,* 3, 393–430.

Received Month Year; revised Month Year; accepted Month Year.