A Unified Approach to Variable Voltage Scheduling for Non-ideal DVS Processors *

Bren Mochocki Xiaobo Sharon Hu
Dept. of CSE
University of Notre Dame
Notre Dame, IN 46556, USA
{bmochock, shu}@cse.nd.edu

Gang Quan
Dept. of CSE
University of South Carolina
Columbia, SC 29208, USA
gquan@cse.sc.edu

Abstract

Voltage scheduling is an essential technique used to exploit the benefit of variable voltage processors. Though extensive research exists in this area, current processor limitations such as transition overhead and voltage level discretization are typically dismissed as insignificant. We show that for hard real-time applications, disregarding these details can lead to sub-optimal or even invalid results. We propose two algorithms that guarantee valid solutions. The first is a greedy yet simple approach, while the second is more complex but significantly reduces energy consumption under certain conditions. Through experimental results on both real and randomly generated systems, we show the effectiveness of both algorithms and explore what conditions make it beneficial to use the complex algorithm over the basic one.

1 Introduction

The demand for mobile and pervasive computing devices has made low power/energy computing a critical technology. One of the most effective ways to reduce energy consumption in CMOS processors is Dynamic Voltage Scaling (DVS), i.e., dynamically varying a processor's supply voltage and clock frequency simultaneously. Various DVS processors are commercially available, including Intel's XScale [1], AMD's Mobile Athlon [2], and Transmeta's Crusoe processor [3]. Several research groups have also developed their own variable voltage systems. For example, Burd and Brodersen implemented a variable voltage system using the ARM8 core [4], while Pouwelse *et al.* constructed a similar system using the SA-1100 [5]. Ideally, a DVS processor would operate at *any* voltage within a specific range and switch from one voltage to another instantaneously. However, due to physical limitations, DVS processors always incur both time and energy overhead during voltage transition. Furthermore, all commercially available DVS processors today can only operate at discrete voltage levels.

To maximally exploit the benefits of a DVS processor, voltage scheduling, the selection of voltage levels and operating frequencies, is indispensable. A large number of research results have been published on voltage scheduling for DVS processors. These results differ in many aspects, such as the type of applications (e.g., real-time or non real-time) considered, the type of systems (e.g., single or multiple processors) used,

*This work is supported in part by NSF under grant numbers MIP-9701416, CCR-9988468 and CCR02-08992.

the location (intra-task v.s. inter-task) where a voltage change is allowed, the execution style (e.g., on-line or off-line) of the voltage scheduling algorithms, etc. It is not difficult to see that non-ideal features of DVS processors, such as discrete voltage levels and transition overheads, can effect voltage scheduling results and deserves careful study.

In this paper, we focus on voltage scheduling for a set of real-time jobs executed by a single DVS processor. Many embedded applications can be described by such a model. In particular, we study off-line, inter-job voltage scheduling where the real-time jobs are executed according to the preemptive Earliest Deadline First (EDF) scheduling scheme [6]. Preemptive EDF is an optimal scheduling algorithm and has been adopted by many real-time systems [7]. Inter-job (or inter-task) voltage scheduling is realized by the operation system, which is less intrusive and more portable for a given application. Offline scheduling does not compete for resources with the actual application and hence can afford to use more sophisticated algorithms. Though off-line scheduling cannot handle dynamic situation, it can often be used in a complementary fashion with on-line approaches. The uniqueness of our work lies in that the voltage scheduling process accounts for non-ideal features of the DVS processors including time and energy transition overhead as well as discrete voltage levels. To the best of our knowledge, this is the first work to incorporate all these practical limitations into voltage scheduling algorithms for the system model under consideration.

1.1 Related Work

Substantial research has been done on voltage scheduling for real-time applications, e.g., [8, 9, 10, 11, 12, 13, 14, 15, 16]. Some of the papers have considered off-line, inter-job voltage scheduling for preemptive EDF based real-time systems, e.g., [14, 16]. A dominating trend in these efforts is to ignore the non-ideal features of DVS processors.

A limited amount of work has examined voltage scheduling in the presence practical limitations. Some of these consider only discrete voltage levels. For example, Chandrasena *et al.* [17] introduce a rate selection algorithm for a variable voltage processor with limited voltage levels, but the algorithm provides no deadline guarantee for the tasks. In [18] and [19], Lee *et al.* consider discrete voltage levels in dynamic (i.e., on-line) voltage scheduling for periodic tasks. Their method, called time slicing, requires that tasks be divided into subtasks (or slots), which is not always possible. Even if the division is possible, preemption is not allowed within a sub-task, so this method cannot be applied to the preemptive scheduling problem we address here. Kwon *et al.* give an optimal intra-task scheduling algorithm under the preemptive EDF scheme to match a discrete set of voltage levels [20]. Saewong and Rajkumar give an in-depth analysis of the ideal placement of a set of discrete voltage levels for a DVS processor, and conclude that the energy increase due to voltage/frequency quantization is inversely proportional to the number of levels [21]. Since the above approaches ignores transition overheads, they tend to introduce more transitions in order to better match discrete voltage levels, which further exasperate the impact transition overhead.

A number of researchers have studied voltage scheduling when transition overhead is not negligible.

Manzak *et al.* [22] address the transition time overhead by linearly increasing the total required execution time or decreasing the processor utilization. Such adjustments may lead to either a deadline miss or an overly pessimistic design. In [23], Hong *et al.* present a heuristic algorithm that accounts for transition overheads during static scheduling but assumes both the availability of any voltage levels and continuous execution of instructions during a transition. Neither assumptions can be satisfied by most DVS systems [2, 3, 1, 5]. AbouGhazaleh *et al.* present an intra-task voltage scheduling method that accounts for transition overhead. Their method, however, requires both compiler support and that the source code be engineered to give voltage selection "hints" to the operating system [24, 25]. Hsu and Kremer also present a compiler driven DVS algorithm, but hard deadlines are not guaranteed [26].

1.2 Our Contributions

In this paper, we present several observations to reveal the effects of transition overhead on executing real-time jobs according to the preemptive EDF scheme. These observations show that time transition overhead can cause deadline misses as well as consideration energy increase if not handled carefully. A basic algorithm is devised to guarantee that no deadline violations occurs in the presence of time transition overhead. Building on the basic algorithm, we have developed an algorithm taking full consideration of both time and energy transition overheads as well as discrete voltage levels. A large number of experiments have been conducted for both real-world examples and randomly generated examples to demonstrate the effectiveness of our algorithms. Careful analysis of the experimental results help us to draw several conclusions regarding the non-ideal features of DVS processors.

The remainder of this paper is organized as follows. Section 2 summarizes the relevant background material, including system models and motivation examples. Section 3 describes the basic algorithm that handles time transition overhead. Section 4 improves the basic algorithm in terms of energy savings and accounts for both energy overhead and discrete voltage levels, in addition to time overhead. Section 5 presents our experimental results, and Section 6 concludes the paper.

2 Preliminaries

2.1 System Model

We consider real-time applications consisting of a set of independent jobs, $\mathcal{J} = \{J_1 ... J_n\}$ with each job $J_i \in \mathcal{J}$ having a release time r_i, a deadline d_i, and worst case execution cycles c_i. The job set is to be executed on a DVS processor, whose power consumption is a convex function of the processor speed (i.e., frequency) [27]. The convexity assumption holds so long as the switching power is one of the main contributors of the total power, which is the case for current and near future CMOS devices [28]. The DVS processor can operate at a finite set of supply voltage levels $\mathcal{V} = \{V_1, ..., V_{max}\}$, each with an associated speed. To simplify the discussion, we normalize the processor speeds by S_{max}, the speed corresponding to V_{max}, and thus we have $\mathcal{S} = \{S_1, ..., 1\}$. Transitioning from one voltage level to another takes a fixed amount of time referred to

as the *transition interval* (denoted as Δt), and consumes a variable amount of *transition energy* (denoted as ΔE). No instructions are executed during a transition. The above DVS processor model captures the main features of most commercial DVS processors [2, 5]. A variable length transition interval (e.g. the one described in [4]) can be approximated by a fixed length interval equal to the maximum switching time. For processors that do not block instructions during a transition (e.g., [4]), a schedule that assumes blocking during a transition can be pessimistic, but it will guarantee to produce valid voltage schedule. As in most voltage scheduling work, we assume that each job consumes an equal amount of energy per cycle at a given speed, which is a valid assumption for many applications.

We introduce some notation below which will be used throughout the paper. We denote an *interval* by $T = [t_s, t_f]$ and the interval *length* by $|T| = t_s - t_f$. The *intensity*, $s(T)$, of an interval T is the processor speed required to finish all the jobs inside the interval by t_f. It can be readily verified that

$$s(T) = \frac{\sum_{J_k \in \mathcal{J}} c_k}{|T|}, r_k \geq t_s, d_k \leq t_f \tag{1}$$

For a given job set $\mathcal{J}$, the *critical interval* is defined as the interval with the highest speed. The algorithms we describe in this paper operate by iteratively identifying and scheduling each critical interval. The critical interval found at iteration i is denoted by $T_i = [t_s(i), t_f(i)]$, which has a speed $s_i = s(T_i)$ and executes a set of jobs $\mathcal{J}_i \in \mathcal{J}$.

2.2 Low-Power Earliest Deadline First (LPEDF)

We briefly review the voltage scheduling algorithm, **LPEDF**, presented in [14], as it is referenced throughout the paper. The algorithm is rephrased in Algorithm 1.

Algorithm 1 $[\mathcal{T}]$ **= LPEDF** $(\mathcal{J})$

```
1:  Input: The job set J.
2:  Output: A valid voltage Schedule T.
3:  i := 0 /* the critical interval index */
4:  while J is not empty do
5:     Find the next critical interval T_i = [t_s(i), t_f(i)];
6:     Insert T_i into T;
7:     Remove all jobs J_i from J;
8:     for every release time or deadline, t_sch, where t_sch > t_s(i) do
9:        if t_sch ≤ t_f(i) then
10:          t_sch := t_s(i);
11:       else
12:          t_sch := t_sch − |T_i|;
13:    i++;
14: Return T;
```

LPEDF finds off-line an optimal voltage schedule for a set of independent tasks executed according to the EDF priority. It assumes an ideal DVS processor without transition overhead. The general idea is to iteratively identify (line 5), schedule (lines 6 and 7) and remove (lines 8-14) each critical interval. Lines 8-14 essentially "squeeze" the critical interval to a single time point at $t_s(i)$ by reducing all release times or

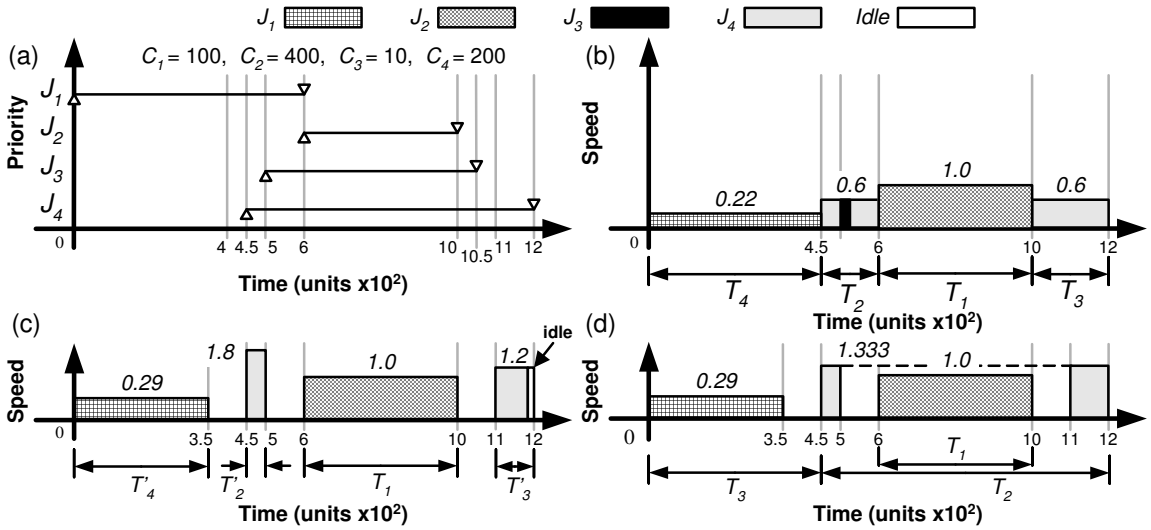

Figure 1: (a) An example set of jobs. (b) Optimal voltage schedule with LPEDF. (c) The LPEDF schedule is modified by inserting $\Delta t = 100$ at each transition, and then rescaling T_2, T_3 and T_4 to maintain the same executed workload. (d) The schedule produced by M-LPEDF.

deadlines inside T_i to $t_s(i)$ and then reducing all release times or deadlines after $t_f(i)$ by $|T_i|$. For the rest of the paper when we refer to squeezing or compressing an interval, we mean performing a similar operation.

Algorithm 1 is greedy in the sense that it always picks the critical interval to schedule first. Consequently, the intervals are identified according to the monotonically non-increasing order of their associated speeds. Due to the fact that the power function is convex, this monotonicity property, summarized formally in Lemma 1, is desired in constructing voltage schedules. For the proof of Lemma 1 and more details on LPEDF, we direct the readers to [14].

Lemma 1 *Critical intervals found by successive iterations of LPEDF are monotonically non-increasing in intensity, that is, $s_i \geq s_j$ if $i \leq j$.*

2.3 Motivational Example

To see the possible impacts of ignoring transition overhead, let's look at the following example. Consider the task set in Figure 1(a), which contains four jobs (where $\triangle$ represents a job release time and $\triangledown$ a job deadline). The optimal voltage schedule by LPEDF, assuming both Δt and ΔE are zero, is given in Figure 1(b). Suppose the same set of jobs is scheduled on a variable voltage processor with $\Delta t = 100$ (Note that compared with the transition interval, the job active durations, i.e., $d_i - r_i$, are relative small for the example job set. This is to make the illustration easier. For real applications, job active durations can be either comparable or much bigger than the transition interval).

A straightforward approach to include time overhead is to (i) insert a transition interval at each speed change and (ii) rescale the speed of the interval with the lower speed to ensure that the same workload is accomplished. (Increasing the speed of the interval with the higher speed would lead to less energy saving

due to the convexity property of the power function.) The new schedule is shown in Figure 1(c). The speed of T_2', the modified interval T_2, was calculated using equation 2.

$$s_i' = \frac{s_i(t_f(i) - t_s(i))}{t_f'(i) - t_s'(i)} = \frac{0.6(6 - 4.5)}{5 - 4.5} = 1.8 \tag{2}$$

There are several problems with the schedule in Figure 1(c). First, s_2' and s_3' are now higher than s_1, i.e., the speed of a critical interval is higher than that of a critical interval identified in a previous iteration of LPEDF. We refer to this as a **monotonicity violation** because it violates Lemma 1. Note that s_2' and s_3' also surpasses the normalized maximum of 1, referred to as **overshoot**, so the required speed is unachievable.

Second, J_3 will never be executed. Notice that $r_3 = 500$ and $d_3 = 1050$, but cycles cannot execute from [500, 600] or [1000, 1100] and the interval [600, 1000] is completely utilized by J_2. We refer to this problem, where a job is not executed because it is contained in transition intervals, as an **execution violation**.

Finally, notice that there is a sliver of idle time in the interval [1192, 1200]. When T_2 was modified to T_2' to accomdate the transition interval, it still included the execution cycles of J_3, even though T_2' is outside $[r_3,d_3]$. The extra time was used by J_4 during T_2', so the idle part of T_3' is not required. Because equation 2 fails to take into account release times and deadlines when rescaling the speed, it can result in schedules that require jobs to execute before they are released or after their deadlines.

A bit more sophisticated modification to LPEDF can be done as follows: Instead of compressing just the critical interval $T_i = [t_s(i), t_f(i)]$ down to a single time point, compress the interval $[t_s(i) - \Delta t, t_f(i) + \Delta t]$ and adjust adjacent jobs accordingly. We refer to this approach as **M-LPEDF**. Applying M-LPEDF to the system in Figure 1(a), we obtain a new voltage schedule shown in Figure 1(d). Unfortunately, the schedule in Figure 1(d) still has monotonicity and execution violations since the speed of T_2 is higher than that of T_1 and J_3 is not executed as it is contained in the transition interval.

From the above discussions, it is clear that the problem of accounting for the time overhead is not just a simple process of locally adjusting the optimal LPEDF solution. Care must be taken during the scheduling process to ensure that the resulting schedule is valid. Considering energy overhead and discrete levels adds more challenges to the problem. In the following, we propose our approaches to solve this problem. To simplify our discussion, we will first assume that ΔE is negligible and that the processor's voltage can vary continuously. We will remove these assumptions later.

3 A Basic Algorithm for Time Transition Overhead

As explained before, the monotonicity and execution violations in the naive extension of LPEDF, i.e., M-LPEDF, will incur the overshooting processor speed, which will lead essentially to infeasible and/or energy inefficient design. Our goal in this section is to derive a better algorithm that can eliminate these violations and provide a feasible and reasonably energy efficient voltage schedule in the presence of voltage transition overhead.

To handle monotonicity violations, we have observed that any critical interval that violates the monotonicity must be adjacent to the critical interval identified in the previous iteration. (Note that generally critical intervals found in subsequent iterations are not necessarily adjacent to one another). This observation is stated formally in Lemma 2.

Lemma 2 *Let T_{i-1} and T_i be two critical intervals obtained by M-LPEDF. If $s(T_{i-1}) < s(T_i)$, then T_{i-1} and T_i are adjacent.*

Proof: To schedule $T_{i-1} = [t_s(i-1), t_f(i-1)]$, M-LPEDF compresses $T'_{i-1} = [t_s(i-1) - \Delta t, t_f(i-1) + \Delta t]$. This modification does not change the workload distribution of jobs outside T'_{i-1}. Only intervals adjacent to or overlapping T'_{i-1} are altered; shortened by up to Δt time or by exactly $2\Delta t$ time if T_i contains T'_{i-1}. Thus, only intervals adjacent to T_{i-1} may experience an increase in intensity in the next iteration. □

As implied in the proof for Lemma 2, monotonicity violation occurs when the execution space for some jobs are shortened due to the transition overhead such that they have to require an even higher speed in order to meet their deadlines. To remove these violations, we can remove the transitions in order to save the transition overhead that cause the violation. Specifically, such violation can be eliminated by incorporating these jobs into those in the previously found critical interval. We formulate this conclusion in Lemma 3.

Lemma 3 *Let T_{i-1} and T_i be two critical intervals obtained by M-LPEDF with $s(T_{i-1}) < s(T_i)$. The minimum speed at which* every *job $J_k \in (\mathcal{J}_{i-1} \cup \mathcal{J}_i)$ can execute and still meet its deadline is $s(T_{i-1})$.*

Proof: According to Lemma 2, T_{i-1} and T_i must be adjacent. Therefore, if a single speed, i.e., $s(T_{i-1})$, is applied to both these intervals, no voltage transition is necessary. According to Lemma 1, $s(T_{i-1})$ is the minimum speed required to guarantee the deadlines for the jobs in $\mathcal{J}_{i-1}$ and greater than that to guarantee the deadlines for jobs in $\mathcal{J}_i$ when no time overhead is presented. Therefore, $s(T_{i-1})$ is the minimum speed for every jobs in $(\mathcal{J}_{i-1} \cup \mathcal{J}_i)$ to meet their deadlines. □

Lemmas 2 and 3 help us to keep track of the execution violation and remove it whenever it occurs. In fact, they also help us to handle the execution violation as well since we can treat the an execution violation as a special case of the monotonicity violation, i.e., the one with the required speed as infinity since we are supposed complete a finite number of cycles in zero time.

Therefore, with Lemmas 2 and 3, we propose a more sophisticated algorithm, called Time Overhead Earliest Deadline First (TOEDF), that can deal with both the monotonicity and execution violations and produce a feasible voltage schedule under the voltage transition overhead. Algorithm 2 shows the salient aspects of TOEDF.

In TOEDF, when a monotonicity or execution violation is encountered (line 6), the previous critical interval is extended to include all jobs in the current interval (lines 7–10). Since "squeezing" a critical interval will change the timing parameters of some jobs (line 16), which are needed if a critical interval causes a violation (line 7), we need to save these parameters (lines 13) until needed. After the critical interval is identified, we may need to extend of this interval as that in M-LPEDF to tolerate the transition

Algorithm 2 $[\mathcal{T}]$ = **TOEDF** ($\mathcal{J}$, Δt)

1: **Input:** The job set $\mathcal{J}$, and time transition overhead length, Δt.
2: **Output:** A valid voltage Schedule $\mathcal{T}$.
3: $i := 1$; /* the critical interval index */
4: **while** $\mathcal{J}$ is not empty **do**
5: Find the next critical interval $T_i = [t_s(i), t_f(i)]$;
6: **if** ($i > 1$ AND $s(T_i) > s(T_{(i-1)})$) /* Monotonicity or execution violation */ **then**
7: Restore the timing information from the previous iteration;
8: Remove $T_{(i-1)}$ from $\mathcal{T}$;
9: Merge the interval T_i with $T_{(i-1)}$;
10: $i--$; /* Roll back the critical interval index */
11: Adjust the end points of interval T_i to accommodate the transition overhead Δt;
12: Backup the timing information;
13: Remove all jobs in T_i from $\mathcal{J}$;
14: Insert T_i to $\mathcal{T}$;
15: Squeeze T_i into a single time point;
16: i++;
17: **RETURN** $\mathcal{T}$;

overhead Δt (line 12). One has to be particularly careful in doing this. For example, if an ending point of the critical interval is overlapped with that of any previous intervals in $\mathcal{T}$, we should not extend the critical interval at this end to avoid accounting for Δt more than once. In what follows, we apply Algorithm 2 on the job set in Figure 1(a) and show more details of this algorithm.

When applying Algorithm 2 for the job set in Figure 1(a), in the first iteration of Algorithm 2, the critical interval [600,1000] with the intensity of 1 is identified. It is extended to be [500,1100] (with $\Delta t = 100$) and then inserted to $\mathcal{T}$. During the second iteration, however, an execution violation happens for job J_3. Therefore, J_3 is merged to the previously identified interval [500,1100], which inherits the intensity, i.e., 1, and is then adjusted to deal with the transition overhead. The result interval [400, 1150] is then re-inserted to $\mathcal{T}$. In the third iteration, a monotonicity violation is found related to job J_4. Therefore, we have to modify the interval [400,1150] again and replace it with the new interval [350,1300]. Finally, during the fourth iteration, another critical interval is identified as [0,350] with intensity $100/350 = 0.29$. Note that the right ending point of this interval 350 is at the same position as one the ending points of the interval in $\mathcal{T}$. Therefore, there is no need to extend the interval at this end, and the interval [0,350] is inserted into $\mathcal{T}$ (we assume no transition overhead at time 0.) One can readily verify that the result voltage schedule does guarantee the schedulability of all the jobs.

To formally prove the correctness and explore the complexity of Algorithm 2, we have the following theorem (Theorem 1).

Theorem 1 *Algorithm 2 will always produce a valid voltage schedule in $O(n^3)$ time, given an initially schedulable job set.*

Proof: Executing jobs $\mathcal{J}_i \in T_i$ at $s(T_i)$ ensures that all deadlines within T_i are met. Monotonicity and execution violations are identified and eliminated with the jobs deadlines are guaranteed to be satisfied

according to Lemmas 2 and 3. The dominating step per iteration is identifying the critical interval, which is $O(n^2)$. The outer loop can repeat up to $O(n)$ times. □

4 A Unified Algorithm for the Non-Ideal Properties of DVS Processors

In the previous section, we present an algorithm, i.e., TOEDF, to produce a feasible and reasonably energy efficient voltage schedule under the transition overhead condition. In this section, we present an enhanced algorithm to improve the energy efficiency of TOEDF and incorporate discrete speeds and energy transition overhead simultaneously.

4.1 Improving the Basic Algorithm

Unnecessary energy may be wasted when using TOEDF since, in order to eliminate the violations, we have to use some "higher-than-necessary" voltage in some intervals. Whereas removing the violations is critical for guaranteeing the feasibility of the voltage schedule, a shorter interval that demands the high processor speed can help to reduce the energy consumption. For example, as explained in last section, when applying TOEDF to the example in Figure 1(a), it is required that the processor speed for the interval [450,1200] be 1.0. However, one can readily verify that using the processor speed of 1. 0 during the interval [590,120] can also guarantee the schedulability of the jobs, i.e., J_2,J_3, and J_4. The energy is saved in two ways: first, the interval length that demands high processor speed is reduced; second, extra space is available for the remaining jobs, such as J_1 which can be used to reduce their required processor speed. Therefore, if we have to use the high voltage, we can manage to save the energy by restricting the usage of the high voltage in an interval as short as possible. We formalize this problem as follows.

Problem 1 *Given a set of jobs, $\mathcal{J}$, and a constant speed, s^* (unless notified otherwise, s^* is a constant speed higher than the minimum one required to meet all deadlines of $\mathcal{J}$), find the shortest interval in which all deadlines of $\mathcal{J}$ are met.*

The key to solving Problem 1 is to realize that we really only have one degree of freedom in the problem; how long we can delay the start of the interval, i.e. delay the use of the higher speed. Of course, by delaying the interval we run the risk of missing job deadlines. To prevent any deadlines from being missed, we next introduce the concept of the *latest start time* for a job set and an important lemma stating how to compute it.

Definition 1 *The* Latest Start Time t_{LS}, for a job set $\mathcal{J}$, *is the latest time at which jobs in $\mathcal{J}$ can begin execution at the speed s^* and still meet all deadlines.*

Lemma 4 *The job set $\mathcal{J}$ scheduled by **EDF** has the latest start time t_{LS} as*

$$t_{LS} = \min\{t_{LS}(i) | t_{LS}(i) = d_i - \sum_{J_k \in hp(J_i)} \frac{c_k}{s^*}, i = 1..|\mathcal{J}|\},$$

where $hp(J_i)$ is the job set containing jobs with priorities equal to or higher than that of J_i.

Proof: First we will prove that starting at a time later than $t_{LS}(i)$ will cause J_i to miss its deadline. Then we will prove that J_i will not miss its deadline if we start execution at or before $t_{LS}(i)$.

(1) By EDF, $hp(J_i)$ includes the jobs with deadlines no later than d_i. To guarantee that all the jobs in $hp(J_i)$ can meet their deadlines, there must be a sufficient amount of time to execute the cycles of these jobs. The total workload of these jobs is $W = \sum_{J_k \in hp(J_i)} c_k$, and the total time needed to execute this workload is W/s^*. It is trivial to see that starting later than $t_{LS}(i)$ will not give enough time to finish W. The minimum $t_{LS}(i)$ is the most restrictive latest start time of all jobs in $\mathcal{J}$, so it is the correct choice for t_{LS}.

(2) Suppose that beginning execution at $t_{LS}(i)$ causes J_i to miss its deadline. Considering the execution of the jobs in $hp(J_i)$, there must be idle time in the interval $[t_{LS}(i), d_i]$ that no job in $hp(J_i)$ can be executed. Note that executing these jobs at any time earlier than $t_{LS}(i)$ will not alter the job release time, so J_i must still miss its deadline. This contradicts the assumption in Problem 1 that all jobs in $\mathcal{J}$ executing at the speed s^* can be finished by their deadlines. □

Lemma 4 helps us to find the latest start time jobs for $\mathcal{J}$ in Problem 1. Then, with a simple plane-sweeping algorithm we can find the last finish time. With these observations, we construct an algorithm, i.e., MININT (Algorithm 3), to find the minimal required interval.

Algorithm 3 $[T]$ **= MININT(**$\mathcal{J}$**,** s^***)**

1: **Input:** The set of jobs to include in the minimum interval, $\mathcal{J}$, sorted from highest to lowest priority, and the speed at which the jobs must execute, s^*.
2: **Output:** The minimum interval $T = [t_s, t_f]$.
3: $t_s := \infty$;
4: $Sum := 0$;
5: **for** $(i := 1; i < |\mathcal{J}|; i++)$ **do**
6: $\quad Sum := Sum + \frac{c_i}{s^*}$;
7: $\quad t_{LS}(i) := d_i - Sum$;
8: $\quad t_s := min\{t_s, t_{LS}(i)\}$;
9: $t_f :=$ Identify_Finish_Given_Start$(\mathcal{J}, t_s, s^*)$;
10: $T := [t_s, t_f]$;
11: **RETURN** T;

To show that Algorithm 3 indeed produces the minimum-length valid interval, we have the following theorem.

Theorem 2 *Given a set of jobs, $\mathcal{J}$, and a constant speed, s^*, Algorithm MININT finds the minimum length interval T needed to complete all $J_k \in \mathcal{J}$ at the speed s^* by their deadlines.*

Proof: We consider the following two cases based on $|T|$:

(1) $|T| = \sum_{i=1}^{|\mathcal{J}|} \frac{c_i}{s^*}$

Based on the arguments used in proving Lemma 4, it follows that $|T|$ in this case is the shortest interval needed in order to complete all jobs in $\mathcal{J}$.

(2) $|T| > \sum_{i=1}^{|\mathcal{J}|} \frac{c_i}{s^*}$

In this case there must be *idle time* within T. Without loss of generality, suppose there is one idle interval, $[t_c, t_r]$, where t_c is the time when the previous job completes execution, and t_r is the time when the next job is released. Note that moving t_s back in time will only increase $|[t_c, t_r]|$ or produced new idle intervals inside T, causing $|T|$ to increase. Lemma 4 states that t_s must be less than or equal to t_{LS} or at least one job will miss its deadline. Because t_r is fixed and all jobs in T execute at the constant speed s^*, the finish time of the final job, t_f, is also fixed. Expanding T beyond this point will introduce idle time, and moving it back will cause the final job to miss its deadline. Therefore, moving t_s or t_f to any other point cannot decrease $|T|$ without causing a deadline miss, and the proof is complete. □

Algorithm MININT can be readily incorporated into TOEDF and improve its energy performance. In TOEDF, when a violation happens and jobs are merged to the previously found interval, we can apply MININT to find the minimal interval within which all the deadlines of the jobs can be satisfied, and more energy can be saved. Moreover, as explained later, Algorithm MININT can also help to improve the energy efficiency when dealing with the discrete voltage levels and energy transition overhead.

4.2 Discrete Voltage/Frequency Levels

Up to now, to simplify our discussion, we assume that the processor speed can be continuously varied. However, current commercial variable voltage processors [2, 3, 1] only have a finite number of speeds. This factor must be integrated into voltage scheduling algorithms to provide a practical, valid, and energy efficient voltage schedule.

One intuitive way to deal with discrete speeds is to round up the required frequency and voltage to some allowed level. Unfortunately, this can be extremely pessimistic and energy inefficient, especially for many commercial processors with only a few voltage levels available [2]. Another better approach is proposed in [29] that can use the two levels immediate above and below the desired voltage/speed value to optimally schedule a single job. Kwon et. al. [20] built upon the results in [29] and develop an optimal voltage/frequency scheduling algorithm for an entire job set, called AllocVT. Although theoretically optimal, AllocVT is not practically applicable on real processors because excessive voltage transitions are acquired, i.e., roughly twice per job according to this algorithm, and the ignorance of the transition overhead will cause the jobs to miss their deadlines.

We believe that it is more advantageous to incorporate the discrete speed effects into the construction of critical intervals and let it propagate to future critical interval construction. Specifically, after a critical interval is identified in Algorithm 2, its speed is increased to the next available level. Recall that when a

higher-than-necessary voltage is applied, we can use MININT to find the minimal necessary interval with the given voltage.

Note that, when increasing the voltage to the next higher level, it can introduce a significant amount of unused idle time even after we apply algorithm MININT to find the minimal necessary interval. A better method to utilize these idle times is to relax the requirement that all jobs originally found in the critical interval must run at the higher speed. Therefore, we keep only one of these busy intervals (intervals without idle time) for the final voltage schedule, with the expectation that the rest of the jobs may benefit from the higher-than-necessary speed assignment for this interval and can execute at a lower speed. We summarize this approach in Algorithm 4.

Algorithm 4 $[T_i', \mathcal{J}_i', s'(T_i)]$ **= DISCRETE** $(T_i, \mathcal{J}_i, s(T_i), \mathcal{S})$

1: **Input:** The initial critical interval T_i, its speed $s(T_i)$, the set of jobs $\mathcal{J}_i$ in T_i, and the set of valid speeds $\mathcal{S}$.
2: **Output:** A valid critical interval T_i', running at $s'(T_i)$ that executes the jobs $\mathcal{J}_i'$.
3: $s'(T_i) := \min\{S_j \in \mathcal{S} | S_j \geq s(T_i)\}$;
4: $T :=$ MININT $(\mathcal{J}_i, s'(T_i))$;
5: Identify one busy interval, $B(i) = [t_s, t_f'] \in T$;
6: $T_i' := B(i)$;
7: $\mathcal{J}_i' :=$ all jobs $J_k \in \mathcal{J}_i$ that finish execution in $B(i)$;
8: **RETURN** T_i', $\mathcal{J}_i'$, and $s'(T_i)$;

In Algorithm 4, one problem is to select a busy interval to keep (line 5). A good choice can lead to low computation cost and higher energy efficiency. There are a number of heuristics, such as always selecting (a) the first, (b) the last, (c) the shortest, or (d) the longest busy interval. Though each of these approaches has its intuitive advantages, none of them dominates the others in our experiments, due to the many patterns of job arrival times, deadlines, and execution cycles. Therefore, we simply choose the first busy interval because it is the most computationally convenient.

4.3 Energy Transition Overhead

So far, we have ignored energy transition overhead in our voltage scheduling algorithms. To deal with the energy transition overhead, like timing overhead, we feel it is valuable to account for this overhead while constructing the critical intervals, thus allowing the effect to propagate throughout the schedule. Our approach works as follows: when a new critical interval is identified, whether this critical interval should be kept depends on whether or not the energy consumed by adopting its speed is smaller than that consumed by merging it with an adjacent critical interval.

If the energy transition overhead is so significant that using different processor speeds for different intervals may not be beneficial at all, we can simply merge these intervals and remove the voltage transitions. Given a critical interval $T_i = [t_s(i), t_f(i)]$, its adjacent critical interval, $T_j = [t_s(j), t_f(j)]$, can be in any one of the following forms: (a) $t_f(j) < t_s(i)$, (ii) $t_f(i) < t_s(j)$, or (c) $t_s(i) < t_s(j)$, $t_f(j) < t_f(i)$. To merge T_i with T_j while guaranteeing the schedulability of the jobs in these intervals, we have to adopt the higher processor

speed for these interval. Now we are in the same situation that a higher-than-necessary voltage is applied in an interval. We can then use algorithm MININT to find the minimal-length interval to better save the energy.

4.4 The Unified Algorithm

By combining the techniques from sections 4.1 through 4.3 with Algorithm 2, a valid voltage schedule with superior energy savings can be achieved while accounting for practical limitations of real-world DVS processors, including time and energy transition overhead and discrete voltage levels. We call this unified algorithm UAEDF (Algorithm 5).

Algorithm 5 $[\mathcal{T}]$ = **UAEDF** ($\mathcal{J}$, Δt, $\mathcal{S}$)

1: **Input:** The job set $\mathcal{J}$, the time transition overhead length, Δt, and the set of allowable speeds, $\mathcal{S}$.
2: **Output:** A valid voltage schedule $\mathcal{T}$.
3: $i := 1$; /* The critical interval index. */
4: $i_{prev} := 1$; /* The index of the previously inserted critical interval */
5: **while** $\mathcal{J}$ is not empty **do**
6: Find the next critical interval $T_i = [t_s(i), t_f(i)]$;
7: $[T_i, \mathcal{J}_i, s(T_i)]$:= DISCRETE(T_i, $\mathcal{J}_i$, $s(T_i)$, $\mathcal{S}$)
8: **if** ($i > 1$ AND $s(T_i) > s(T_{(i-1)})$) **then**
9: Remove $T_{i_{prev}}$ from $\mathcal{T}$;
10: $\mathcal{J}_{i_{prev}} := \mathcal{J}_{i_{prev}} \cup \mathcal{J}_i$
11: $T_{i_{prev}}$:= MININT($\mathcal{J}_{i_{prev}}$, $s(T_{i_{prev}})$);
12: $i--$;
13: **else**
14: $i_{prev} := i$;
15: Adjust the end points of interval T_i to accommodate the transition overhead Δt;
16: $\mathcal{T}_{adj}$:= the set of all intervals adjacent to T_i;
17: Find $T_k \in \mathcal{T}_{adj}$ such that merging T_i with T_k has the maximal gain in energy saving;
18: **if** the maximal gain > 0 **then**
19: $\mathcal{J}_i := \mathcal{J}_i \cup \mathcal{J}_k$;
20: T_i := MININT($\mathcal{J}_i$, $s(T_k)$);
21: Adjust the end points of interval T_i to accommodate the transition overhead Δt;
22: Replace T_k in $\mathcal{T}$ with T_i;
23: **else**
24: Insert T_i to $\mathcal{T}$;
25: Squeeze T_i into a single time point;
26: i++;
27: **RETURN** $\mathcal{T}$;

Algorithm 5 follows the same general flow as Algorithm 2. First it identifies the next critical interval under the discrete voltage level assumption (line 6-7). Then it removes the possible monotonicity or execution violations and shorten the critical interval with algorithm MININT (line 9-11). A greedy approach is adopted in Algorithm 5 to deal with the energy overhead during the voltage transition (line 17-26). After a critical interval is found, we check to ensure whether or not it is more advantageous to merge it with an adjacent interval (line 17-18). If such a merge is necessary (line 19), we simply merge it with one of its adjacent intervals that will lead to the maximal energy saving gain, and then apply MININT to minimize the interval

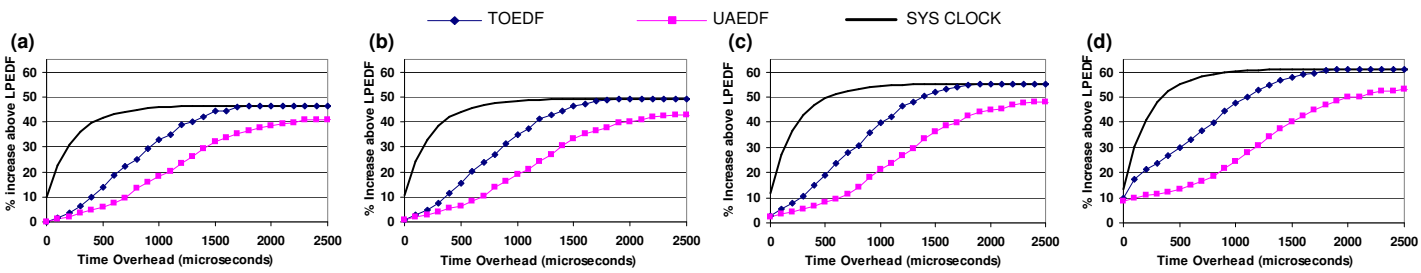

Figure 2: Increase in the energy consumption above LPEDF of randomly generated jobs on a simulated AMD Mobile Athlon4 processor. (a) Results from a continuous voltage supply, (b) 14 voltage levels (c) 5 voltage levels and (d) 2 voltage levels.

length (line 20-23). Finally, as that in TOEDF, the newly generated interval is "squeezed" into one single time point and one iteration of the algorithm is completed. Theorem 3 formally states that Algorithm 5 finds a valid schedule in $O(n^3)$ time.

Theorem 3 *Algorithm 5 always produces a valid voltage schedule with a time complexity of* $O(n^3)$.

Proof: Theorem 1 states that TOEDF always produces a valid voltage schedule. Algorithm 5 includes time overhead and handles the same violations as TOEDF, but uses MININT instead of simply matching release times and deadlines. It follows from Theorem 2 that this method always produces a valid schedule. Energy overhead is also accounted for using MININT, so Theorem 2 ensures the validity of this step as well. Discrete voltage levels are matched by rounding up to a valid level and taking the first busy interval. The jobs in the busy interval will obviously meet their deadlines, and the rest of the jobs will be rescheduled by methods we have just shown to be valid.

The most time consuming step per iteration is identifying the next critical interval in $O(n^2)$ time, which can repeat $O(n)$ times. Therefore, the overall complexity is $O(n^3)$. □

5 Experimental Results

In this section, we quantify the energy consumption due to the transition overhead and discrete voltage levels and evaluate the energy savings of our proposed algorithms with both the randomly generated job sets and real-world examples.

We first constructed and tested 100 randomly generated sets of 20 jobs each with our algorithms. The jobs are assumed to have worst-case execution times and release times uniformly distributed between [0,800] and [0,1000] μS, respectively. The relative deadlines of the jobs are normally distributed with an average of 810 μS and a standard deviation of 280μS. For each job set, we applied both Algorithm 2 and Algorithm 5 with the overhead ranging from 0 (no overhead) to 1 mS.

We also apply our algorithms to two real-world examples, i.e. CNC [30] and Avionics [31].

6 Summary

In this paper, we studied the impact that practical limitations of current processors can have on existing voltage schedules. We have shown through examples and analysis that limitations such as transition overhead or discrete voltage levels can cause a theoretically optimal schedule to become invalid if not correctly accounted for during the scheduling process. Accounting for such limitations is not a trivial matter, as trying to make adjustments to the optimal schedule by inserting overhead between voltage intervals will likely cause jobs to miss their deadlines. Our experiments show that for practical processors and real applications, the largest contributor to energy consumption is discrete voltage levels, the second is time transition overhead, and finally energy transition overhead. Although they vary in importance, they must be considered in unison to ensure a valid schedule is reached. We have presented two algorithms, which guarantee a valid voltage schedule given an initially schedulable job set. The basic algorithm, TOEDF, offers a simple implementation, while the improved algorithm, UAEDF, can significantly reduce energy consumption when compared to TOEDF.

Currently, the optimality of our algorithms is not guaranteed, so further algorithm development may improve results even more. Also, for the scheduling process to give a practical voltage schedule for an even wider range of systems, we will need to account for other implementation details, such as context switching overhead, support for different transition models and support for other prioritization schemes, such as fixed priority scheduling. Future work must account for these limitations.

References

[1] Intel, "The intel xscale microarchitecture," Intel Corporation, Tech. Rep., 2000.

[2] AMD, "Mobile amd athlon 4 processor model 6 cpga data sheet rev:e," Advanced Micro Devices, Tech. Rep. 24319, Nov. 2001.

[3] M. Fleischmann, "Longrun power management: Dynamic power management for crusoe processors," Advanced Micro Devices, Tech. Rep., Jan. 2001.

[4] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," in *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISPLED)*. New York: IEEE, July 2000, pp. 9–14.

[5] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *Proceedings of the 7th annual international conference on Mobile computing and networking (MOBICOM)*. New York: ACM Press, July 2001, pp. 251–259.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[7] J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2000.

[8] W. Kim, J. Kim, and S. L. Min, "Dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *Design, Automation and Test in Euroope Conference and Exhibition (DATE)*. New York: IEEE, Mar. 2002, pp. 788–794.

[9] T. Okuma, T. Ishihara, and H. Yasuura, "Software energy reduction techniques for variable-voltage processors," *Design & Test of Computers*, vol. 18, no. 2, pp. 31–41, March – April 2001.

[10] ——, "Real-time task scheduling for a variable voltage processor," in *Proceedings of teh 12th International Symposium on System Synthesis (ISSS)*. New York: IEEE, Nov. 1999, pp. 24–29.

[11] G. Quan and X. S. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proceedings of the 2001 Design Automation Conference (DAC)*. New York: IEEE, June 2001, pp. 828–833.

[12] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proceedings of the 36th Design Automation Conference (DAC)*. New York: IEEE, June 1999, pp. 21–25.

[13] D. Shin, S. Lee, and J. Kim, "Intra-task voltage scheduling for low-energy hard real-time applications," *Design & Test of Computers*, vol. 18, no. 2, pp. 20–30, March – April 2001.

[14] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science (FOCS)*. New York: IEEE, Oct. 1995, pp. 374–382.

[15] H.-S. Yun and J. Kim, "On energy-optimal voltage scheduling for fixe-priority hard real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 2, no. 3, pp. 393–430, Aug. 2003.

[16] H. Aydin, R. Melhem, D. Moss, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *To appear in IEEE Transactions on Computers*, 2003.

[17] L. H. Chandrasena, P. Chandrasena, and M. Liebelt, "An energy efficient rate selection algorithm for voltage quantized dynamic voltage scaling," in *Proceedings of teh 14th International Symposium on System Synthesis (ISSS)*. New York: IEEE, 30 Sept. - 3 Oct. 2001, pp. 124–129.

[18] S. Lee and T. Sakurai, "Run-time power control scheme using software feedback loop for low-power real-time applications," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference (ASP-DAC)*. New York: IEEE, Jan. 2000, pp. 381–386.

[19] ——, "Run-time voltage hopping for low-power real-time systems," in *Proceedings of the 37th Design Automation Conference (DAC)*. New York: IEEE, June 2000, pp. 806–809.

[20] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," in *Proceedings of the 2003 Design Automation Conference (DAC)*. New York: IEEE, June 2003, pp. 125–130.

[21] S. Saewong and R. Rajkumar, "Practical voltage-scaling for fixed-priority rt systems," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. New York: IEEE, May 2003, pp. 106–114.

[22] A. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design (ISPLED)*. New York: IEEE, Aug. 2001, pp. 9–14.

[23] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors," in *Proceedings of the 19th Real-Time Systems Symposium (RTSS)*. New York: IEEE, Dec. 1998, pp. 178–187.

[24] N. AbouGhazaleh, B. Childers, D. Moss, R. Melhem, and M. Craven, "Energy management for real-time embedded applications with compiler support," in *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems (LCTES)*. New York: ACM Press, June 2003, pp. 284–293.

[25] N. AbouGhazaleh, D. Moss, B. Childers, R. Melhem, and M. Craven, "Collaborative operating system and compiler power management for real-time applications," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. New York: IEEE, May 2003, pp. 133–141.

[26] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation (PLDI)*. New York: ACM Press, June 2003, pp. 38–48.

[27] A. Chandrakasan, S. Sheng, and R. W. Brodersen, *Low-Power Digital CMOS Design*. New York, NY: Kluwer Academic Publishers, 1996.

[28] (2003) Technology roadmap for semiconductors. online. International SEMATECH. [Online]. Available: http://public.itrs.net/

[29] T. Ishihara and H. Yasurra, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISPLED)*. New York: IEEE, Aug. 1998, pp. 197–202.

[30] N. Kim, M. Ryu, SeongsooHong, M. Saksena, C. ho Choi, and H. Shin, "Visual assessment of a real-time system design: a case study on a cnc controller," in *Proceedings of the 17th Real-Time Systems Symposium (RTSS)*. New York: IEEE, Dec. 1996, pp. 300–310.

[31] C. D. Locke, D. R. Vogel, and T. J. Mesler, "Building a predictable avionics platform in ada: a case study," in *Proceedings of the 12th Real-Time Systems Symposium (RTSS)*. New York: IEEE, Dec. 1991, pp. 181–189.