A Realistic Variable Voltage Scheduling Model for Real-Time Applications *

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

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

ABSTRACT

Voltage scheduling is indispensable for exploiting the benefit of variable voltage processors. Though extensive research has been done in this area, current processor limitations such as transition overhead and voltage level discretization are often considered insignificant and are typically ignored. We show that for hard, real-time applications, disregarding such 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 demands for mobile and pervasive computing devices have made low power computing a critical technology. One of the most effective ways of reducing energy is so called Dynamic Voltage Scaling (DVS), i.e., dynamically varying supply voltage and frequency (or speed) simultaneously. Several variable voltage processors are appearing in the market e.g., Intel's XScale [7], AMD's Mobile Athlon [1], and Transmeta's Crusoe processor [5]. Several research groups have also developed their own variable voltage systems. Burd and Brodersen implemented a variable voltage system using the ARM8 core [2], while Pouwelse, Langendoen and Sips constructed a similar system using the SA-1100 [19]. To effectively exploit the benefit provided by a variable voltage processor, careful selection of voltage levels and frequencies, often referred to as voltage scheduling, is crucial.

While substantial research, e.g., [6, 8, 14, 21, 23, 9, 10, 11, 20, 18, 17, 22] has gone into developing efficient algorithms to utilize this emerging technology, often there is a sizeable gap between the simulated environment and an actual, tangible implementation. A dominating trend is to ignore seemingly insignificant implementation details during the research process to simplify the problem, e.g. [8, 9, 18, 17, 21, 22, 23]. One such detail is voltage transition overhead (or simply transition overhead for brevity), i.e., the time and energy overhead incurred whenever a voltage transition takes place. Another detail is the discrete voltage levels, i.e., some variable voltage processors provide only a limited number of voltage levels. No quantitative analysis has been published regarding the validity of these simplifications. One potential problem is that scheduling algorithms that ignore this transition effect may produce sub-optimal or even completely invalid results, especially when applied to real-time systems, where missed deadlines could significantly degrade performance.

Several researchers do address the timing overhead and/or discrete voltage level issues in their research. For example, in [10] and [11], Lee *et al* propose a time slicing method to dynamically schedule a set of periodic tasks. However, the energy overhead is ignored which can grow rapidly with the slicing granularity. Manzak *et al* [14] address the transition delay by literally increasing the total required execution time or decreasing the processor utilization. Such adjustments will mostly likely lead to either a deadline miss or an over pessimistic design. Chandrasena *et al* [4] introduce a rate selection algorithm for variable voltage processor with limited voltage levels, but it provides no deadline guarantee for the tasks. In [6], Hong *et al* present a heuristic algorithm that includes the transition overhead during static scheduling, but requires both continuous control over the voltage level and that the processor continues executing instructions during the transition, neither of which is guaranteed in actual systems [1, 2, 5, 7, 11, 19].

The problem we are interested in is the off-line scheduling of real-time tasks while accounting for the practical limitations of currently available processors. In this paper, we show that transition overhead can cause deadlines to be missed and, if accounted for in a naïve manner, can be a considerable source of energy consumption above the optimal schedule. Then we present a simple, yet not intuitively obvious method of accounting for transition timing overhead. We further improve on this method to produce a better energy saving performance and to integrate other practical considerations such as transition energy overhead and discrete voltage levels. Through ex-

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

0-7803-7607-2/02/$17.00 ©2002 IEEE

perimental results, we demonstrate quantitatively that the transition overhead and discrete voltage levels can significantly increase the power consumption for a voltage schedule. We also how effectively our proposed algorithms can deal with these factors. To the best of our knowledge, this is the first work that accounts for transition overhead (both time and energy) and discrete voltage levels in the process of deriving voltage schedules for real-time applications.

The remainder of this paper is organized as follows. Section 2 summarizes the relevant background material, including system models, motivation examples, and so forth. Section 3 describes a basic algorithm to deal with transition timing overhead. Section 4 improves upon the basic algorithm in terms of energy saving and accounts for energy overhead and discrete voltage levels, in addition to the timing overhead. Section 5 presents our experimental results, and section 6 concludes the paper.

2. PRELIMINARIES

The real-time system we are interested in consists of a set of jobs $\mathcal{J} = \{J_1 ... J_n\}$, each of which is characterized with release times r_i, deadlines d_i, and worst case execution cycles, c_i. The job set is scheduled with EDF scheme [12]. Throughout this paper, we will often make a reference to the optimal off-line scheduling algorithm (**LPEDF**) for an EDF priority system without transition overhead [23]. The general idea of the algorithm is to iteratively identify the critical interval, i.e., the interval that requires the highest speed in order to finish the jobs within it on time, and then squeeze the critical interval into one single point. Readers can refer to [23] for more details on LPEDF.

For the variable voltage processor used in our system, we assume that its power is a simple convex function of speed, i.e., $P(s) \approx s^3$, and its speed is proportional to the voltage, i.e., $s(V) \approx V$ (we use processor voltage and speed interchangeably because of their one-to-one correspondence.) These assumptions hold when $V >> V_T$(V_T is the threshold voltage) [3], which is usually true in practice. Also, the processor can work at a set of voltage levels, $\mathcal{V} = \{V_1, ..., V_m\}$. For simplicity, the processor voltage is normalized with $V_{max} = 1$. During the scheduling process, a constant *transition interval* (denoted as Δt) and a variable *transition energy overhead*(denoted as ΔE) which may vary depending on the start and ending voltage levels, are associated to each voltage transition.

A seemly true but misleading assumption for the variable voltage processor is that the time transition overhead Δt is proportional to the difference between the starting and ending transition voltage. In several variable voltage systems the Phase-Locked Loop used to set the clock frequency requires a fixed amount of time to lock on a new frequency. This lock time is independent of the source and target frequencies, and is typically larger than the time it takes for the voltage to change [1, 19]. It is also possible that the maximum voltage change rate is asymmetric for rising and falling voltages [19]. A more reasonable assumption is therefore to assume that the transition time overhead as a constant Δt, as we use in this paper. Another common assumption is that instructions can be executed during a voltage/speed transition. As pointed out by Burd and Broderson [2], designing a processor that can execute instructions during transitions can be quite difficult. In fact, for this reason, actual processors/systems, e.g., AMD's

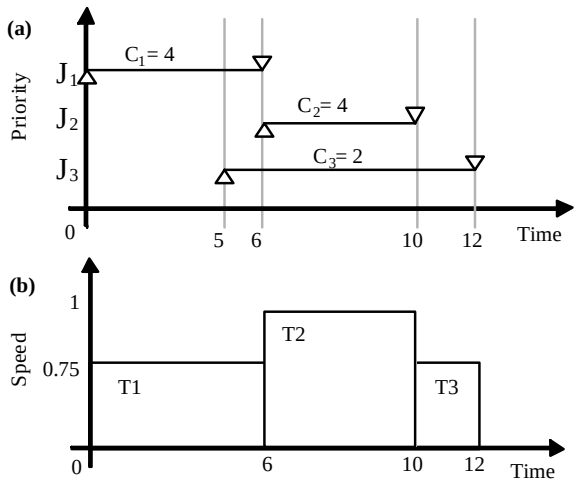

Figure 1: (a) An example set of jobs. (b) Optimal voltage schedule with LPEDF.

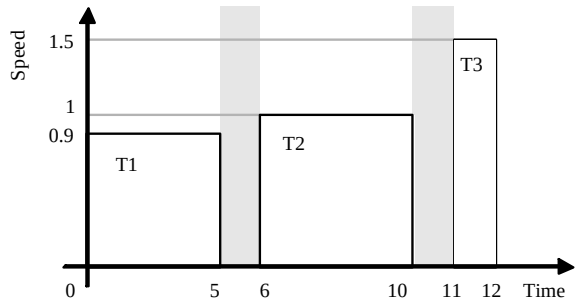

Figure 2: The voltage schedule from Fig. 1(b) modified by inserting transition overhead.

mobile Athlon processor [1] and the SA-1100 based system in [19], often do not allow instructions to be executed during a transition. Simply ignoring this implementation detail can have a significant negative impact on resulting system performance, as we will show later. Hence, in this paper, we assume that no instructions can be executed during a transition interval.

To see the possible impacts of ignoring the transition overhead, let us look at a simple example. Consider the task set in Figure 1(a), which contains three jobs (where $\triangle$ represents a job release time and $\bigtriangledown$ 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 to be scheduled on a variable voltage processor with $\Delta t = 1$. One approach could be to simply insert a transition interval on both sides of $T2$, the interval with the largest speed, and then adjust the speeds of $T1$ and $T3$ accordingly, as shown in Figure 2.

There are several problems with the schedule in Figure 2. First, $S3$'s speed has surpassed the normalized maximum of 1, so the required speed is unachievable. Second, J_3 will miss its deadline even if the speed of 1.5 is possible. Note that the schedule in Figure 1(b) executes a part of J_3 from $t = 5.3$ to 6. On the other hand, the schedule in Figure 2 requires that the same part of J_3 be executed from $t = 4.4$ to 5, which is

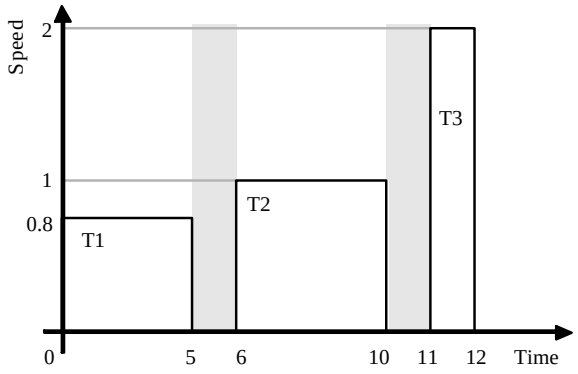

Figure 3: The voltage schedule obtained by simple modification of LPEDF.

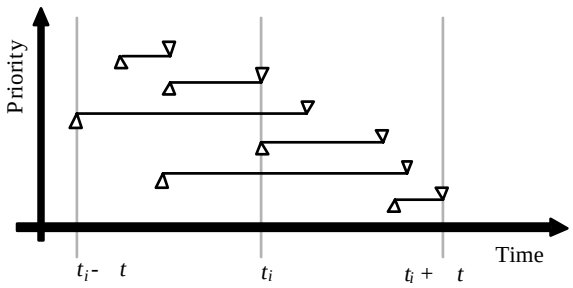

Figure 4: Job arrangements about the interval T_i squeezed down into a single time point ti, which will cause execution violations.

impossible since J_3 is not released until 5. Given that $T2$ is completed utilized by J_2, J_3 can only be executed from $t =$ 11 to 12, which requires the speed in $T3$ be increased to 2!

Obviously, the problem of accounting for transition overhead is not just a simple process of locally adjusting the optimal solution by LPEDF. Care must be taken during the voltage scheduling process to ensure that the generated schedules are valid. In the following, we will propose our approaches to solve this problem. To simplify our discussion, we will first assume that ΔE is negligible and processor can vary continuously. We will remove these assumptions later.

3. A BASIC ALGORITHM

To integrate the voltage transition timing overhead into LPEDF, a reasonable way is to extend the critical interval to accommodate the timing overhead and adjust its speed and the neighboring jobs accordingly such that the effect of fixing the critical interval can be propagated to the construction of future critical intervals. We propose the modification to LPEDF as follows: Instead of compressing just the critical interval, $T = [t_s, t_f]$, down to a single time point, we compress the interval $[t_s - \Delta t, t_f + \Delta t]$ and adjust the job sets accordingly. Applying this modified algorithm to the system in Figure 1(a), we obtain a new voltage schedule shown in Figure 3.

Readers would immediately point out that the schedule in Figure 3 is still not valid since the required voltage is higher than the maximum available. Of course, such voltage overshoots must be eliminated. In general, our simple modification of LPEDF may lead to voltage schedules in which the voltage of a critical interval is higher than the voltage of another critical interval obtained in an earlier iteration. We will refer to this problem as *monotonicity violation*. Monotonicity violation occurs when less time is available to execute the instructions in jobs that overlap a transition interval. To guarantee that all the voltage speeds are less than the maximal one and given that energy consumption increases faster for higher voltage levels, it is desirable to have monotonically non-increasing voltage levels for the subsequently identified critical intervals. To handle the above problem, we have observed that any critical interval that violates the monotonically non-increasing property must be adjacent to the critical interval identified in the immediately previous iteration. Readers can refer to [15] for the detailed illustrations and proofs. This observation helps us in proposing an algorithm which will always produce a voltage schedule with monotonically non-increasing voltage levels.

Furthermore, there is one additional complication with the simple modification introduced at the beginning of this section. Note that the transition interval is on the order of hundreds of microseconds, i.e., thousands of cycles. It is possible the executions of some jobs fall completely inside a transition interval. Moreover, some jobs may have their release times fall in the left transition, and/or their deadlines fall in the right transition. These jobs are not correctly taken care of during the critical interval construction simply because they are not totally contained in interval $[t_s, t_f]$. We will refer to this problem as *execution violation* since the schedule will fail to execute such jobs. Possible arrangements that cause execution violations are illustrated in Figure 4. To solve these problems, we propose a more delicate algorithm as shown in Algorithm 1.

Algorithm 1 LPEDF with transition timing overhead

1: **Input:** The job set $\mathcal{J}$, and time transition overhead length, Δt.
2: **Output:** A valid voltage Schedule $\mathcal{T}$.
3: $i = 0$;
4: **while** $\mathcal{J}$ is not empty **do**
5: i++; //The critical interval index
6: Find the next interval $T_i = [t_s(i), t_f(i)]$ with the maximal speed s_i;
7: **if** ($i > 1$ AND $s_i > s_{(i-1)}$) //monontonicity violation **then**
8: $\mathcal{J} = \mathcal{J}_{backup}$; //Restore related job timing information
9: Adjust $t_s(i)$ and $t_f(i)$;
10: $t_s(i-1) = \min\{t_s(i), t_s(i-1)\}$;
11: $t_f(i-1) = \max\{t_f(i), t_f(i-1)\}$;
12: $i--$;
13: **end if**
14: **while** ($\exists J_i \in \mathcal{J}$ AND $[r_i, d_i] \subseteq [t_s(i) - \Delta t, t_f(i) + \Delta t]$ AND $[r_i, d_i] \nsubseteq [t_s(i), t_f i]$) **do**
15: $t_s(i) = \min\{t_s(i), r_i\}$;
16: $t_f(i) = \max\{t_f(i), d_i\}$;
17: **end while**
18: $\mathcal{J}_{backup} = \mathcal{J}$; //Backup job timing requirements
19: Remove all jobs in $\mathcal{J}$ that fall within $T_i = [t_s(i) - \Delta t, t_f i + \Delta t]$;
20: Squeeze T_i into a single time point and adjust the rest of jobs correspondingly;
21: **end while**

The general idea of Algorithm 1 is: if monotonicity is encountered, the previous critical interval is extended to include all jobs in the current interval, and the resultant interval may need to be extended further to cover all the jobs that would otherwise cause execution violation. Since "squeezing" a critical interval will change the timing parameters of the jobs, which are needed if this critical interval is to be extended for the following one, we use the variable $\mathcal{J}_{backup}$ to backup and restore the job information when necessary. It is worth mentioning that if an end point of a critical interval is the same time point into which some previous critical interval has been squeezed, it would be unnecessary to add another transition interval at that end since the voltage transition overhead has been accounted for by the previous critical interval. In the worst case, the entire schedule will run at the speed of the largest critical interval, which we know will meet all deadlines. The problem when using Algorithm 1 is that unnecessary energy may be wasted. Also, transition energy overhead and discrete voltage level considerations are not yet considered.

4. AN IMPROVED ALGORITHM

In this section, we improve the energy efficiency of Algorithm 1 and incorporate transition energy overhead and discrete voltage levels into considerations.

Unnecessary energy may be wasted when using Algorithm 1. Since energy consumption increases faster at a higher voltage levels, it is desirable that the length of the critical intervals be kept as short as possible, especially during the early execution stage of Algorithm 1. Unfortunately, Algorithm 1 takes a rather greed approach to identify the valid critical interval by simply extending to the earliest release time or the latest deadline (see line 14 to 17) of the jobs. More energy savings can be achieved if we can find a smaller time interval needed to complete the jobs under the given speed/voltage. We formalize the problem as follows.

PROBLEM 1. *Given a set of jobs, $\mathcal{J}$, and a predefined voltage/speed, s^* (s^* is higher than the highest speed needed to complete the jobs), find the shortest interval in which all the jobs can be completed by their deadlines.*

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 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 on how to compute it.

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

LEMMA 1. *The job set $\mathcal{J}$ scheduled by EDF has the latest start time t_{LS} as*

$$t_{LS} = \min\{t_{LS}(i) | t_{LS}(i) = d_i - \sum_{j=1}^{i} \frac{c_j}{s^*}, i = 1..|\mathcal{J}|\},$$

where d_i is job J_i's deadline, s^* is the given speed, c_j is job J_j's worst case execution cycles, and the set $\{J_1, ..., J_i\}$ are the jobs having a greater (or equal) priority than J_i.

The proof of Lemma 1 is omitted due to page limit. After the jobs contained in a critical interval are identified (line 19 in Algorithm 1), Lemma 1 helps us to find the latest start time for these jobs, i.e., t'_s. Then, with a simple simulation of the execution (in linear complexity), we can find the finish time for these jobs, i.e. t'_f. We can prove [15] that $[t'_s, t'_f]$ is the minimum-length valid interval for the execution of jobs removed in line 19 of Algorithm 1.

So far, we have ignored transition energy overhead in our voltage scheduling algorithms. One way to account for transition energy is to include a post-processing step. For example, given the length and speed of current interval, the energy transition overhead, and the expected speeds for both of its neighboring intervals, we can determine if the voltage transitions for current interval will indeed lead to energy saving, or we simply should merge it to one of its neighboring intervals. However, *before* we use the post-processing scheme, we can be more aggressive and take care of the transition energy overhead during the construction of critical intervals, thus allowing the effect to be propagated to the rest of the critical intervals. Our idea is that, when a new critical interval is identified, whether this critical interval should be kept depends on whether the energy consumption by adopting its speed is smaller than that by simply merging it to one of its neighboring critical intervals. The problem, however, is how to *precisely* evaluate the resultant energy saving in each of the cases.

Given a critical interval $T_i = [t_s, t_f]$, its neighboring critical interval $T_j = [t'_s, t'_f]$ can be in any one of the following forms: $(i) t'_f = t_s$, $(ii) t_f = t'_s$, and $(iii) t_s < t'_s, t'_f < t_f$. If there exists one or more neighboring critical intervals for T_i, given the monotonicity property of the critical intervals, we know that the speed of T_i is always lower than its neighboring critical intervals already identified. One way to merge the interval to one of its neighbors is to extend its neighboring interval to cover this interval. Recall that Lemma 1 can help us find the minimal-length interval when a speed higher than necessary is applied for the jobs in the interval. Therefore, we can apply Lemma 1 find the minimal-length interval and hence the minimal energy consumption for executing the jobs in the merged interval. This information can greatly help to prevent the unnecessary voltage transitions and reduce the corresponding energy overhead.

Finally, since current commercial variable voltage processors [1, 7, 5] only have a finite number of voltage levels, this factor must be integrated into voltage scheduling algorithms to provide a practical, valid, and energy efficient voltage schedule. One way to deal with discrete voltage levels is to round up the required voltage to some allowed levels. In [8], the authors proved that the two allowed levels immediately above and below the desired voltage value can be used for this purpose. However, the theorem is only true if the combination of the two allowed voltages can lead to an execution time interval that is the same as the original execution time interval and contains no idle time [24], which may or may not be the case. When considering jobs with both release times and deadlines, to guarantee the validity of the schedule, one can always use another post-processing strategy for the resultant voltage schedule, i.e., rounding up the required voltage to the next higher level. Unfortunately, this can be extremely pessimistic and energy inefficient, especially for many commercial processors with only a few voltage levels available [1].

We believe that, instead of simply rounding up the voltage for the final voltage schedule, it is more efficient to incorporate the discrete voltage level effects into the construction of critical intervals and let its effect be propagated to the construction of future critical intervals. Therefore, after a critical interval is identified in Algorithm 1, its speed is increased to the immediately higher available voltage level. Again, we can use Lemma 1 to find the minimal necessary interval with the given voltage. Note that, while this method would work, it can introduce a significant amount of unused idle time within the critical interval. A better method to better 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 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 be executed at a lower voltage level.

Now the problem becomes how to select the busy interval to be kept. A good choice would directly 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 can really dominate another in our experiments due to the diversity of the patterns of jobs' arrival times, deadlines, and execution cycles. For our results in Section 5, we always choose the first busy interval because it is the most computationally convenient.

By combining the above techniques with Algorithm 1, a valid voltage schedule with superior energy savings can be achieved while accounting for practical limitations of real-world variable-voltage processors, such as transition energy overhead and discrete voltage levels. Due to the page limit, we omit the pseudo code for this algorithm. Reader can refer to [15] for more details.

5. EXPERIMENTAL RESULTS

In this section, we quantify the energy consumption due to the transition overhead and discrete voltage levels and evaluate the energy saving performance 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 time and release time uniformly distributed between [0,800] and [0,1000], respectively. The relative deadlines of the jobs are normally distributed with an average of 810 and a standard deviation of 280. For each job set, we applied both Algorithm 1 and the improved algorithm (Section 4) with the overhead ranging from 0% (no overhead) to 100% of the average deadline of the jobs. A continuous voltage range between [0, 1] was used first to generate the comparison base. This process was repeated for a processor with 5 discrete levels (6 including idle) as with the AMD processor [1], and a processor with 14 discrete levels (15 including idle) as with the SA-1100 system [19]. Since we are still in the process of finding exact energy overhead for real-world processors, our current experiments do not include energy overhead.

The results were then normalized against the optimal voltage schedule without any overhead consideration (using LPEDF).

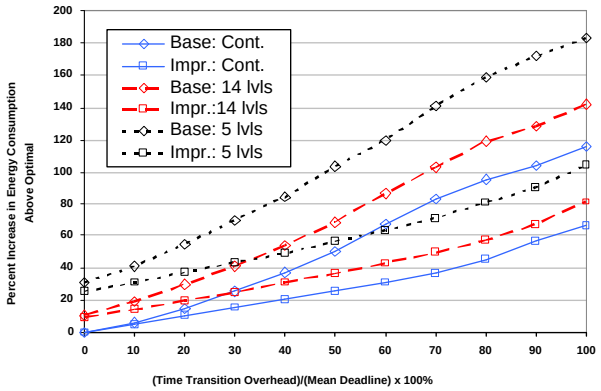

Figure 5: Energy consumption for randomly generated job sets.

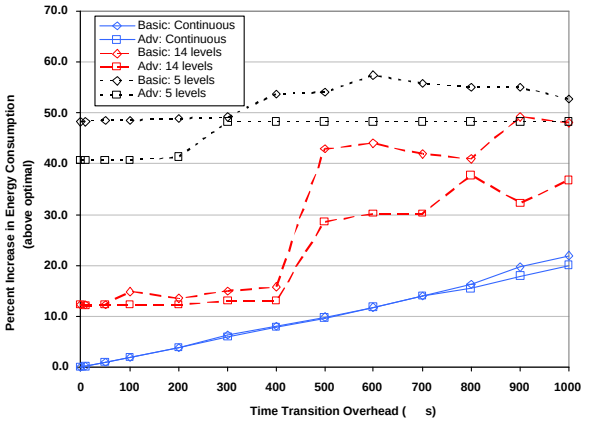

Figure 6: Energy consumption for CNC.

The results are displayed in Figure 5. We also apply our algorithms to two real-world examples, i.e., CNC [16] and Avionics [13]. Our algorithms are tested with the transition timing overhead ranging from $5\mu s$ to $1ms$ for CNC job sets, and $5\mu s$ to $1.5ms$ for Avionics job sets. The results are shown in Figure 6 and 7, respectively.

From Figure 5, 6, and 7, one can immediately notice that the transition overhead (both timing and energy) and discrete voltage levels can cause dramatic increase of energy consumption to the voltage schedule. And, as we expected, the energy consumption grows rapidly with the longer transition overhead and fewer available voltage levels. For example, from Figure 5, a processor model with 5 voltage levels and timing overhead as 5% of the average deadline will increase the total energy consumption by 24% compared with the ideal processor model. Therefore, a valid and energy efficient voltage schedule scheme must take all these factors into consideration. Also, it is not difficult to see that our improved algorithm (Section 4) has a significantly better performance than the basic one, 1.e. Algorithm 1. For example, in Figure 5, for the processor with 5 available voltage levels, our improved algorithm can save nearly 50% of the energy compared with Algorithm 1 when the transition timing overhead is around 50% of the job dead-

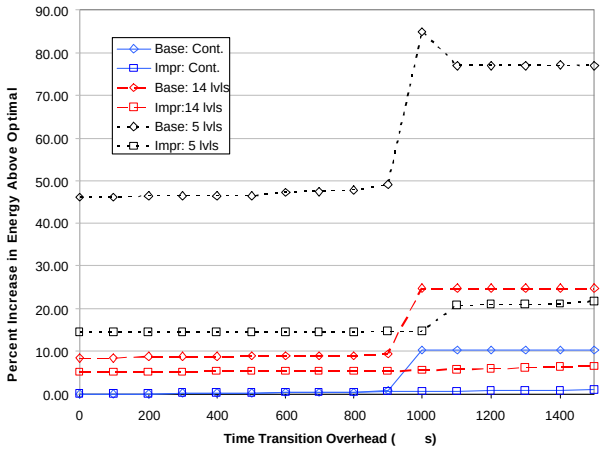

Figure 7: Energy consumption for Avionics.

lines. And the maximal energy saving can be as high as 17% for CNC example (Figure 6) and 82.5% for Avionics example (Figure 7).

6. SUMMARY

In this paper, we studied the impact of practical limitations of current processors on 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. We have presented two algorithms, which are guaranteed to yield a valid voltage schedule given an initially schedulable job set. The base algorithm offers a simple implementation, while the improved one can give significant energy savings over the base algorithm.

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, other implementation details will need to be accounted for, including support for different transition models and other prioritization schemes, such as fixed priority scheduling. Future work must account for these limitations.

7. REFERENCES

[1] I. Advanced Micro Devices. Amd power now technology. 2000.

[2] T. D. Burd and R. W. Brodersen. Design issues for dynamic voltage scaling. *ISSCC*, 2000.

[3] A. Chandrakasan, S.S., and R. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, Apr 1992.

[4] L. Chandrasena, P. Chandrasena, and M. Liebelt. An energy efficient rate selection algorithm for voltage quantized dynamic voltage scaling. *ISSS*, pages 124–129, 2001.

[5] M. Fleischmann. Longrun power management: Dynamic power management for crusoe processors. *RTSS*, 1998.

[6] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. *RTSS*, pages 178–187, 1998.

[7] Intel. The intel xscale microarchitecture. *Technical Summary*, 2000.

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

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

[10] S. Lee and T. Sakurai. Run-time power control scheme using software feedback loop for low-power real-time applications. *ASPDAC*, 2000.

[11] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. *DAC*, 2000.

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

[13] C. D. Locke, D. R. Vogel, and T. J. Mesler. Building a predictable avionics platform in ada: A case study. *RTSS*, 1991.

[14] A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy. *ISLPED*, 2001.

[15] B. Mochocki, X. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. *Technical Report TR-03*, 2002.

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

[17] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. *ISSS*, 1999.

[18] T. Okuma, T. Ishihara, and H. Yasuura. Software energy reduction techniques for variable-voltage processors. *IEEE Design and Test of Computers*, 2001.

[19] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. *MOBICOM*, pages 251–259, 2001.

[20] J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. *ISLPED*, 2001.

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

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

[23] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. *FOCS*, pages 374–382, 1995.

[24] Y. Zhang, X. Hu, and D. Chen. Task scheduling and voltage selection for energy minimization. *DAC*, 2002.