Transition Overhead Aware Voltage Scheduling for Fixed-Priority Real-Time Systems

BREN MOCHOCKI and XIAOBO SHARON HU
University of Notre Dame
and
GANG QUAN
University of South Carolina

Time transition overhead is a critical problem for hard real-time systems that employ dynamic voltage scaling (DVS) for power and energy management. While it is a common practice of much previous work to ignore transition overhead, these algorithms cannot guarantee deadlines and/or are less effective in saving energy when transition overhead is significant and not appropriately dealt with. In this paper we introduce two techniques, one off-line and one on-line, to correctly account for transition overhead in preemptive fixed-priority real-time systems. We present several DVS scheduling algorithms that implement these methods that can guarantee task deadlines under arbitrarily large transition time overheads and reduce energy consumption by as much as 40% when compared to previous methods.

Categories and Subject Descriptors: C.3 [**Special Purpose and Application Based Systems**]: Real-Time and Embedded Systems

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Dynamic Voltage Scaling, Fixed Priority, Low-Power, Scheduling, Transition Overhead

1. INTRODUCTION

Real-time scheduling plays a key role in the low-power design of real-time embedded systems, not only because timing issues are critical, but also because low power design is essentially a resource-usage optimization problem. How to employ scheduling techniques to manage energy sources (such as batteries) to extend the system lifetime and simultaneously meet timing requirements has become a wide spread research area. Many scheduling methods have been published (e.g., [Yao

A preliminary version of this paper appears in [Mochocki et al. 2005]
Author's address: B. Mochocki and X. Hu, Department of CSE, University of Notre Dame, Notre Dame, IN 46556, {bmochock, shu}@cse.nd.edu
G. Quan, Department of CSE, University of South Carolina, Columbia, SC 29208, gquan@cse.sc.edu
This work is supported in part by NSF under grant numbers MIP-9701416, CCR-9988468, CCR02-08992, CNS-0410771 and the CAREER Award CNS-0545913 and by the University of South Carolina Research Program under the Research Productive Scholarship Award.

et al. 1995; Pillai and Shin 2001; Gruian and Kuchcinski 2003; Kim et al. 2004; Seo et al. 2006]). These methods differ in many ways, such as scheduling being done off-line/on-line, handling hard/soft deadline requirements, or assuming fixed/dynamic priority assignments. The core idea of these approaches is to employ scheduling techniques that can exploit modern dynamic configuration capabilities of embedded processors, according to the current or expected workload, to achieve energy efficiency. One such capability is Dynamic Voltage Scaling (DVS).

The performance of a DVS processor can be dynamically adjusted by changing its operational voltage and frequency. A significant limitation of DVS processors, however, is its inability to change the operation voltage and frequency *instantaneously*. This limitation, known as *transition time overhead* can be on the order of tens of microseconds ([AMD 2001; Burd 2001]) to tens of milliseconds ([Compaq 2000]). For systems where execution is blocked during a transition (which is common in many existing commercial processors [AMD 2001; Compaq 2000]), this translates to anywhere from 10^4 to 10^8 lost execution cycles. Ignoring time overhead in this case will likely cause deadline misses, which in turn can result in degraded system performance or even system failure. Another related problem is transition energy overhead, which can actually cause the system's energy consumption to increase if DVS is not used judiciously. Despite these limiting factors, a common practice in the real-time system community is to focus on the "ideal case" in which all overheads are considered negligible.

In this paper, we study the problem of reducing the energy consumption of fixed-priority periodic real-time systems consisting of a single DVS processor with *non-negligible* transition time and energy overhead. Many real-time embedded applications adopt a fixed-priority scheme, such as Rate Monotonic (RM), due to its high predictability, low overhead, and ease of implementation [Liu 2000].

We present two approaches, one off-line and an one on-line, to handle the transition time and energy overhead of DVS processors. The off-line approach generates the schedule during design time and is based on the *a prior* known system specifications. This approach has a very small run-time overhead because it does not compete with running applications for system resources. However, as with other off-line techniques, this approach tends to be pessimistic because it must consider the worst case. Still, off-line methods can be effectively used for practical systems with limited run-time variation or to study the energy-saving potential of design alternatives during design space exploration. For systems with high run-time variability, we present a novel on-line approach, called low-power Limited Demand Analysis with Transition overhead (lpLDAT), which can effectively accommodate run-time variations and save energy. Through experimentation, we demonstrate that lpLDAT can result in significant energy savings when compared to enhancing previous methods to be overhead aware.

The remainder of this paper is organized as follows. Section 2 summarizes the background material, Section 3 presents a motivational example, Section 4 develops the off-line method, Section 5 describes the on-line approach and Section 6 presents the experimental results. Finally, Section 7 concludes the paper. Note that all proofs may be found in the appendix.

2. BACKGROUND AND RELATED WORK

First, the type of systems under consideration and the necessary notation is specified. Next, the pertinent related work is presented.

2.1 System Model

We consider real-time applications consisting of a set of n periodic tasks, $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$. Task T_i is said to have a *higher priority* than task T_j if $i < j$. Each task, T_i, is described by its worst case execution cycles, wc_i, average case execution cycles, ac_i, and best case execution cycles, bc_i, with $wc_i \geq ac_i \geq bc_i$. In addition, each task has a period, p_i, and relative deadline, d_i, with $d_i \leq p_i$. The *utilization* of a task set is the sum of each task's worst case cycles over its period. That is, the worst-case utilization can be computed as

$$U_{wc} = \sum_{i=1}^{n} \frac{wc_i}{p_i}. \tag{1}$$

The average-case utilization, U_{ac}, and the best-case utilization, U_{bc}, can be computed by substituting ac_i and bc_i for wc_i, respectively.

We refer to the k-th invocation of task T_i as job J_i^k. Each job is described by a release time, r_i^k, deadline, d_i^k, finish time, f_i^k, the number of cycles that have been executed, ex_i^k, and *actual* total execution cycles, c_i^k, with $0 \leq ex_i^k \leq c_i^k$ and $bc_i \leq c_i^k \leq wc_i$. During run-time, we refer to the earliest job of each task that has not completed execution as the **current** job for that task, and we index that job with cur, e.g., J_i^{cur} is the current job for task T_i. The *estimated* work remaining for job J_i^{cur}, denoted by w_i^{cur}, is equal to $wc_i - ex_i^{cur}$. If a set of jobs is not associated with a set of periodic tasks, then the superscript is dropped and the subscript indicates the priority of the job (e.g., $J_1 = (r_1, d_1, f_1, wc_1, ex_1, c_1)$ has a higher priority than $J_2 = (r_2, d_2, f_2, wc_2, ex_2, c_2)$). A **ready job** is any job J_i at time t that satisfies $r_i \leq t$, $d_i > t$ and $f_i > t$, while the **active job** is the ready job at time t with the highest priority. As in most DVS work, we assume that each job consumes an equal amount of energy per cycle at a given speed.

A **scheduling point** is any time point t that satisfies either $t = r_i^k$, $t = d_i^k$ or $t = f_i^k | i = 1..n, k = 1..\infty$. We use $\mathcal{TS}$ to represent all scheduling points sorted in ascending order. An individual scheduling point is indexed by i and denoted by ts_i. Note that finish times are estimated based on the worst-case execution cycles for off-line scheduling. For on-line algorithms, they are inserted into $\mathcal{TS}$ as they occur. Once a finish time is inserted, the corresponding deadline is removed from $\mathcal{TS}$. The subset of $\mathcal{TS}$ that includes all points greater than r_i^k and less than or equal to d_i^k is called the set of J_i^k-scheduling points and is denoted by $\mathcal{TS}_i^k$.

The DVS processor used in our system can operate at a finite set of 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}, resulting in the set $\mathcal{S} = \{S_1, ..., 1\}$. Changing from one voltage level to another takes a fixed amount of time, [1] referred to as the *transition interval* (denoted Δt) within which

[1] A variable length transition interval (e.g. the one described in [Burd and Brodersen 2000]) can be approximated by a fixed length interval equal to the maximum switching time.

no tasks can be executed. [2] The transition interval length for a DVS processor alone is usually on the order of 10 to 120 μs ([Intel 2000; AMD 2001; Burd and Brodersen 2000; Pouwelse et al. 2001]). This results from the DC-DC converter changing V_{DD} and the phase-locked loop (or similar technology) changing f_{clk}. However, when considering synchronization with other components in a system, such as off chip memory, the length can be on the order of milliseconds ([Saewong and Rajkumar 2003; Compaq 2000]).

A voltage transition also consumes a variable amount of *transition energy*, denoted as ΔE. Transition energy includes three major parts: (1) the energy consumed by the DC/DC converter, (2) the energy consumed by the CPU during the transition and (3) the energy increase due to executing cycles displaced by the transition interval at higher speeds. This is similar to the model used in [Mochocki et al. 2002].

2.2 Related Work

2.2.1 *DVS scheduling with transition overhead.* A number of researchers have studied voltage scheduling when transition overhead is not negligible. Hong *et al.* [Hong et al. 1998] present two algorithms that solve the off-line voltage scheduling problem. These algorithms take the voltage transitions into consideration, but assume that computation can be performed during a transition, which is often not the case ([AMD 2001; Compaq 2000]). In addition, these algorithms assume no upper limit for the supply voltage, which is not practical. In [Saewong and Rajkumar 2003], Saewong and Rajkumar present an off-line algorithm to schedule FP job sets with a very large transition interval. They essentially select the smallest speed that meets all task deadlines, thus avoiding transitions altogether. Zhang and Chakrabarty consider both overheads when scheduling voltage levels and checkpoint times for fault-tolerant, hard, real-time systems with periodic tasks ([Zhang and Chakrabarty 2004]). They assume that each task can meet all deadlines when running at the smallest processor speed if no faults are present. This assumption eliminates the benefit of DVS in a fault-free environment. Mochochi *et al.* presented an algorithm called Unified Algorithm for Earliest Deadline First (UAEDF), to guarantee task deadlines and minimize the energy consumption while managing transition overhead for EDF-based systems [Mochocki et al. 2002]. A drawback of this algorithm is that it cannot be directly applied to FP systems, as will be shown in Section 3.

There are also a number of DVS scheduling approaches for distributed systems or systems with dependent sub-tasks that consider transition overhead. In a recent work, first presented in [Seo et al. 2004] and extended in [Seo et al. 2006], Seo *et al.* propose an optimal intra-task voltage scheduling method that handles transition overhead during compile time. Though effective, this method cannot be directly applied to the inter-task FP preemptive system we consider. In [Seo et al. 2005], they combine the intra-task method from [Seo et al. 2004] with the EDF-based

[2] Most commercial processors (e.g. [AMD 2001; Compaq 2000]) block executions during the transition process. For processors that do not block instructions during a transition (e.g., [Burd and Brodersen 2000]), a schedule that assumes blocking could be pessimistic, but will guarantee that a valid voltage schedule is reached.

inter-task method from [Yao et al. 1995]. They do not, however, consider inter-task transition overhead in that algorithm. Zhang *et al.* in [Zhang et al. 2003] present an ILP formulation that optimally solves the voltage scheduling problem for multiple processors while considering transition *energy* overhead. They also present an approximation formulated as an LP. Both methods are too complex to be used on-line and fail to account for transition *time* overhead. Andrei *et al.* in [Andrei et al. 2004] also solve the voltage scheduling problem for multiple processors using an ILP that considers both time and energy transition overhead. Because an ILP is used, the run time is exponential with respect to the input size, making this method impractical for many systems. They later proposed a combined off-line on-line technique [Andrei et al. 2005], which has a very small run-time overhead. Even though this algorithm is very efficient, it is only applicable to a statically ordered, non-preemptive systems, which is different from the FP preemptive system we address here.

2.2.2 *On-line DVS scheduling for FP systems.* Some previous research has been conducted regarding on-line DVS for FP real-time tasks, e.g., [Gruian and Kuchcinski 2003; Kim et al. 2003; Pillai and Shin 2001], none of which accounts for transition overhead. Pillai and Shin proposed an algorithm called ccRM, which first computes off-line the maximum speed necessary to meet all task deadlines based on worst-case response time analysis. On-line, the processor speed is scaled down when task instances complete early [Pillai and Shin 2001].

In [Gruian and Kuchcinski 2003], Gurian describes a method to order tasks based on their best to worst case execution cycle ratio, i.e., tasks that are more likely to finish early should be executed first so the resulting slack can be used to save energy. His method is complementary to the approach we present here, as the priority of tasks can be set to match the order prescribed in [Gruian and Kuchcinski 2003].

Kim *et al.* in [Kim et al. 2003] developed a method called **lpWDA** that uses a greedy, on-line algorithm to estimate the amount of slack available and then apply it all to the current job. This algorithm is unique in that it takes slack from both lower and higher priority tasks, as opposed to the method presented [Pillai and Shin 2001] that waits for slack to filter down from higher priority tasks. A serious drawback, in addition to discounting transition time overhead, is that lpWDA is often too aggressive, resulting in wasted energy. We show in Section 3 that the modifications to lpWDA that are required to correctly account for time overhead are not trivial. A later work in [Kim et al. 2004] seeks to minimize the impact of preemptions on the overall system energy consumption. This technique can easily be incorporated into the method we present here for further energy reduction when the preemption overhead is not negligible.

3. MOTIVATIONAL EXAMPLE

In this section we first show that the algorithm, UAEDF [Mochocki et al. 2002], cannot guarantee deadlines when jobs are scheduled according to a FP scheme, such as RM. Next, we show that lpWDA cannot guarantee deadlines when the transition time overhead is not negligible.

Observe the two-task system in Figure 1(a), with $p_1 = d_1 = 3$, $wc_1 = bc_1 = 1$, $p_2 = d_2 = 4$, and $wc_2 = bc_2 = 1$. Figure 1(b) shows the results when the task set is

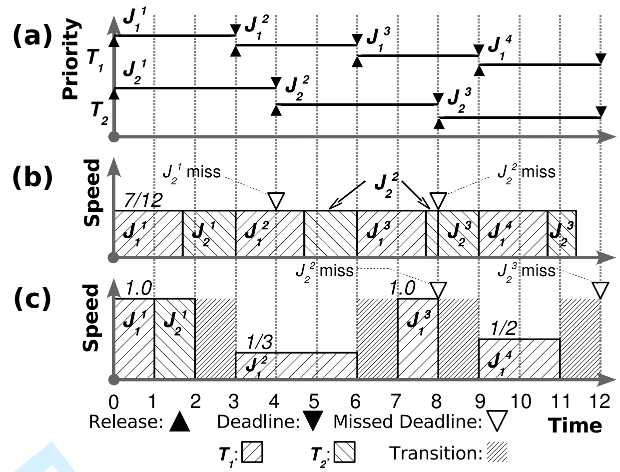


Fig. 1. (a) An example task set consisting of two tasks; (b) Jobs miss deadlines when scheduled according to UAEDF; (c) Jobs miss deadlines when scheduled by lpWDA with a simple modification to account for transition time overhead.

scheduled off-line using UAEDF. Two deadlines of task T_2 are missed at time 4 and time 8. This is due to the fundamental difference in preemption patterns between fixed-priority and dynamic-priority systems. In Section 4, UAEDF is modified to work for FP systems.

Figure 1(c) shows the task execution pattern when tasks always take the worst-case execution cycles and lpWDA is used, assuming a transition interval of $\Delta t = 1$. Note that lpWDA does not explicitly account for time overhead. We could try to solve this problem by reducing the slack identified by Δt time units during every slack calculation. Notice that according to this schedule, jobs J_2^2 and J_2^3 miss their deadlines. Clearly, when transition time overhead is significant, one cannot be too aggressive when employing DVS, otherwise deadlines will be missed. A method to account for time overhead on-line is presented in Section 5.

4. OFF-LINE APPROACH

In this section, an off-line algorithm is developed to handle an arbitrarily large transition time overhead for FP systems. We begin by introducing a class of algorithms that have been shown to effectively manage transition overhead in EDF systems. We then develop a corresponding algorithm for FP systems.

4.1 Critical-interval based scheduling algorithms

A class of voltage scheduling algorithms called *critical-interval scheduling algorithms* (e.g., [Yao et al. 1995; Quan and Hu 2001; 2002]) are particularly suited to statically handling transition overhead. A **critical interval** is the maximum length interval such that the minimum constant speed must be continuously applied to avoid a deadline miss, while a **critical-interval scheduling algorithm** is any algorithm that iteratively identifies, schedules and removes the critical interval. The interval is scheduled by deleting jobs that execute in the interval from the list of unscheduled jobs, while it is removed by adjusting release times and deadlines of unscheduled jobs (based on equation (2) and (3), for example) so that time reserved for the critical interval is not reused.

$$r'_i = r_i - \max\{\min\{r_i, e_j\} - s_j, 0\}, \tag{2}$$

$$d'_i = d_i - \max\{\min\{d_i, e_j\} - s_j, 0\}, \tag{3}$$

$$i = 1..|\mathcal{J}|$$

UAEDF builds on the critical-interval scheduling algorithm called Low Power Earliest Deadline First (LPEDF) [Yao et al. 1995], to deal with the transition time overhead. LPEDF is an optimal critical-interval scheduling algorithm for EDF systems, when time overhead is negligible. UAEDF modifies equations 2 and 3 by extending the new critical interval to include two voltage/speed transitions, i.e., $[s_j - \Delta t, e_j + \Delta t]$. Note that transitions are not inserted twice between adjacent intervals. In the absence of transition overhead and given a continuous voltage range, UAEDF will produce the same schedule as LPEDF.

4.2 Adaptation to FP Systems

Although UAEDF is effective in managing transition overhead in EDF systems, we have shown in section 3 that UAEDF cannot be directly applied to FP systems. Thus, a corresponding FP critical-interval scheduling algorithm is a reasonable starting point for the off-line approach. We utilize an efficient polynomial-time heuristic[3] called fixed-priority Voltage Scheduling for Low Power (VSLP) [Quan and Hu 2001]. The critical interval identified by VSLP is associated with a particular job, the intensity of which can be computed using the following definition [Quan and Hu 2001].

Definition 1. ***J_n-intensity***– *Let t_a and t_b the release or deadline of jobs with priority j where $j \le n$. The J_n-intensity in the interval $[t_a,t_b]$, denoted by $\mathcal{I}_n(t_a,t_b)$, is defined to be:*

$$\mathcal{I}_n(t_a, t_b) = \frac{\sum_{i-1}^{n} \delta(J_i) * wc_i}{t_b - t_a}$$

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

One desirable feature of critical interval scheduling algorithms such as VSLP is that intervals are identified in a monotonically non-increasing order of speed when the transition time overhead is negligible, as shown in Lemma 1 [Quan and Hu 2001].

Lemma 1. *let $\mathcal{A}$ be an arbitrary critical-interval scheduling algorithm. Further, let $\Delta t = 0$. Critical intervals identified by $\mathcal{A}$ are identified in a monotonically non-increasing order by speed.*

[3] Although an optimal voltage scheduling algorithm has also been devised for FP, the FP case is known to be NP-complete [Quan and Hu 2002].

Transition overhead is accounted for by extending the critical interval by Δt in each direction, i.e., $[s_j - \Delta t, e_j + \Delta t]$, where the critical interval for iteration j is $[s_j, e_j]$. However, this additional time may cause consecutive intervals to require a higher speed, which we refer to as a *monotonicity violation*, given in Definition 2.

DEFINITION 2. ***Monotonicity Violation***– *The situation that occurs when the speeds S_{i-1} and S_i of two consecutively identified critical intervals satisfy $S_{i-1} < S_i$.*

Two key observations regarding monotonicity violations are presented in Lemmas 2 and 3.

LEMMA 2. *Let $I_{i-1} = [s_{i-1}, e_{i-1}]$ and $I_i = [s_i, e_i]$ be two consecutively identified critical intervals identified by VSLP with speeds $S_{i-1} = \mathcal{I}_{n_{i-1}}(s_{i-1}, e_{i-1})$ and $S_i = \mathcal{I}_{n_i}(s_i, e_i)$. Additionally, let the scheduled interval for iteration $i-1$ be $[s_{i-1} - \Delta t, e_{i-1} + \Delta t]$. If $S_i > S_{i-1}$ then I_{i-1} and I_i are adjacent.*

LEMMA 3. *Let $I_{i-1} = [s_{i-1}, e_{i-1}]$ and $I_i = [s_i, e_i]$ be two consecutively identified critical intervals identified by VSLP with speeds $S_{i-1} = \mathcal{I}_{n_{i-1}}(s_{i-1}, e_{i-1})$ and $S_i = \mathcal{I}_{n_i}(s_i, e_i)$ such that $S_i > S_{i-1}$ (i.e., a monotonicity violation has occurred). Additionally, let the scheduled interval for iteration $i-1$ be $[s_{i-1} - \Delta t, e_{i-1} + \Delta t]$. The minimum speed at which every job in $\mathcal{J}_{i-1} \cup \mathcal{J}_i$ can execute without a deadline miss is S_{i-1}.*

UAEDF removes monotonicity violations by merging the monotonicity-violation interval with the previously identified critical interval and also adopting its processor speed. Clearly the processor speed is *"higher than necessary"* for the jobs in the monotonicity-violation interval. To be energy efficient, it is desirable that the merged critical interval be kept as short as possible. This will free the maximal amount of time for scheduling any remaining jobs and also minimize the chance of future monotonicity violations. UAEDF delays the execution of the jobs in the merged interval without violating their deadlines and therefore reduces the interval length. In [Quan et al. 2004], an efficient algorithm is presented to find the latest start time and the minimum length critical interval for a given set of FP jobs at a specific speed. We refer to this algorithm as LSTFP.[4] It is tempting to believe that LSTFP can be used directly for handling monotonicity violations, similar to UAEDF. However, the naive usage of this algorithm may lead to deadline misses as explained in Figure 2.

Observe the set of jobs in Figure 2(a). Figure 2(b) shows that the critical interval of J_2 is higher than that of J_1, which is a monotonicity violation that must be dealt with. One method to find the minimum-length interval is to merge the two jobs into one by finding the latest start time of both jobs. However, a FP system scheduled in this way results in the execution pattern displayed in Figure 2(c). Notice that the interval [10, 11] allocated to J_3 is actually used by J_2. Because the deadline of J_3 is earlier than that of J_2, J_3 cannot finish the remaining work and its deadline at time 16 is missed. This situation is referred to as execution inversion.

DEFINITION 3. ***Execution Inversion***– *The situation that occurs when a job J_i is scheduled to be executed during an interval $[t_1, t_2]$ but is instead preempted by a job J_j during that interval, where $j < i$.*

[4]Note that LSTFP was used in [Quan et al. 2004] to reduce leakage energy.

Execution inversion can occur because once a job is scheduled, it is removed from further consideration, provided that a second monotonicity violation does not occur. As shown in Figure 2(c), this is fine for EDF systems, but not suitable for FP systems. To prevent execution inversion, we propose bounding the latest start time to Δt plus the earliest release time of all jobs in current and previous critical intervals. This ensures that J_3 from Figure 2(a) will be completed by time 10 at a speed of 3/7. Lemma 4 formally demonstrates that this method eliminates execution inversion.

LEMMA 4. *Bounding the latest start time of a job set $\mathcal{J}_i$ to $r_i^{min} + \Delta t$ will prevent execution inversion. The value r_i^{min} is the earliest release time of all jobs in $\mathcal{J}_i$.*

Bounding the latest start time according to 4 results in a latest start time of 12 for the critical interval containing J_1 and J_2 in Figure 2. The new algorithm, called UAFP, is presented in Algorithm 1. Lines 3–7 deal with monotonicity violations due to time transition overhead as described throughout this section. Before a final schedule is produced in line 13, transitions to critical intervals insufficient in length to justify the incurred *transition energy overhead* are removed (Lines 9–12). Theorem 1 gives the correctness and complexity of Algorithm 1.

Algorithm 1 UAFP

1: **INPUT:** The job set to be scheduled, $\mathcal{J}$, and the transition interval size, Δt.
2: **OUTPUT:** A valid voltage schedule.
3: **while** $\exists$ an unscheduled job in $\mathcal{J}$ **do**
4:     Identify the next critical interval I_i according to VSLP;
5:     **if** a monotonicity violation is encountered **then**
6:         I'_{i-1} = the minimum interval that completes all jobs in I_i and I_{i-1} such that the interval start time is equal to $\min\{\min\{r_{i-1}^{min}, r_i^{min}\} + \Delta t$, (latest start time by LSTFP)$\}$;
7:         Replace I_i and I_{i-1} with I'_{i-1};
8: *// Handle energy overhead*
9: **for** each critical interval I_i in order of increasing speed **do**
10:     Identify the adjacent interval with minimum speed, I_j, such that the speed of I_j is greater than that of I_i
11:     **if** If energy(I_i) + energy(I_j) + $\Delta E(I_i, I_j)$ > energy(I_i merged with I_j) **then**
12:         merge I_i with I_j
13: Construct the voltage schedule from the resulting set of critical intervals

THEOREM 1. *Algorithm 1 always produces a valid voltage schedule in* $\boldsymbol{O}(N^3)$ *time, given an initially schedulable job set, where N is the number of jobs.*

5. ON-LINE APPROACH

In the previous section, we present a method that can statically account for transition overhead. This technique is advantageous for two reasons. First, the voltage schedule is stored in a table and can be accessed quickly, thus it will not compete with the other tasks for cpu resources. Second, it can effectively exploit the

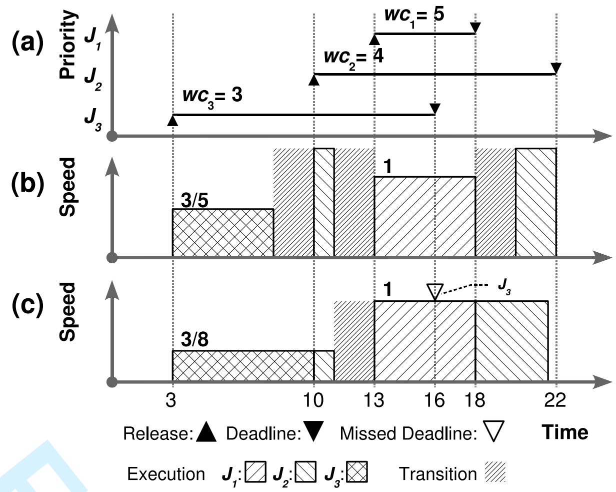


Fig. 2. (a) An example job set. (b) An example monotonicity violation. (c) LSTFP is used to remove the monotonicity violation. The deadline of J_3 is missed at time 16.

known specifications of a particular system to enhance the energy savings and is also able to quantify the energy-saving potential of design alternatives, which can be extremely important during the design space exploration process. The main disadvantage of the off-line scheme is that it is less flexible and adaptive for the dynamic run-time environment, especially when tasks complete much earlier than the worst case. This is the situation where an on-line algorithm becomes more effective.

As mentioned in Section 2.2, the most recent on-line algorithm for preemptive FP systems is called lpWDA [Kim et al. 2003]. Unfortunately, this algorithm suffers from two major drawbacks: *(i)* It is too greedy when selecting a speed for the active job, and *(ii)* it does not account for voltage transition time and energy overhead. We address each of these concerns in Sections 5.1 through 5.3.

5.1 Limited Demand Analysis

The on-line scheduling algorithm can significantly outperform an offline scheduling algorithm if it can effectively exploit slack produced when real-time jobs complete much earlier than the worst case. However, in order to do so, one must be careful when consuming the slack time. Algorithm lpWDA always selects the smallest feasible speed when slack time is available, resulting in an algorithm that aggressively steals slack from future jobs. This may not be the most energy-efficient technique in all situations. For example, observe the schedule in Figure 3, which is the task set from Figure 1 scheduled according to lpWDA. Notice that the speed alternates between a very low speed (1/2 or smaller) and S_{max}. This is because all of the slack is being used by the active job. A more efficient schedule would be aware of the average case cycles (which in this example is equal to the worst case) and be less aggressive when using slack.

Limiting the slack used by the active job in lpWDA requires a careful trade-off

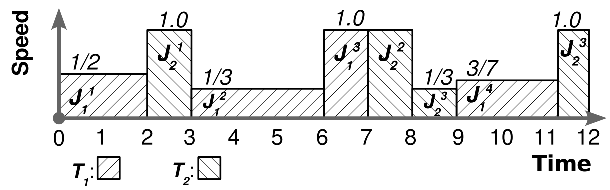


Fig. 3. The schedule produced by lpWDA when executing the task set from Figure 1, without transition overhead. The energy consumed equals 4.25 assuming $power = speed^3$

between being aggressive and being conservative. If one could compute an efficient speed based on the average-case workload, this speed could be used as a *limiter*. If the limiter is higher than the speed predicted by lpWDA, we know that lpWDA is being too aggressive and the limiter speed should be used. An often used concept in DVS research is the **minimum constant speed** that can meet all job deadlines. Due to the convexity of the power function, it is generally not energy efficient for the processor to go below this speed and then switch to a higher speed later, unless there is reason to expect newly available slack ([Pillai and Shin 2001]). Thus the minimum constant speed can serve as a proper limiter.

To find the minimum constant speed for a periodic task system, one can simply examine the case when all tasks are released simultaneously, i.e.,

$$S_{MC} = \max_{i=1}^{n} \min_{ts \in \mathcal{TS}_i^1} Speed(i, ts) \tag{4}$$

and

$$Speed(i, ts) = \frac{\sum_{j=1}^{i} \lceil \frac{ts}{p_j} \rceil \times wc_j}{ts} \tag{5}$$

where $\mathcal{TS}_i^1$ is the set of J_i^1-scheduling points. Our idea is to perform a similar operation as above on-line. Directly applying the formulas in (4) and (5) is not desirable due to its pessimism and time complexity. To overcome unnecessary pessimism, we recompute the minimum constant speed for each job whenever it starts/resumes execution. This allows the actual execution cycles of jobs executed earlier to be considered when appropriate. This also removes the pessimistic assumption of the worst-case phasing. Furthermore, instead of using the worst-case execution cycles, we use the *average*-case execution cycles. Finally, we opt to use the deadline d_i^{cur} of job J_i^{cur} rather than checking every scheduling point in $\mathcal{TS}_i^{cur}$ for the minimum speed. This reduces the time needed to calculate the limiter.

The proposed on-line algorithm, called *low-power Limited Demand Analysis* (lpLDA), is given in Algorithms 2 and 3. Lines 4 and 5 of Algorithm 2 initialize the estimated worst and average case higher priority cycles that must complete before the next deadline of each task. These values are maintained every time a preemption/completion occurs in line 6 of Algorithms 2 and lines 6 – 16 of Algorithm 3. The slack of lower priority jobs is determined in line 9 of Algorithm 2 (for more details on this operation, see [Kim et al. 2003]). Line 10 finds the lowest effective feasible speed for the active task, while Line 11 calculates the speed of the limiter. Finally, Line 12 selects the maximum of the two speeds, essentially restrict-

Algorithm 2 lpLDA

if on system start **then**
  for Each Task $T_i \in \mathcal{T}$ **do**
    $d_i^{cur}{:=}d_i$; $w_i^{cur}{:=}wc_i$;
    $H_i{:=}\sum_{j=0}^{i-1}(\lceil \frac{d_i^{cur}}{p_j} \rceil \times wc_j)$;
    $A_i{:=}\sum_{j=0}^{i-1}(\lceil \frac{d_i^{cur}}{p_j} \rceil \times ac_j)$;
if finish/preempt the active job J_α^{cur} **then updateLoadInfo($\mathcal{T}$,α)**;
if on execute the active job J_α **then**
  Identify $T_\beta | \beta \geq \alpha$ AND d_β^{cur} is minimized;
  Compute $slack_\alpha$ based on workload with respect to T_β;
  $f_{clk} := \frac{w_\alpha^{cur}}{slack_\alpha + w_\alpha^{cur}} \times f_{max}$;
  $f_{limit} := max\{\frac{A_i + ac_\alpha - ex_\alpha^{cur}}{d_i^{cur} - t} |\ i = 1..n\}$;
  $f_{clk} := max\{f_{clk},\ f_{limit}\}$;
  Set the voltage according to f_{clk};

Algorithm 3 updateLoadInfo($\mathcal{T}$,α)

input: Tasks $\mathcal{T}$ and the preempted/completed task index α.
output: Workloads are updated to reflect current execution information.
if T_α is completed **then**
  for each task $T_i \in \mathcal{T}$ with $i < \alpha$ **do**
    $d_\alpha^{cur} := d_\alpha^{cur} + p_\alpha$;
    $H_\alpha := H_\alpha + \sum_{j=0}^{\alpha-1}(\lceil \frac{d_i^{cur}}{p_j} \rceil - \lceil \frac{d_i^{cur} - p_\alpha}{p_j} \rceil) \times wc_j$;
    $A_\alpha := A_\alpha + \sum_{j=0}^{\alpha-1}(\lceil \frac{d_i^{cur}}{p_j} \rceil - \lceil \frac{d_i^{cur} - p_\alpha}{p_j} \rceil) \times ac_j$;
  for each task $T_i \in \mathcal{T}$ with $i > \alpha$ **do**
    $H_i := H_i - (wc_\alpha - ex_\alpha^k)$;
    $A_i := A_i - max\{0, ac_\alpha - ex_\alpha^k\}$;
  $w_\alpha^{cur} := wc_\alpha$; // reset for next job of T_α
else
  $temp := wc_\alpha - ex_i^{cur}$;
  for each task $T_i \in \mathcal{T}$ with $i > \alpha$ **do**
    $H_i := H_i - w_\alpha^{cur} + temp$;
    $A_i := A_i - w_\alpha^{cur} + temp$;
  $w_\alpha^{cur} := temp$;

ing the amount of slack that would be used by lpWDA alone. Applying lpLDA to the example task set in Figure 1 produces the results in Figure 4 when all jobs require their worst case cycles. Notice that the energy is reduced by about 42% when compared to lpWDA. Theorem 2 states the correctness of lpLDA in terms of satisfying real-time requirements.

THEOREM 2. *The schedule produced by lpLDA guarantees all system deadlines, and has a computational complexity of $O(n)$ per scheduling point, where n is the number of tasks in the system.*

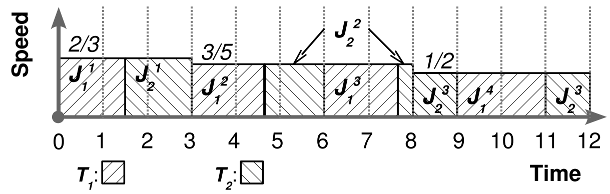


Fig. 4. The schedule produced by lpLDA when executing the task set from Figure 1, without transition overhead. The energy consumed equals 2.47 assuming $power = speed^3$

5.2 Transition Time Overhead

Figure 4 shows that lpLDA can significantly outperform lpWDA in term of energy savings. However, it still suffers from the same drawback as lpWDA with regard to transition time overhead: When transition time overhead is not negligible, real-time jobs can miss their deadlines. In this sub section, we develop a technique based on lpLDA that can deal with the transition time overhead.

Transition time overhead can complicate on-line voltage scheduling in several ways. The most straightforward effect is that time overhead reduces the available slack. Since the job set is schedulable with S_{max}, the processor speed can only be reduced to a lower speed if there is enough slack for at least two transition intervals, one to the lower speed and another to return to S_{max} if necessary. Furthermore, if the available slack can only tolerate less than two transitions but more than one, we can still run the job with the current speed and later return later to S_{max} to guarantee the deadlines if necessary. Otherwise, we need to run the jobs with S_{max}.

It seems that, once we determine the processor speed on-line similar to that in Algorithm **lpLDA**, the above strategy could resolve the transition time overhead problem. However, there are two complications need to be dealt with. *First*, a higher priority job may be released during a transition. In this case, the target speed of the current transition may be insufficient to meet the deadline of the new job. We refer to this problem as a *transition error*. *Second*, the slack consumed by a transition interval that starts at the release of a higher priority task (i.e., the preemption of the current task) may cause the minimum feasible speed of the new task to be greater than the maximum processor speed. We refer to this problem as a *preemption error*. These scenarios are illustrated graphically in Figure 5.

In Figure 5, notice that job J_2^{cur} is released within one transition interval of the scheduling point t. Thus, scheduling will not take place at r_2^{cur} if there is a transition starting at time t. This will cause a transition error if the speed requirement of J_2^{cur} is greater than the new speed. Next, J_1^{cur} is within two transition intervals of t. If the speed selection for J_1^{cur} is delayed until r_1^{cur}, then there may not be sufficient time to change to the speed required to meet the deadline of J_1^{cur}, thus causing a preemption error.

In order to account for time overhead when estimating the available slack, one must predict potential future necessary transitions. To further our discussion, we first present the following definitions.

DEFINITION 4. ***Pre-release Scheduling Point***– *The time point t that satisfies $t = r_i^k - \Delta t | i = 1..n, k = 1..\infty$. Pre-release scheduling points replace the correspond-*

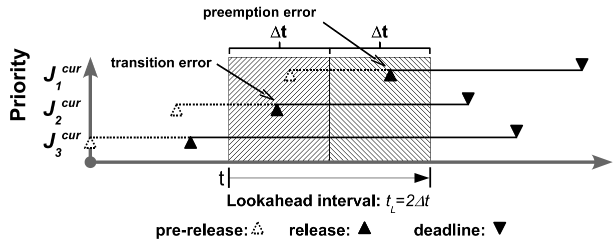


Fig. 5. An example of pre-release points, the lookahead interval at time t, the system lookahead (t_L) and possible transition and preemption errors. The critical job at time t is J_1^{cur}.

ing scheduling points that satisfy $t = r_i^k | i = 1..n, k = 1..\infty$ *in* $\mathcal{TS}$ *whenever time transition overhead is not negligible.*

The pre-release scheduling points of J_1^{cur}, J_2^{cur} and J_3^{cur} are illustrated in Figure 5. Pre-release scheduling points warn the system that a preemption may occur in the near future and are essential if a speed is to remain feasible after a voltage transition. However, the pre-release points alone will do no good if the associated job is not included in the scheduling process. To this end, we introduce the concept of *lookahead interval* to limit the number of observed pre-release scheduling points.

Definition 5. ***Lookahead Interval***– *An interval that begins at a specific scheduling point,* ts_i*, and ends at time* $ts_i + t_L$*. The value* t_L *is referred to as the system lookahead. The lookahead interval at time* ts_i *is denoted by* L_i*.*

The lookahead interval is given by the arrow extending to the right from time t in Figure 5.

Examining jobs in the lookahead interval helps to identify if the processor speed can be updated and at the same time avoid transition and preemption errors. However, how much *lookahead* is needed deserves careful examination since too much will increase scheduling complexity and too little may not be sufficient for meeting deadlines. Theorem 3 shows that $t_L = 2\Delta t$ is the sufficient and necessary lookahead to prevent transition and preemption errors for an arbitrary real-time job set.

Theorem 3. *Let job set* $\mathcal{T}$ *be schedulable under* S_{max} *and the transition time overhead be* Δt*. Assume that the processor is set to a new processor speed other than* S_{max} *only when the available slack is large enough to contain at least two transition overheads. Then the system lookahead* $2\Delta t$ *is sufficient and necessary to obtain a transition/preemption-error-free on-line schedule for an arbitrary real-time job set.*

Further, note that the ready job with the highest priority during the lookahead interval L_i will eventually possess the processor after the transition. We therefore name this job the **critical job**. The critical job of a look-ahead interval L_i is denoted $JC(L_i)$. For example, J_1^{cur} is the critical job at time t in Figure 5. When determining the speed for the critical job, we adopt a heuristic rather than determine the exact execution pattern for the sake of computation efficiency. Our heuristic assumes that no jobs may be executed from the time the transition begins to the release of the critical job. In Figure 5, this heuristic pessimistically assumes that neither J_2^{cur} or J_3^{cur} can execute between $t + \Delta t$ and r_1^{cur}. The advantage of this heuristic is that the slack estimation is as fast as that of lpLDA and the estimated speed requirement of the critical job will always be greater than or equal to the largest actual speed requirement of all ready jobs in the look-ahead interval. [5]

Based on Theorem 3 and above observations, we develop a new algorithm that extends Algorithm **lpLDA** to deal with transition overhead. The implementation of this algorithm, known as lpLDAt, is given in Algorithm 4. For now ignore lines marked with ***. First, pre-release scheduling points are inserted to ensure that scheduling occurs with enough time to include a speed/voltage transition. When a scheduling point ts_i is encountered (line 8), the critical job J_α is selected by scanning each task and finding the highest priority task that has a ready job in the interval $[t, t + 2\Delta t]$ (line 11). Next, the same method used by lpWDA is used to estimate the slack available to J_α (line 14). Then, the speed for the processor is calculated by determining if the slack is sufficient for one or two transition intervals (lines 15–17). If a speed different from the current speed and lower than S_{max} is selected, then the processor is set either to that speed or the limiter, whichever speed is higher (lines 18).

THEOREM 4. *The schedule produced by* ***lpLDAt*** *guarantees all system deadlines and has a computational complexity of $O(n)$ per scheduling point, where n is the number of tasks in the system.*

5.3 Energy Overhead

Extra energy will be consumed during a voltage transition because (1) a voltage transition itself consumes energy and (2) voltage transitions take a fixed amount of time within which no jobs can be executed. Therefore, the processor needs to adopt a higher speed elsewhere to accommodate the transition interval. To illustrate effect (2), examine the data in Figure 6. The data is obtained for the task set in Figure 1 as the best case vs. worst case execution cycle ration (BC/WC) is varied. All energy numbers are normalized against the energy consumed when executing at the maximum processor speed, S_{max}, without DVS. The solid line represents the energy consumed when executing at the minimum constant speed without DVS, i.e., S_{MC}.

Figure 6 shows the energy consumed by executing the task set from Figure 1 at the speeds selected by lpLDAt for various sizes of Δt, and various values for the BC/WC of each task. The reader will immediately notice that as Δt becomes large, the benefit gained from DVS quickly vanishes. This result is due to the fact that lpLDAt frequently varies the processor speed to exploit slack, which can introduce more transitions than necessary when transition overhead is not negligible.

To ensure that time overhead does not cause an energy increase over using just S_{MC}, we propose exploiting slack less aggressively, thus avoiding transitions that

[5] This statement is intuitively correct because lpLDA will attempt to guarantee the deadline of all lower priority jobs as well as the critical job, so removing some slack from the calculation will inflate the required speed. If the pessimistically estimated speed is greater than S_{max}, then S_{max} is selected, which must be correct if the task set is schedulable.

Algorithm 4 lpLDAt ***(lpLDAT with ***)***

```
if system start then
    Initialize each task as is done in Algorithm 2;
    *** energyS_max := estimate(T,S_max);
    *** energyS_MC := estimate(T,S_MC);
    S'_max := S_max;
    *** if energyS_max ≥ energyS_MC then S'_max := S_MC;
    f_clk := S'_max;
if current time ts ∈ TS and ts is a job completion/pre-release then
    Find J_α, the active job;
    updateLoadInfo(T,α);
    J_α := JC([ts,ts + 2Δt]);
    t := max{ts + Δt, r_α^cur};
    Set f_limit as is done in Algorithm 2;
    Compute slack_α based on workload starting at t;
    if slack_α is large enough for 2 transitions at the speed S_2Δt where S_2Δt ≤
    S'_max then f_clk := S_2Δt;
    else if slack_α is large enough for 1 transition at the speed S_Δt where S_Δt ≤
    f_prev and f_prev is the previous speed then f_clk := f_prev;
    else f_clk := S'_max;
    if f_clk < f_limit AND f_clk ≠ f_prev then f_clk := f_limit;
    *** if f_clk < f_prev then check_energy_overhead();
    Set the voltage according to f_clk;
```

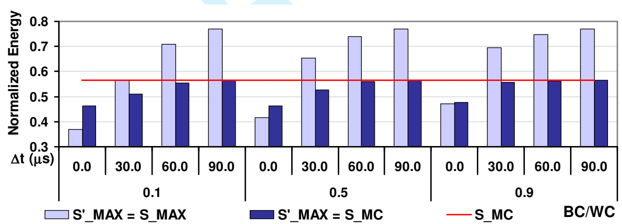


Fig. 6. The energy consumed by to the task set from Figure 1 when applying lpLDAt with two different values for S'_{max}.

would increase energy instead of reducing it. The maximum speed, S_{max} (which is assumed to equal 1 when normalized) is used implicitly in Lines 9 and 10 of Algorithm 2 when determining f_{clk}. This leads to an "overestimation" of slack time when transition overhead is not negligible. If we choose a lower speed for S_{max}, it can be readily used to scale these workload values and will result in a more conservative slack exploitation. We refer to the adjusted maximum speed as S'_{max}.

Our strategy is to compute a good value for S'_{max} off-line and use it on-line. One may be tempted to select S_{MC} as the speed for S'_{max}. Though this ensures that no curve will appear above the S_{MC} line in Figure 6 and also guarantees all task deadlines, doing so drastically reduces the amount of slack available when Δt is small and jobs finish much earlier than the worst case. This situation is illustrated in Figure 6. Notice that when BC/WC = 0.1 and the time overhead is zero, S_{max} is a better choice than S_{MC}.

We choose a good S'_{max} as follows. We estimate the energy that lpLDAt will consume at speed S_{MC} and S_{max}, which is done by simulating the execution of tasks up to the system hyper-period a fixed number of times. The process is repeated twice, once with $S'_{max} = S_{max}$ and once with $S'_{max} = S_{MC}$. The level that consumes less energy during this estimation step is selected[6].

Algorithm 4 with the lines marked by *** represents the complete on-line algorithm, called lpLDAT. Here we will focus on the parts that are different from lpLDAt. Lines 3–6 determine the adjusted maximum speed S'_{max} as described above. This step occurs off-line. On-line, after a new speed is selected, if the selection results in a voltage transition from S_i to S_j, then there is one final check (Line 19), which ensures that ΔE does not locally dominate the energy saved by changing the lower voltage level. If executing the workload of J_α^{cur} at S_i consumes less energy than executing at $S_j + 2\Delta E$ and $S_i > S_j$, then the voltage transition is rejected. Otherwise, S_j is adopted as the new processor speed. Theorem 5 states the correctness of Algorithm 4.

THEOREM 5. *The policy followed by **lpLDAT** will guarantee all system deadlines, and has a computational complexity of $O(n)$ per scheduling point, where n is the number of tasks in the system.*

6. EXPERIMENTAL RESULTS

In this section we quantify the effectiveness of the off-line and on-line algorithms on both randomly generated task sets as well as several real-world task sets. First, the system power model used in the experiments is presented. Next, the algorithms evaluated in this section are described, along with the run-time complexity of the each algorithm. Finally, the performance of the proposed algorithms are evaluated in detail.

6.1 System Power Model

The processor model used is from the work in [Martin et al. 2002], while the DC/DC converter model can be found in [Burd 2001], which includes the following four equations:

$$P_{AC} = C_{eff} V_{dd}^2 f_{clk} \tag{6}$$

$$P_{DC} = V_{dd} K_3 e^{K_4 V_{dd} + K_5 V_{bs}} + |V_{bs}| I_j \tag{7}$$

$$f_{clk} = \frac{((1 + K_1)V_{dd} + K_2 V_{bs} - V_{th1})^\alpha}{L_d K_6} \tag{8}$$

$$\Delta E(V_1, V_2) = C_r |V_1^2 - V_2^2| \tag{9}$$

[6] Although our experiments show that selecting a speed between S_{max} and S_{MC} may be beneficial, identifying the ideal S'_{max} is not a trivial problem. This is because it may change during run-time and depends on the actual execution cycles of each job. Because the focus of this paper is handling time overhead while meeting deadlines, we leave this problem for future work.

Variable	Value	Variable	Value	Variable	Value
K_1	0.053	K_6	$51x10^{-12}$	C_{eff}	$1.11x10^{-9}$ F
K_2	0.140	K_7	-0.132	L_d	37
K_3	$3.0x10^{-9}$	V_{bs}	0 V	C_r	$1x10^{-6}$ F
K_4	1.63	L_g	$4x10^6$	α	1.5
K_5	3.65	V_{th1}	0.359 V	I_j	$2.4x10^{-10}$ A

Table I. Power and delay parameters for the 180-nm process.

where V_{dd} is the supply voltage, P_{AC} is the dynamic power, P_{DC} is the leakage power, f_{clk} is the clock frequency, and $\Delta E(V_1, V_2)$ is the energy consumed by the DC/DC converter when switching between voltage levels V_1 and V_2. The remaining symbols are device-dependent constants, the values of which are given in Table I. For all experiments we assume there are 32 frequency levels available in the range of [60, 600] MHz, with corresponding voltage levels distributed evenly in the range [0.6, 1.4] V, unless noted otherwise.

6.2 Algorithms and On-line Complexity

We examined three off-line and three on-line algorithms, including:

—**UAFP** The proposed off-line algorithm from Section 4.

—**O_UAFP** The proposed off-line algorithm from Section 4, with knowledge of actual execution cycles (i.e., Oracle UAFP). This algorithm is used as a reference to see how much improvement to the other algorithms is possible.

—**S_MC** The off-line algorithm that selects the minimum constant speed for each task instance (i.e., DVS is not applied).

—**lpLDAT** The proposed on-line algorithm from Section 5.

—**ccRM_Dt** The on-line algorithm ccRM from [Pillai and Shin 2001], modified with the results from Section 5 to account for transition overhead[7].

—**lpWDA_Dt** The on-line algorithm lpWDA from [Kim et al. 2003], modified with the results from Section 5 to account for transition overhead[7].

Because the bulk of the work of UAFP is done off-line, its run-time complexity only consists of maintaining **(1)** a table index that increases sequentially and **(2)** a timer interrupt that is reset at each scheduling point. The run-time of UAFP is clearly constant with respect to the number of tasks in the system. However, the on-line algorithms each have a linear run-time, so a more in-depth look is necessary.

Figure 7 illustrates the maximum execution time of each algorithm on the target CPU vs. the number of tasks in the system. The graph was generated by executing

[7]Neither ccRM nor lpWDA consider time transition overhead, which is required to guarantee deadlines when the overhead is not negligible. The time overhead modifications to lpWDA are identical to lpLDAT, while those for ccRM are very similar. The energy-overhead modifications for both algorithms are identical to lpLDAT Line 19

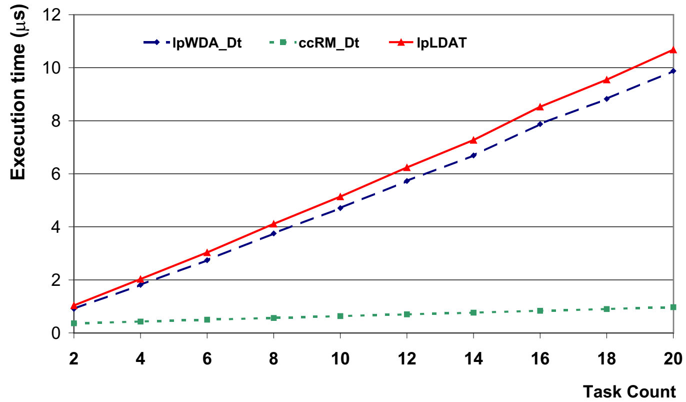


Fig. 7. Maximum execution time for the on-line scheduling algorithms vs. the number of tasks on a 600-MHz processor.

each algorithm on a cycle-accurate instruction-set simulator of the ARM instruction set, called SimIT-ARM [Qin 2004]. Clearly, ccRM_Dt has a large advantage with respect to time complexity when compared to other on-line algorithms. However, because the worst-case execution times of the benchmark applications are on the order of milliseconds, the execution times of even the more complex algorithms are less than 1% of the task execution times until there are 20 or more tasks in the system. This is true even at lower operating frequencies, because the task and scheduler execution times scale simultaneously. Because the scheduler is invoked upon each release and completion, it is sufficient to increase the worst-case execution time of each task by 2 times the worst-case scheduler execution time to account for the scheduler overhead.

6.3 Results

In this section we examine the performance of each algorithm on real-world and randomly-generated task sets. For all experiments, the values illustrated with lines represent off-line algorithms (UAFP and O_UAFP) and those with columns illustrate on-line algorithms (ccRM_Dt, lpWDA_Dt and lpLDAT).

6.3.1 *Computerized Numeric Controller.* The first benchmark is a Computerized Numeric Controller (CNC) task set based on the work by Kim *et al.* in [Kim et al. 1996]. The results are displayed in Figure 8.

Figure 8(a) displays the energy consumed by each algorithm as the length of the transition interval, Δt is increased from 0 to 1800 μs. As expected, the oracle off-line algorithm O_UAFP has the best performance in all cases. At a high BC/WC execution cycle ratio (around 0.9), the practical off-line algorithm, UAFP performs as good or better than the on-line algorithms, and achieves up to 12% energy reduction when the time overhead is 1800 μs. UAFP retains these savings at smaller BC/WC ratios with Δt values of 900 or 1800μs, but the on-line algorithms become superior when the overhead is less than 300 μs and the BC/WC ratio is 0.5 or less. Clearly, the most effective on-line algorithm is lpLDAT, which saves as much as 20% of the energy of the next best on-line algorithm. With a large

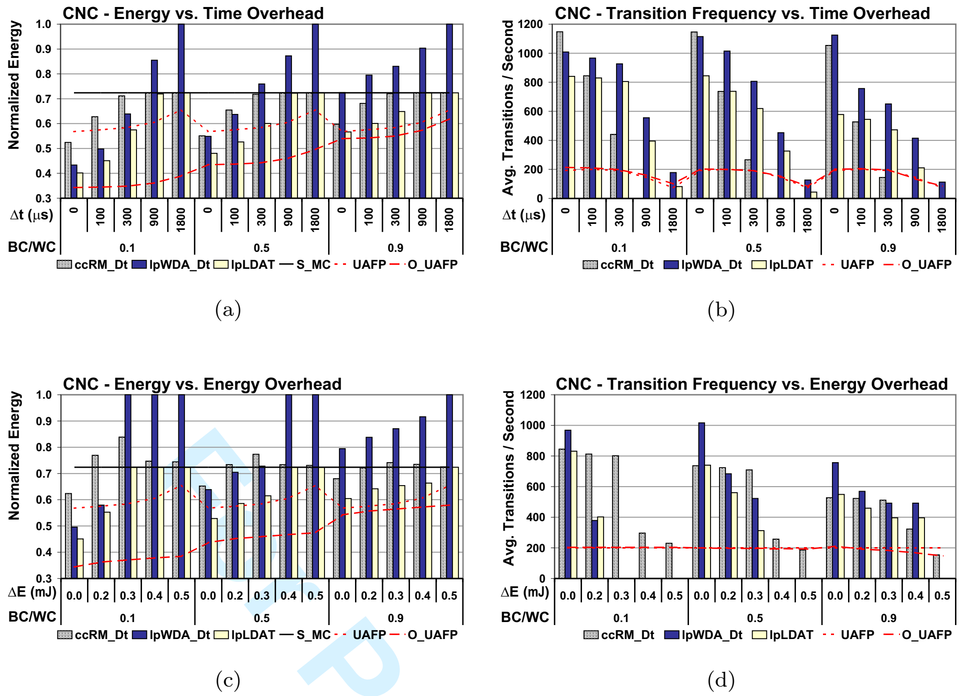
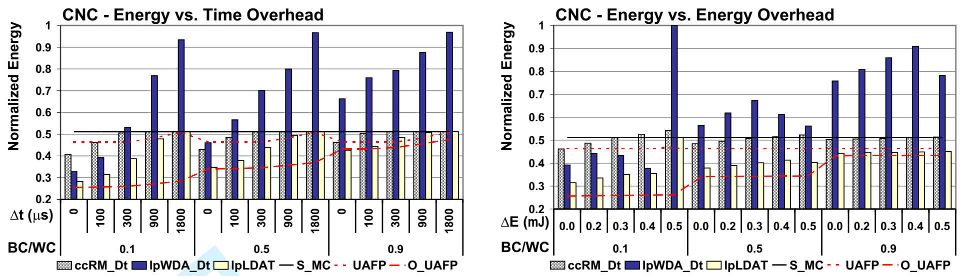


Fig. 8. CNC task set. (a) The normalized energy and (b) average transition frequency vs. an increasing **time overhead**. (c) The normalized energy and (d) average transition frequency vs. an increasing **energy overhead**. All graphs also vary the BC/WC ratio.

transition interval, the on-line algorithms saturate to the two non-DVS methods, S_MC in the case of lpLDAT and ccRM_Dt, and 1 in the case of lpWDA_Dt. The off-line algorithms, however, can still maintain some savings when the overhead extends past 1800μs.

Figure 8(b) illustrates how the impact of transition overhead is minimized. The x-axis shows the BC/WC ratio and the size of Δt, while the y-axis shows the average number of transitions per second. One observation is that as the overhead increases, the transition frequency decreases. This is clearly necessary, as longer transition intervals make meeting deadlines more difficult if there is an excessive number of transitions. Another key observation is that the off-line algorithms tend to have fewer transitions. This is because there is more time available off-line to explore the range of possible voltage and frequency schedules.

Figures 8(c) and 8(d) show the energy performance of each algorithm when the maximum energy overhead is increased. At lower BC/WC ratios, lpLDAT is superior when the energy overhead is small, but quickly saturates to S_MC when the energy overhead becomes large. This is because even though there are more opportunities to reduce the voltage level as jobs finish early, the relative impact of energy overhead is larger because **(1)** there are fewer cycles to execute at the lower speeds and **(2)** the voltage transitions tend to be between levels that are farther apart. Once again, at larger BC/WC ratios, UAFP has the best energy performance of the practical algorithms. Figure 8(d) again illustrates that time overhead is managed by reducing the number of voltage/frequency transitions.



(a) (b)

Fig. 9. CNC task set with a maximum frequency of 2.5 GHz and voltage of 3.0 V. In (a) the time overhead is varied while in (b) the energy overhead is varied.

One additional experiment was run to evaluate the effect of higher voltage and frequency levels. For this experiment, the maximum frequency was set to 2.5 GHz with a corresponding voltage of 3.0 V. Figure 9(a) displayes the energy vs. time overhead. Notice that the trend is similar to Figure 8(a), but the potential energy reduction is larger. This is due to the squared increase in energy resulting from the larger maximum voltage. Figure 9(b), on the other hand, looks considerably different from Figure 8(c). This is because the energy consumed by the energy transitions is less significant when compared to the increased dynamic energy from the higher maximum voltage and frequency.

6.3.2 *Inertial Navigation System.* Figure 10 illustrates the performance of each algorithm on an Inertial Navigation System (INS), also examined in [Burns et al. 1995].

The results for the on-line algorithms are similar to the CNC task set, but the results for UAFP are not as significant as before. Due to the nature of the task set, there are very few opportunities for transitions off-line if the worst-case execution cycles must be assumed, resulting in a very modest 3% decrease in energy below S_MC when using UAFP. However, the on-line algorithms can take advantage of tasks that complete early, resulting in savings of as much as 40% below S_MC when using lpLDAT. Again, lpLDAT achieves superior energy savings, equaling those of the oracle algorithm when the BC/WC ratio is high and Δt is less than 300μs.

The performance of lpLDAT is surprisingly good as energy overhead increases, as it again achieves energy savings close to the oracle algorithm. As shown in Figure 10(d), this occurs because lpLDAT makes better use of fewer voltage transitions.

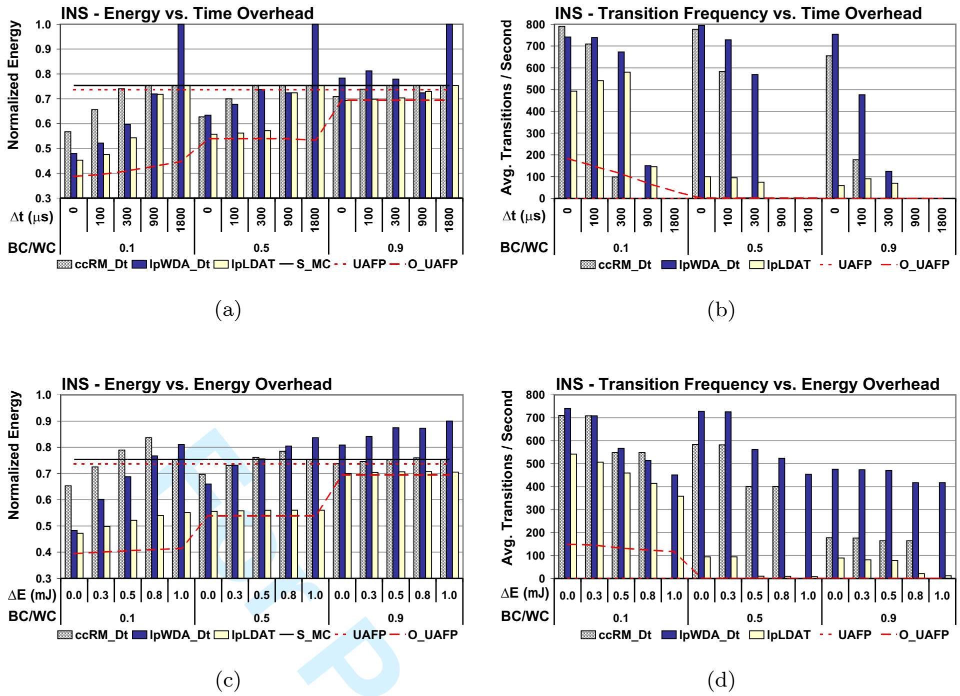


Fig. 10. INS task set. (a) The normalized energy and (b) average transition frequency vs. an increasing **time overhead**. (c) The normalized energy and (d) average transition frequency vs. an increasing **energy overhead**. All graphs also vary the BC/WC ratio.

6.3.3 *Randomly Generated Tasks.* In this section, we examine the energy performance of each algorithm on randomly generated task sets of 4 and 8 tasks with a utilization of 0.4 and 0.8. Each task has its period and deadline randomly selected from a uniform distribution in the range [1, 10] ms. The results are illustrated in Figures 11 and 12.

In the first set of experiments, Δt was varied from 0 to 1800 μs and the BC/WC ratio was varied from 0.1 to 0.9. In Figure 11(a), there are 4 tasks per set, each with a utilization of 0.4. Again, with a low time overhead and BC/WC ratio, lpLDAT is the best choice, consuming about 20% less energy than lpWDA_Dt with Δt=0 and a BC/WC ratio of 0.1. Unlike CNC and INS, when Δt is greater than 900μs, even UAFP saturates at S_MC. Increasing the worst-case utilization to 0.8 in Figure 11(b) increases the potential savings for the on-line algorithms when the BC/WC ratio is 0.5 or less. This is likely due both to the larger range of possible execution times and the better lineup of the scheduled speeds and the voltage range of the CPU. The gains from UAFP are modest, with around a 2% savings compared to the on-line algorithms when BC/WC is 0.9 and Δt is 100 or 300 μs.

Next, the number of tasks is increased to 8 in Figure 11(c). In this case, the results are very similar to the 4 task case, other than lpWDA_Dt, which experiences a 20%

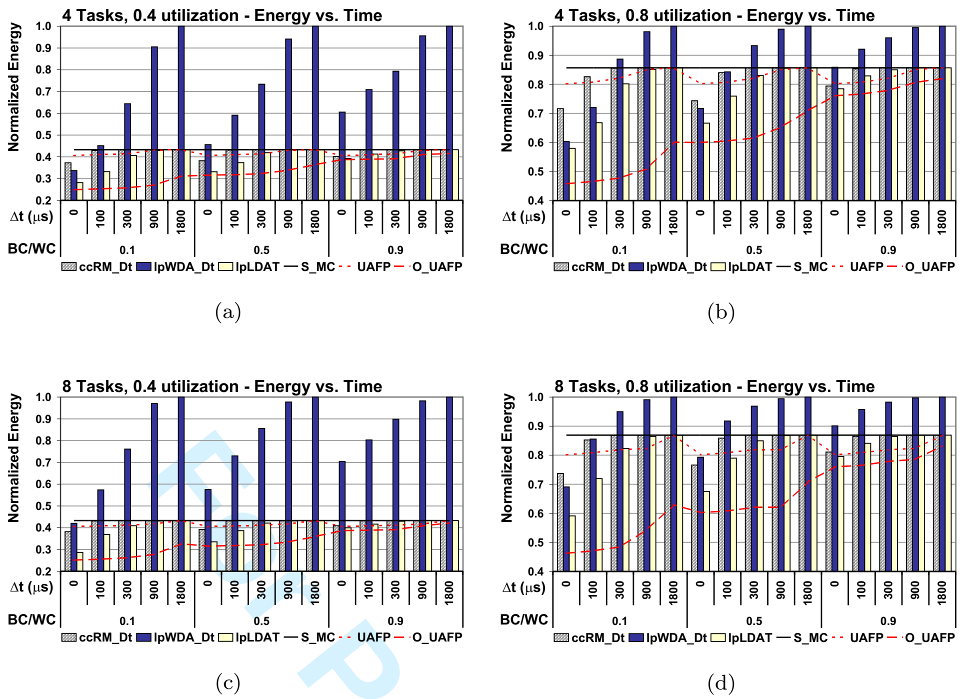
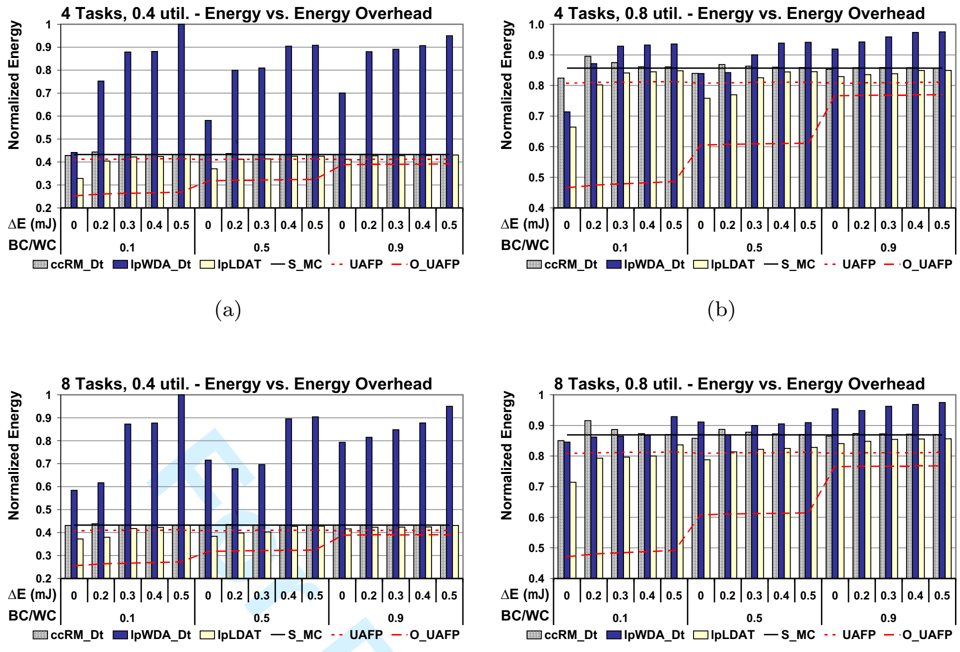


Fig. 11. Randomly-generated tasks. In each graph, the normalized energy consumption vs. time overhead and best-case/worst-case ratio is displayed. The parameters for each graph are (a) 4 tasks with a worst-case utilization of 0.4, (b) 4 tasks with a worst-case utilization of 0.8, (c) 8 tasks with a worst-case utilization of 0.4 and (d) 8 tasks with a worst-case utilization of 0.8.

increase in energy. There is also around a 9% increase in energy for lpLDAT with a small Δt, but the energy remains the same as the 4 task case when Δt increases. For the 0.8-utilization 8-task experiment in Figure 11(d), both the increase in potential savings of the on-line algorithms from the higher utilization and the jump in energy consumed for lpWDA_Dt and lpLDAT are apparent.

In the final set of experiments, the energy overhead is increased from 0 to 0.5 mJ. In all cases, a time overhead of 100μs was used. In Figure 12(a) with 4 tasks and a utilization of 0.4, lpLDAT outperforms UAFP only when the energy overhead is very small. As the energy overhead increases, the on-line algorithms quickly approach either S_MC or 1.0. Additionally, UAFP only reduces the energy consumption by around 5% for all domain values. Interestingly, O_UAFP still reports a potential energy savings of up to 42% with a BC/WC ratio of 0.1. However, the potential savings quickly decrease as the BC/WC ratio increases. This implies that better estimations of the actual execution cycles are required to optimally reduce the impact of an increasing energy overhead.

Increasing the utilization to 0.8 in Figure 12(b) increases the savings for all algorithms in a similar way to the Δt experiment. With a non-zero energy overhead,



(a) (b) (c) (d)

Fig. 12. Randomly-generated tasks. In each graph, the normalized energy consumption vs. energy overhead and best-case/worst-case ratio is displayed. The parameters for each graph are (a) 4 tasks with a worst-case utilization of 0.4, (b) 4 tasks with a worst-case utilization of 0.8, (c) 8 tasks with a worst-case utilization of 0.4 and (d) 8 tasks with a worst-case utilization of 0.8.

lpLDAT is about 5% better than UAFP with a BC/WC of 0.5 and a maximum ΔE of 0.2 mJ, but for larger energy overhead values, UAFP is the clear winner, beating S_MC by about 6%.

As was seen in the Δt experiment, increasing the number of jobs causes the energy consumed by the lpWDA_Dt schedule to increase. Otherwise, the results for Figures 12(c) and 12(d) are similar to the 4-task case.

7. SUMMARY

Time transition overhead is a critical problem for hard real-time systems that employ dynamic voltage scaling for power and energy management. In this paper we presented both off-line and on-line techniques to correctly account for arbitrarily large transition intervals and developed several DVS scheduling algorithms that implement these methods. This includes an off-line algorithm called UAFP and an on-line algorithm called lpLDAT. Additionally, two previous algorithms, ccRM and lpWDA were updated to include our on-line time overhead technique. We have shown on both randomly generated and real-world task sets that lpLDAT outperforms the other algorithms in most scenarios. In cases where the BC/WC ratio is near 1 or the transition interval is large, UAFP could be a better choice. With a high worst-case utilization and time/energy overhead but a small BC/WC ratio, the oracle off-line reports a significant potential for energy reduction compared to both the on-line and off-line practical algorithms. This shows that better estimates of the actual execution cycles can in turn lead to better transition overhead management.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/todaes/2006-1-1/p1-mochocki`.

REFERENCES

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

Andrei, A., Schmitz, M., Eles, P., Peng, Z., and Al-Hashimi, B. M. 2004. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, Washington, DC, USA, 10518.

Andrei, A., Schmitz, M. T., Eles, P., Peng, Z., and Hashimi, B. M. A. 2005. Quasi-static voltage scaling for energy minimization with time constraints. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, Washington, DC, USA, 514–519.

Burd, T. D. 2001. Energy-efficient processor system design. Ph.D. thesis, University of California, Berkeley, Berkeley, CA.

Burd, T. D. and Brodersen, R. W. 2000. Design issues for dynamic voltage scaling. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISPLED)*. 9–14.

Burns, A., Tindell, K., and Wellings, A. 1995. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering 21,* 5 (May), 475–480.

Compaq 2000. Compaq ipaq h3600 hardware design specification - version 0.2f. online- http://www.handhelds.org/ Compaq/ iPAQH3600/ iPAQ_H3600.html.

Gruian, F. and Kuchcinski, K. 2003. Uncertainty-based scheduling: energy-efficient ordering for tasks with variable execution time. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED)*. ACM Press, New York, NY, USA, 465–468.

Hong, I., Qu, G., Potkonjak, M., and Srivastava, M. B. 1998. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of the 19th Real-Time Systems Symposium (RTSS)*. 178–187.

Intel. 2000. The intel xscale microarchitecture. Tech. rep., Intel Corporation.

Kim, N., Ryu, M., Hong, S., Saksena, M., ho Choi, C., and Shin, H. 1996. 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)*. 300–310.

Kim, W., Kim, J., and Min, S. L. 2003. Dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using work-demand analysis. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISPLED)*. 396–401.

Kim, W., Kim, J., and Min, S. L. 2004. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the 2004 international symposium on Low power electronics and design (ISLPED)*. ACM Press, New York, NY, USA, 393–398.

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

Martin, S. M., Flautner, K., Mudge, T., and Blaauw, D. 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-Aided design (ICCAD)*. 721–725.

Mochocki, B., Hu, X. S., and Quan, G. 2002. A realistic variable voltage scheduling model for real-time applications. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-Aided design (ICCAD)*. 726–731.

Mochocki, B. C., Hu, X. S., and Quan, G. 2005. Practical on-line dvs scheduling for fixed-priority real-time systems. In *11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 224–233.

Pillai, P. and Shin, K. G. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP)*. 89–102.

Pouwelse, J., Langendoen, K., and Sips, H. 2001. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th annual international conference on Mobile computing and networking (MOBICOM)*. 251–259.

Qin, W. 2004. Simit-ARM: Very fast functional and cycle-accurate simulators for ARM. `http://http://sourceforge.net/projects/simit-arm/`.

Quan, G. and Hu, X. S. 2001. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the Design Automation Conference (DAC)*. 828–833.

Quan, G. and Hu, X. S. 2002. Minimum energy fixed-priority scheduling for variable voltage processors. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*. 782–787.

Quan, G., Niu, L., Hu, X. S., and Mochocki, B. 2004. Fixed priority scheduling for reducing overall energy on variable voltage processors. In *Proceedings of the 25th Real-Time Systems Symposium (RTSS)*. 309–318.

Saewong, S. and Rajkumar, R. 2003. Practical voltage-scaling for fixed-priority rt systems. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 106–114.

Seo, J., Kim, T., and Chung, K.-S. 2004. Profile-based optimal intra-task voltage scheduling for hard real-time applications. In *Proceedings of the Design Automation Conference (DAC)*. IEEE Computer Society, Los Alamitos, CA, USA, 87–92.

Seo, J., Kim, T., and Chung, K.-S. 2006. Poptimal intratask dynamic voltage-scaling technique and its practical extensions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25,* 1 (Jan.), 47–57.

Seo, J., Kim, T., and Dutt, N. D. 2005. Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 450–455.

Yao, F., Demers, A., and Shenker, S. 1995. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science (FOCS)*. 374–382.

Zhang, Y. and Chakrabarty, K. 2004. Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems. *Proceedings of the conference on Design, Automation and Test in Europe (DATE) 2*, 21170.

Zhang, Y., Hu, X. S., and Chen, D. Z. 2003. Energy minimization of real-time tasks on viariable voltage processors with transition energy overhead. In *Proceedings of the 2003 Asian and South-Pacific Design Automation Conference (ASPDAC)*. 65–70.

This document is the online-only appendix to:

Transition Overhead Aware Voltage Scheduling for Fixed-Priority Real-Time Systems

BREN MOCHOCKI and XIAOBO SHARON HU
University of Notre Dame
and
GANG QUAN
University of South Carolina

ACM Transactions on Design Automation of Electronic Systems, Vol. 1, No. 1, April 2006, Pages 1–26.

A. SECTION 4 PROOFS

Lemma 1 let $\mathcal{A}$ be an arbitrary critical-interval scheduling algorithm. Further, let $\Delta t = 0$. Critical intervals identified by $\mathcal{A}$ are identified in a monotonically non-increasing order by speed.

Proof. The proof is by contradiction. Let I_i and I_j be two critical intervals with speeds S_i and S_j such that $S_i < S_j$ and $j = i + 1$. According to the definition of a critical interval, $S_i = S_{MC}(\mathcal{J})$ at iteration i and must be *continuously* applied to avoid a deadline miss (i.e., I_i can contain no idle time, otherwise a speed of zero could be applied during some portion of I_i). Because $S_i < S_j$, at least one job, J has a higher speed requirement at iteration j than at i. This is only possible if **(a)** idle time that would otherwise be used to execute a portion of J is contained in I_i, or **(b)** $S_i \neq S_{MC}(\mathcal{J})$. Both of these cases contradict the definition of a critical interval. □

Lemma 2 Let $I_{i-1} = [s_{i-1}, e_{i-1}]$ and $I_i = [s_i, e_i]$ be two consecutively identified critical intervals identified by VSLP with speeds $S_{i-1} = \mathcal{I}_{n_{i-1}}(s_{i-1}, e_{i-1})$ and $S_i = \mathcal{I}_{n_i}(s_i, e_i)$. Additionally, let the scheduled interval for iteration $i-1$ be $[s_{i-1} - \Delta t, e_{i-1} + \Delta t]$. If $S_i > S_{i-1}$ then I_{i-1} and I_i are adjacent.

Proof. To schedule $I_{i-1} = [s_{i-1}, e_{i-1}]$, the interval $[s_{i-1} - \Delta t, e_{i-1} + \Delta t]$ was reserved to account for transition time overhead. The additional $2\Delta t$ time reserved for I_{i-1} does not reduce the available execution time of jobs that do not intersect this interval, so the speed for the critical intervals containing unmodified jobs will not change according to Definition 1. Only intervals containing jobs that intersect I_{i-1} are modified, shortened by up to an additional $2\Delta t$. Thus, only intervals adjacent to I_{i-1} may experience an increase in speed in the next iteration. □

Lemma 3 Let $I_{i-1} = [s_{i-1}, e_{i-1}]$ and $I_i = [s_i, e_i]$ be two consecutively identified critical intervals identified by VSLP with speeds $S_{i-1} = \mathcal{I}_{n_{i-1}}(s_{i-1}, e_{i-1})$ and $S_i = \mathcal{I}_{n_i}(s_i, e_i)$ such that $S_i > S_{i-1}$ (i.e., a monotonicity violation has occurred). Additionally, let the scheduled interval for iteration $i-1$ be $[s_{i-1} - \Delta t, e_{i-1} + \Delta t]$. The minimum speed at which *every* job in $\mathcal{J}_{i-1} \cup \mathcal{J}_i$ can execute without a deadline miss is S_{i-1}.

PROOF. According to Lemma 2, I_{i-1} and I_i are adjacent. Therefore, if S_{i-1} is applied to both of these intervals, no voltage transition occurs. According to the definition of a critical interval, S_{i-1} is the minimum speed required to guarantee the deadlines for jobs in $\mathcal{J}_{i-1}$ and is greater than or equal to the speed required by jobs in $\mathcal{J}_i$ when no overhead is present. Because attempting to execute at a speed lower than S_{i-1} will induce a voltage transition, S_{i-1} is the minimum speed that can be used by any job in $\mathcal{J}_{i-1} \cup \mathcal{J}_i$ to meet its deadline. □

Lemma 4 Bounding the latest start time of a job set $\mathcal{J}_i$ to $r_i^{min} + \Delta t$ will prevent execution inversion. The value r_i^{min} is the earliest release time of all jobs in $\mathcal{J}_i$.

PROOF. The proof is by contradiction. Let job set $\mathcal{J}_i$ be contained in an interval $I'_i = [t_1, t_2]$ which is the minimum length interval that can complete all jobs in $\mathcal{J}_i$ at the speed S_i when t_1 is restricted to the range $[r_i^{min}, r_i^{min} + \Delta t]$. Further, let J^* be a job scheduled to execute in an adjacent critical interval I_j. Finally, assume that the execution of J^* is preempted by some job in $\mathcal{J}_i$. Because $J^* \notin \mathcal{J}_i$, this can only occur in intervals before or after I'_i. The preemption cannot occur before I'_i because cycles cannot execute during a transition interval, i.e., $[t_1 - \Delta t, t_1]$, and any time before $t_1 - \Delta t$ is also before the earliest release time, r_i^{min}. If the execution inversion occurs after I'_i, then at least one job in $\mathcal{J}_i$ was not completed in I'_i, a contradiction. □

Theorem 1 Algorithm 1 always produces a valid voltage schedule in $\mathbf{O}(N^3)$ time, given an initially schedulable job set, where N is the number of jobs.

PROOF. The correctness of Algorithm 1 follows directly from the correctness of VSLP (see [Quan and Hu 2001]), the minimum interval algorithm (see [Quan et al. 2004]), and Lemmas 2 through 4. Lines 4 and 6 each require $\mathbf{O}(N^2)$ time according to [Quan and Hu 2001] and [Quan et al. 2004] respectively. Both lines can be repeated up to $\mathbf{O}(N)$ times by the while loop at Line 3, for a total complexity of $\mathbf{O}(N(2N^2))$, or $\mathbf{O}(N^3)$. □

B. SECTION 5 PROOFS

Theorem 2 The schedule produced by *lpLDA* guarantees all system deadlines and has a computational complexity of $O(n)$ per scheduling point, where n is the number of tasks in the system.

PROOF. The speed selected by lpLDA is always greater than or equal to the speed selected by lpWDA, which always produces a valid voltage schedule when time overhead is negligible [Kim et al. 2003]. The computational complexity is on the same order of lpWDA (only a constant factor larger), which is $O(n)$ according to [Kim et al. 2003]. □

Theorem 3 Let job set $\mathcal{T}$ be schedulable under S_{max} and the transition time overhead be Δt. Assume that the processor is set to a new processor speed other than S_{max} only when the available slack is large enough to contain at least two transition overheads. Then the system lookahead $2\Delta t$ is sufficient and necessary to obtain a transition/preemption-error-free on-line schedule for an arbitrary real-time job set.

PROOF. **Necessity:** Figure 5 already illustrates that a lookahead interval less than $2\Delta t$ can lead to a preemption error. Therefore, to ensure transition/preemption-error-free for an *arbitrary* real-time job set, the lookahead interval has to be no less than $2\Delta t$.

Sufficiency: Assume that the current (pre)scheduling point is ts_i, and that the feasible speed selected at the previous scheduling point, ts_{i-1}, is S_{i-1}. We know that $|ts_{i-1} - ts_i| > 2\Delta t$ and S_{i-1} is set under the assumption that there is enough slack available at ts_i to set the processor speed to S_{i-1}, run job $JC(L_{i-1})$ at this speed, and return to a higher processor speed if necessary. We want to show that, with lookahead interval $2\Delta t$, we can always set a feasible processor speed (i.e., not higher than S_{max}) at ts_i with no transition or preemption error. We consider three possible outcomes of scheduling at ts_i: (i) $JC(L_{i-1})$ completes its execution by ts_i, (ii) Only lower priority jobs are released in L_i, and (iii) One or more higher priority jobs are released in L_i.

(i) While the slack available at ts_{i-1} is consumed by setting processor speed to S_{i-1} and also the extended execution time of $JC(L_{i-1})$, there must be at least enough slack time left for making another processor speed change. On the other hand, since ts_{i-1} is at most the pre-scheduling point of the next coming job, no transition error or preemption error will occur.

(ii) In this scenario, $JC(L_{i-1}) = JC(L_i)$. There is no need for transition since S_{i-1} ensures the feasibility of $JC(L_{i-1})$ and low priority jobs it may interfere. Therefore, there is no transition error or preemption error.

(iii) In this case, $JC(L_{i-1})$ will be preempted by $JC(L_i)$. If the feasible speed for $JC(L_i)$ is less than that for $JC(L_{i-1})$. There is no need for transition and thus no transition/preemption error. We next consider the case when $S_{i-1} < S_i$. As explained in case i, there is enough slack available at ts_{i-1} for a processor speed transition. Therefore, changing the processor speed at ts_i will not compromise the schedulability of $JC(L_{i-1})$ and the other ready jobs. On the other hand, since ts_i is a pre-scheduling point, which means that $JC(L_i)$ arrives no earlier than $ts_i + \Delta t$, the processor speed can be updated to S_i before $JC(L_i)$ arrives. Therefore there will be no transition error. Furthermore, by looking ahead $L_i = 2\Delta t$, the arrival of the next critical interval, i.e., $JC(L_{i+1})$, will be no earlier than $ts_i + 2\Delta t$. Since the processor will finish the transition to S_i at $ts_i + \Delta t$, the processor speed can still be updated to S_{i+1} in time to avoid a preemption error.

□

Theorem 4 The schedule produced by **lpLDAt** guarantees all system deadlines and has a computational complexity of $O(n)$ per scheduling point, where n is the number of tasks in the system.

PROOF. The conclusion for the schedulability guarantee comes straightforwardly from Theorem 3. Finding the critical job takes one comparison for each task in the system. The speed decision step takes $O(n)$ time for the slack estimation (because it uses lpWDA) and constant time to select the correct speed based on the slack. Calculating the limiter also takes $O(n)$ time. Hence, the overall time complexity is $O(n)$. □

Theorem 5 The policy followed by **lpLDAT** will guarantee all system deadlines, and has a computational complexity of $O(n)$ per scheduling point, where n is the number of tasks in the system.

PROOF. Theorem 4 states that lpLDAt guarantees all deadlines. Algorithm lpLDAT is identical to lpLDAt, with two key differences. First, S_{max} is conditionally set to the minimum constant speed that meets all deadlines under the worst case phasing condition off-line. If S_{max} is scaled down, then Lemma 4 still holds for lpLDAT, because the system is still schedulable at the new speed S'_{max}. The second change accounts for transition energy overhead. The modification is to only scale down to a lower speed if the energy overhead of the transition is offset by the energy gained from executing at a lower speed. Because the alternative is executing at a higher speed than the identified overhead-feasible speed, the alternative speed is also overhead-feasible. The first modification is off-line, so it does not alter the on-line complexity. The energy estimation for the second modification takes constant time, so the complexity of lpLDAT is $O(n)$. □