On-line Algorithms for Dynamic Voltage Scaling in Fixed-Priority Real-Time Systems

Abstract— **Dynamic Voltage Scaling (DVS) is an effective technique for reducing energy consumption in real-time embedded systems. On-line DVS can exploit the run-time variations in task execution to further reduce energy. We present an on-line scheduling algorithm called low power Limited Demand Analysis with Transition overhead (lpLDAT) for hard real-time systems that execute periodic, fixed-priority tasks. lpLDAT can reduce the energy consumption by as much as 40% when compared to previous algorithms in its class, and it explicitly accounts for voltage transition time and energy overhead, which previous methods do not.**

I. Introduction

With the pervasiveness of embedded and portable computing devices, such as cell-phones, PDAs and sensor/actuator networks, decreasing the power consumption and increasing system lifetime can have an enormous impact on system usability, cost and safety. Dynamic Voltage Scaling (DVS) is an effective technique to reduce power/energy consumption in such systems. A processor equipped with DVS can vary its operation voltage and frequency during runtime to take advantage of the quadratic relationship between power consumption and supply voltage of CMOS technology. When scaling, care must be taken so the delay incurred from scaling the supply voltage does not violate timing constraints. Several major IC producers have designed their modern processors with DVS capability, including AMD's Mobile Athlon [1], Intel's XScale [2] and Transmeta's Crusoe processor [3].

Real-time scheduling plays a key role in the low power design for real-time embedded systems not only because timing issues are critical but also because low power design is essentially a resource usage optimization problem for such systems. There has been substantial research for scheduling real-time applications on DVS processors, e.g., [4], [5], [6], [7], [8], [9]. These approaches differ in many aspects, such as the scheduling algorithms being on-line/off-line, handling hard/soft deadline requirements, assuming fixed/dynamic priority assignment, allowing intra-task/inter-task voltage transitions, and single/multiple processor systems.

In this paper, we focus on reducing the energy consumption for fixed-priority periodic real-time tasks executing on a single DVS processor. Many real-time embedded applications adopt a fixed-priority scheme, such as the Rate Monotonic (RM), due to its high predictability, low overhead, and ease in the implementation [10]. Specifically, we study on-line voltage scheduling techniques, while taking into consideration transition time and energy overhead. Current research shows that time overhead can be anywhere from tens of microseconds ([1], [11]) to tens of milliseconds ([12]). Because real-time systems exist that have some task deadlines of 1 or 2 milliseconds [13] it is not always possible to merge the time overhead into the worst case execution time of the tasks. Therefore, researchers are developing algorithms to handle overheads of an arbitrary size [14], [15]. While an off-line approach can fully leverage known system specifications, using off-line techniques alone may lead to a large waste of energy because, to guarantee timing constraints, one must always consider the worst case. However, the average workload of an application may vary largely from the worst case. Due to the dynamic and reactive nature of embedded systems, an on-line approach, which makes decisions during run time, is more flexible and adaptive.

Some previous research has been conducted regarding this problem, e.g., [4], [5], [6]. Pillai and Shin proposed the ccRM algorithm, which first computes off-line the maximum speed necessary to meet all task deadlines based on the worst-case response time analysis. On-line, the processor speed is scaled down when task instances complete early [6]. Although the ccRM approach guarantees job deadlines, it is not aggressive enough to fully exploit slack in the system when tasks finish closer to their best case execution times. In [4], Gruian presents a method of off-line task stretching coupled with on-line slack distribution. In addition, the paper presents an intra-task voltage scheduling method that computes the optimal speed for each execution cycle of the active task. Similar to ccRM, Gruian's off-line scheme is conservative. The intra-task method is not useful in practice because it may require the speed to change on a cycle by cycle basis. When considering transition overhead, this method is not feasible. Kim in [5] developed a method called **lpWDA** that uses a greedy, on-line algorithm to estimate the amount of slack available and then apply it to the current job. It is unique in that it takes slack from lower priority tasks, as opposed to the methods presented in [6] and [4] that wait for slack to filter down from higher priority tasks. A serious drawback is that it often too aggressive, resulting in wasted energy.

The previous methods do not consider transition overhead. A number of researchers have studied voltage scheduling when transition overhead is not negligible. Mochocki *et al.* present a method that accounts for transition overhead while scheduling a set of jobs using the Earliest Deadline First (EDF) priority scheme off-line ([14]). In [15], Saewong and Rajkumar present an algorithm to schedule fixed priority jobs sets with a very large transition time overhead off-line. AbouGhazaleh *et al.* propose an intra-task voltage scheduling method that uses compiler support and specially designed code to account for transition overhead ([16]). Hsu and Kremer also present a compiler driven DVS algorithm, but hard deadlines are not guaranteed [17]. Zhang and Chakrabarty consider all three limitations when scheduling voltage levels and checkpoint times for fault-tolerant hard real-time systems with periodic tasks ([18]). 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 for the a fault free environment.

In this paper, we present our algorithm, called low power Limited Demand Analysis with Transition overhead (lpLDAT). Through experimentation we demonstrate that combining what we call the *average case limiter* with work demand analysis results in an energy reduction of more than 40% when compared to previous methods. In addition, our algorithm takes into account both the time and energy overhead associated with a voltage transition.

The remainder of this paper is organized as follows. Section II summarizes the background material, Section III describes our algorithm, Section IV presents the experimental results and Section V concludes the paper.

II. PRELIMINARIES

In this section, we first specify the type of system under consideration and introduce the necessary notation. We then briefly review lpWDA since our new algorithm is built on top of lpWDA. A motivational example is given to show why lpWDA is not adequate in harvesting maximally the benefit provided by DVS.

A. 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 utilization over all tasks in the system. 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 similarly. Each task is invoked periodically, and 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, the number of cycles that have already 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 latest 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}, w_i^{cur}, is equal to $wc_i - ex_i^{cur}$. A *scheduling point* of J_i is a time instant that is equal to either the release time of any higher priority job in $[r_i, di]$ or d_i itself.

The DVS processor used in our system can operate at a finite set of supply voltage levels $\mathcal{V} = \{V_1, ..., V_{max}\}$, each with an associated speed. To simplify the discussion, we normalize the processor speeds by S_{max}, the speed corresponding to V_{max}, giving $\mathcal{S} = \{S_1, ..., 1\}$. Changing from one voltage level to another takes a fixed amount of time, referred to as the *transition interval* (denoted as Δt), and consumes a variable amount of *transition energy*, denoted as ΔE. This is the same as the model used in [14].

B. Low-Power Work Demand Analysis (lpWDA)

To help put our contributions in perspective, we briefly review the on-line DVS algorithm called **lpWDA**, given in Algorithms 1 and 2 (for more details on lpWDA, see [5]). For now, ignore lines marked by ***. Algorithm lpWDA works in the following manner. First, the system is initialized by setting the execution cycles and deadlines of each task and by setting the initial values of H, where H_i is an *over estimate* of the higher priority cycles that must be executed before d_i^{cur} (Lines 1–4). Next, on each preemption or completion, the remaining cycles of the preempted or completed job (w_α^{cur}) and the estimates of higher priority cycles are updated by the **updateLoadInfo** algorithm. Finally, when a job J_α^{cur} is scheduled for execution, the processor speed is scaled according to the amount of slack available (see Lines 8–10). Essentially, lpWDA takes all the slack that it can steal from lower priority tasks in linear time and applies that slack to the currently executing job. Theorem 1 guarantees that all task deadlines are met when using lpWDA. For the proof of Theorem 1, we refer the reader to [5].

Theorem 1: The schedule produced by **lpWDA** 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.

Algorithm 1 lpWDA

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

Algorithm 2 updateLoadInfo($\mathcal{T}$,α)

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

C. Motivational Example

Although lpWDA is effective in estimating the amount of slack available to the currently executing job, the simple heuristic that uses all available slack as soon as it is identified is too aggressive. Observe the two-task system in Figure 1, where the task parameters are given at the top of the figure. Figure 1(a) shows the task execution schedule when tasks always take the worst-case execution cycles and lpWDA is used. Because lpWDA is so aggressive at minimizing the speed of J_1^1, J_2^1 is forced to execute at 1.0. This situation is repeated for all instances of T_1 and T_2, resulting in a total energy consumption of 4.25 (with $Power = speed^3$). However, the schedule in Figure 1(b) also meets all deadlines and only consumes 2.47 units of energy. The situation in which lpWDA is effective is when tasks end much earlier than the worst case. For example, if every task ends exactly after the average case cycles, lpWDA follows the schedule of Figure 1(c) and consumes only 0.806 units of energy.

The difficulty with on-line scheduling is that we do not have *a priori* knowledge of the execution cycles of the upcoming jobs. However, quite often we do know both the average and worst-case execution cycles. Exploiting this added knowledge can help achieve more energy efficient schedules. In the next section, we present an algorithm which uses the slack identifying feature of lpWDA to reduce energy consumption when jobs finish early, but is also conservative enough to prevent the large energy spikes of lpWDA

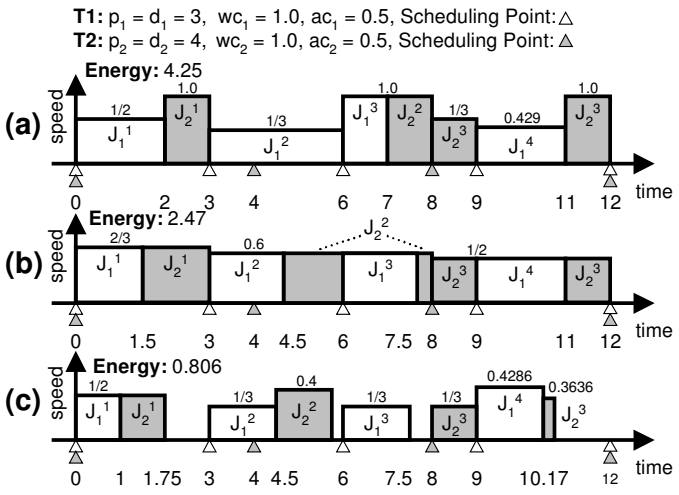

Fig. 1. An example task set consisting of two tasks (a) scheduled by lpWDA, (b) scheduled to minimize energy consumption in the worst case, and (c) scheduled by lpWDA when each job only requires the average case execution cycles.

when jobs execute near their worst-case cycles. Additionally, we account for time and energy voltage transition overhead during the scheduling process to ensure that all job deadlines are met.

III. Our Approach

In this section, we first introduce the main idea behind our approach to tapering the aggressiveness of lpWDA and discuss the modifications to lpWDA. We then describe the method used to account for transition overhead and present our overall algorithm (lpLDAT).

A. Average-Case Limiter

As we have shown in Figure 1, lpWDA is effective at exploiting the slack of lower priority jobs, but suffers later when near worst case execution cycles are required. An effective algorithm would use just enough slack given the stochastic information available about the task set under consideration.

Limiting the slack used by higher priority tasks in lpWDA requires a careful tradeoff 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. By limiter we mean that, if this speed is higher than the speed predicted by lpWDA, we know that lpWDA is being too aggressive in stealing slack from lower priority jobs and the limiting speed should be used.

In off-line voltage scheduling algorithms, an often used concept is the *minimum constant speed* that can meet all job deadlines (e.g., [6], [7], [9]). Due to the convexity of the power function, it is not generally energy efficient for the processor to go below this speed and switch to a higher speed later on, unless there is reason to expect newly available slack ([6]). Thus the minimum constant speed can serve as a proper limiter. To find the minimum constant speed for a periodic task system where every job of a task assumes the same execution cycles, one only needs to examine the case when all tasks are released simultaneously. This time instant is known as the *worst case phasing* of the task set because it represents the time that will require the maximum speed to meet all deadlines.

$$S_{MC} = \max_{i=1}^{n} \min_{t_s \in TS_i} Speed(i, t_s) \tag{2}$$

where TS_i is the set of scheduling points for J_i^1 under the worst-case phasing. The required speed to complete T_i by t_s is given in (3).

$$Speed(i, t_s) = \frac{\sum_{j=1}^{i} \lceil \frac{t_s}{p_j} \rceil \times wc_j}{ts} \tag{3}$$

Our idea is to perform a similar operation as above on-line. Directly applying the formulae in (2) and (3) 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 for the minimum speed. This reduces the time needed to calculate the limiter.

The necessary changes to lpWDA are marked by *** in Algorithms 1 and 2. We refer to this addition to lpWDA as the *average case limiter*, or just *limiter* for short. In Algorithm 1, we add Line 5 which initializes the average number of cycles that must be completed before each job deadline. Lines 6, 9 and 15 in Algorithm 2 ensure that the current phasing and execution information is stored. Line 11 in Algorithm 1 calculates the speed required by each job to meet its deadline on average. Finally, Line 12 of Algorithm 1 selects the maximum of the speeds requested by lpWDA and the limiter, essentially restricting the amount of slack that lpWDA can use. We refer to this algorithm as lpLDA. Applying lpLDA to the example task set in Figure 1 results in schedule (b) when all jobs require their worst case cycles, and the schedule in (c) when they require only their average cycles. We will show in the result section that lpLDA indeed leads to more energy saving than lpWDA. Theorem 2 states the correctness of lpLDA in terms of satisfying the real-time requirements.

Theorem 2: The voltage schedule constructed by lpLDA guarantees that all jobs are completed at or before their deadlines. The time complexity of lpLDA is $O(n)$.

Proof: Executing at the speed identified by lpWDA guarantees all deadlines according to Theorem 1. Because the limiter speed is only selected if it is *larger* than the speed identified by lpWDA, the deadline guarantee is preserved. The time complexity of lpLDA is the same as lpWDA. □

B. Voltage Transition Overhead

As pointed out in Section II, a voltage transition induces both time overhead and energy overhead. Although lpLDA is effective in further reducing energy compared with lpWDA, it cannot guarantee a feasible speed when time overhead is not negligible. Observe the schedule in Figure 1(b). If the transition from a speed of $2/3$ to 0.6 at time 3 requires one time unit, then the speed of 0.6 can no longer guarantee the deadline of J_2^2. We could try to solve this problem by reducing the slack identified in Line 12 of Algorithm 1 by one time unit. At time 3, lpLDA calls for a speed of 1/3 for job J_1^2. (This speed is identified by Line 10 of Algorithm 1 where w_α^{cur} = 1 and the estimated slack is 2, so $1/(1+2) = 1/3$.) Removing one time unit from the estimated slack will result in a speed of $1/2$, which is still less than the limiter speed of 0.6. Execution of J_1^2 will thus begin at time 4 with a speed of 0.6, and complete at time 5.5. Now we have a problem, because to meet the deadline of J_2^2 at time 8 requires a speed of $2/(8-5.5)$, or 0.8. To reach 0.8 requires another transition, and adjusting for the time overhead

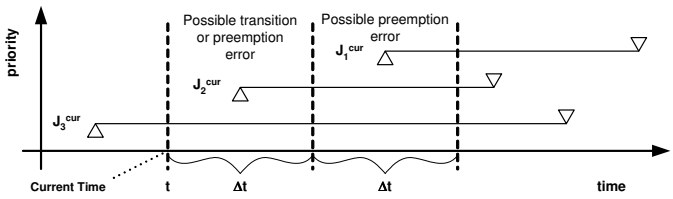

Fig. 2. An example of possible transition and preemption errors.

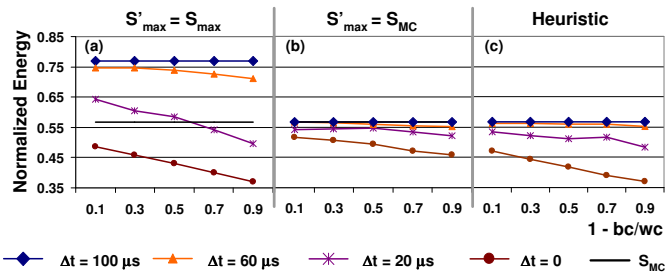

Fig. 3. The energy consumed when applying (a) lpLDAt with $S'_{max} = S_{max}$, (b) lpLDAt with $S'_{max} = S_{MC}$ and (c) lpLDAT to the task set from Figure 1.

results in a speed of $2/(8 - 5.5 - 1)$, or 1.333, which is faster than the normalized maximum of 1! Similarly, none of the existing on-line voltage scheduling algorithms can guarantee task deadlines when time overhead must be considered. Clearly, the speed selection must be more careful in the presence of time overhead to guarantee that a feasible speed is selected. Existing methods for handling transition overhead are all for off-line algorithms and thus cannot be readily used on-line. In the following we present our approach to minimize the impact of transition overhead while guaranteeing deadlines.

Transition time overhead can complicate voltage scheduling in several ways. The most straightforward effect is on slack utilization since time overhead essentially reduces available slack. To guarantee deadlines, we will always reduce the estimated slack to ensure that there is enough time to make a transition now to a speed lower than S_{max}, and also make another transition to S_{max} later when necessary. Second, a job may be released during a transition. This could happen in the task set of Figure 1 if $\Delta t > 1$. We refer to this problem as a *transition error*. The presence of a transition error may induce an unexpected voltage transition. Third, notice that lpLDA only checks the deadlines of tasks with an equal or lesser priority than the currently executing job (Line 8 of Algorithm 1). This means that we could scale down to a speed that guarantees the current job, only to be preempted later by a higher priority job that requires a higher speed. This is not a problem when transitions happen instantly, but with time overhead we must be more careful. We refer to this problem as a *preemption error*. These scenarios are illustrated graphically in Figure 2. If not handled properly, such errors could cause deadline violations.

To account for the above effects, we introduce several modifications to lpLDA. Essentially, the idea is to look ahead and predict potential future necessary transitions. How much *lookahead*, is needed deserves careful examination since too much will increase complexity and too little may not be sufficient for meeting deadlines. We achieve the proper scope as follows. First, a scheduling point exactly Δt time prior to the release time of *every* job is inserted into the schedule. These *pre-release* scheduling points ensure that higher priority jobs do not sneak up during execution without enough time to have a transition. (See the case of J_2^{cur} in Figure 2.) Second, when a scheduling point is encountered (standard or pre-release), instead of selecting the speed according to the currently executing job, we first identify the highest priority task that is released within $2\Delta t$ of the current time. Looking ahead $1\Delta t$ prevents transition errors, while looking ahead an additional Δt ensures when a transition is complete, a higher priority job that requires a larger speed is not closer than Δt away from the current time, thus preventing possible preemption errors directly after a transition (See the case of J_1^{cur} in Figure 2).

We denote the algorithm that includes the above modifications to lpLDA to guaranteeing deadlines in the presence of time overhead as lpLDAt, and Theorem 3 provides the deadline guarantee. The proof is omitted due to the page limit.

Theorem 3: The voltage schedule constructed by lpLDAt guarantees that all jobs are completed at or before their deadlines. The time complexity of lpLDAt is $O(n)$.

Another challenge in dealing with time overhead is to reduce the increase in energy consumption due to time overhead. To show how time overhead can lead to undesirable consequence in DVS, let us examine the data in Figure 3(a). Figure 3 (a) 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 bc/wc of each task. (Note that all times from Figure 1 are multiplied by 100 for the simulation in Figure 3.) The black line represents the energy consumed when executing at the minimum constant speed, S_{MC}, without DVS. All energy numbers are normalized against the energy consumed when executing at S_{max} without DVS. The reader will immediately notice that as Δt grows, the benefit gained from DVS quickly vanishes. Such results are due to the fact that lpLDAt uses S_{max} for workload estimation and slack computation, which can adversely 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 scaling back the maximum speed used by lpLDAt when predicting how much slack is available. Intuitively, this will allow less aggressive slack exploitation and thus avoid transitions that would increase energy instead of reducing energy. The maximum speed, S_{max} (which is assumed to equal 1 when normalized) is used implicitly in Lines 9 and 10 of Algorithm 1 when determining f_{clk}. That is, the normalized maximum of 1.0 is divided into the workload values, resulting in equivalent time values. If we choose a lower speed for S_{max}, it can be readily used to scale these workload values and can effectively scale down the maximum system speed. We refer to the adjusted maximum speed as S'_{max}.

An interesting problem is how to select this S'_{max}. One may be tempted to select S_{MC} as the speed for S'_{max}. Though this ensures that no curves will appear above the S_{MC} line in Figure 3, and also guarantees all task deadlines, doing so would drastically reduce the amount of slack available when Δt is zero and jobs finish much earlier than the worst case. This situation is illustrated in Figure 3(b). Setting S'_{max} to a speed between S_{MC} and S_{max} is not efficient in general, because the reason that we scale back the max speed is that we expect there to be very little slack available (due to near worst-case cycles or high Δt). Because there is little slack, we expect to execute at S'_{max} a large percentage of the time and if that is the case, the lower S'_{max} is the less energy the system will consume. Through experimental study, we have observed that when Δt is zero, it is always better to execute with $S'_{max} = S_{max}$, and when $\Delta t \geq 10\%$ of the minimum deadline in the system, it is generally better to set $S'_{max} = S_{MC}$. When Δt is between these two values, we

must use a heuristic that selects either S_{max} or S_{MC}. We give this heuristic in the detailed algorithm description below. The result of our S'_{max} scaling heuristic is given in Figure 3(c). Notice that no curve exceeds S_{MC}, and the curve with $\Delta t = 0$ retains the energy savings of lpLDAt from Figure 3(a).

Another issue is how to deal with transition energy overhead. Because energy overhead cannot cause deadline misses, the only concern is reducing its impact on energy consumption. This introduces a relatively minor modification to lpLDAt.

In Algorithm 3, we present all the modifications to lpLDA. We refer to this new algorithm as lpLDAT. Here we will focus on the parts that are different from lpLDA. Lines 3–6 determines the adjusted maximum speed S'_{max}. Essentially, we track the ratio of Δt to the minimum deadline of the task set and the ratio of the average-case utilization to the worst-case utilization. If the latter is smaller, we choose the maximum processor speed since in this case the processor tends to have more slacks and more aggressive stealing of slack is acceptable. Otherwise, we set the maximum speed to S_{MC}. At Line 7, we initiate the computation of the system speed not only at job completion and preemption points as is done by lpLDA but also the pre-release scheduling point as discussed above to prevent transition and preemption errors. In Line 10, we look ahead $2\Delta t$ time to prevent preemption errors as discussed above. Lines 12–15 compute all the possible choices of new speeds and select the one that gives the minimum energy. 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 17), which ensures that ΔE doesn't locally dominate the energy saved by changing the 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. The following theorem states the correctness of Algorithm 3. We omit the proof due to the page limit.

Theorem 4: Executing at the speed identified by lpLDAT at each scheduling point will guarantee all deadlines for any arbitrarily large time transition overhead. The time complexity of lpDAT is $O(n)$.

Algorithm 3 lpLDAT

1: **if** system start **then**
2: Initialize each task as Algorithm 1 with ***;
3: $\Delta tRatio := max\{0, \frac{0.1 \times minDeadline - \Delta t}{0.1 \times minDeadline}\}$;
4: $utilRatio := \frac{U_{ac}}{U_{wc}}$;
5: **if** $utilRatio < \Delta tRatio$ **then** $S'_{max} := S_{max}$;
6: **else** $S'_{max} := S_{MC}$;
7: **if** job completion/preemption/pre-release **then**
8: Find T_α, the currently executing task;
9: **updateLoadInfo**($\mathcal{T}$,α);
10: $T_\alpha :=$ the highest priority incomplete task released before $t + 2\Delta t$;
11: $t_s := max\{t, r_\alpha^{cur}\}$;
12: Compute $slack_\alpha$ based on workload starting at t_s;
13: **if** there is enough time for 2 transitions **then** $f_{clk} :=$ the new speed;
14: **else if** there is enough time for one transition **then** f_{clk} stays the same;
15: **else** $f_{clk} := S'_{max}$;
16: **if** $f_{clk} < f_{limit}$ **then** $f_{clk} := f_{limit}$;
17: **if** $f_{clk} <$ the previous speed **then check_energy_overhead**();
18: Set the voltage according to f_{clk};

IV. Experimental Results

In this section we quantify the effectiveness of lpLDAT on several real-world and randomly generated task sets, and compare its energy consumption with ccRM and lpWDA. Both ccRM and lpWDA were modified to account for time and energy transition overhead; lpWDA with exactly the same method as lpLDAT, excluding the scaling of S_{max} to S'_{max} and ccRM with a very similar method, e.g., it only scales down when there is enough time to transition back up later. We also compare with an algorithm we call MIN which we use as a reference lower bound to all three algorithms. MIN operates similar to the average-case limiter described in the previous section with three key differences: First, it uses the exact execution cycle information for every task when computing the best end time for a particular job. Second, every scheduling point is checked when looking for the best end time, not just the job's deadline. Finally, Δt is always considered zero. Clearly MIN is not applicable in practice and cannot find the minimum energy schedule in general. However, it is a lower bound for the algorithms we present here.

The processor model we use is representative of the ARM8 core. For all experiments we assume there are 32 frequency levels available in the range of 10 to 100 MHz, with corresponding voltage levels of 1 to 3.3 Volts. When idling, the processor is assumed to consume one half the power consumed when executing at the minimum processor speed. Transition energy overhead is modeled using $\Delta E = \eta \times C_{DD} \times |V_1^2 - V_2^2|$, with $\eta = 0.9$ and $C_{DD} = 5$ μF as presented by Burd in [11]. The energy of all the results presented in this section are normalized against a processor running at S_{max} without DVS.

The first set of experiments was conducted on randomly generated task sets with the number of tasks per set varied from 2 to 10 in two task increments. Each grouping has 100 separate task sets, with periods and deadlines uniformly distributed in the range [1, 100] ms, hyper periods less than or equal to 5 s, and a U_{wc} normally distributed in the range [0.2, 0.8]. Additionally the S_{MC} speed of all task sets is less than or equal to the normalized maximum of 1. Δt is assumed to be 100 μs. The results are given in Figure 4. Clearly lpLDAT outperforms the other two algorithms in all cases, and the margin of its improvement increases with the number of jobs in the system and as the best case execution cycle to worst-case execution cycle ratio (bc/wc) decreases. The improvement of lpLDAT compared to ccRM varies between about 2% with 2 tasks and a bc/wc ratio of 1 to over 40% with 10 tasks and a bc/wc ratio of 0.1. The improvement over lpWDA varies from less than 1% with 2 tasks and a bc/wc ratio of 0.1 to as much as 60% with 10 tasks and a bc/wc ratio of 0.1.

Next, each algorithm was applied to two real-world examples: A Computerized Numeric Controller (CNC) task set based on the work by Kim *et al.* in [19] and an avionics task set based on Locke's work in [13]. For each task set, Δt was varied from 0 to 180 μs in 60 μs steps. The results are displayed in Figures 5(a–d) and 5(e–h) respectively. MIN was also applied to offer a lower bound on energy consumption. For the CNC tasks set, lpLDAT is always as good as or better than ccRM and lpWDA. Note that instead of displaying $\frac{bc}{wc}$ on the x-axis, we display $1 - \frac{bc}{wc}$. With $\Delta t = 0$, lpLDAT is within 2% of the lower bound and consumes as much as 40% less energy than ccRM or lpWDA. As Δt increases, lpLDAT eventually saturates to ccRM, due to the scaling of S'_{max}. The avionics task set is different because ccRM doesn't outperform lpWDA in this case. The reason is that the normalized S_{MC} for the avionics set is 0.97, or very close to the maximum speed. In this case the aggressive nature of lpWDA wins out. However, lpLDAT still outperforms lpWDA by as much as 20%.

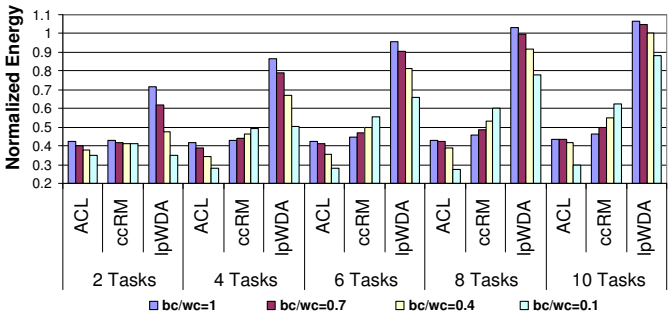

Fig. 4. Energy consumption of lpLDAT, ccRM and lpWDA when scheduling randomly generated task sets. In each case the transition time overhead is is set to 100 μs.

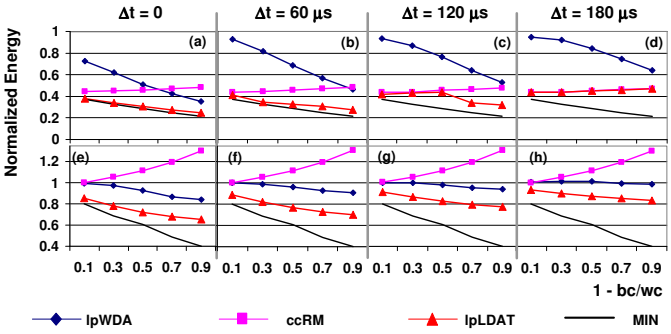

Fig. 5. Energy consumption of lpLDAT, ccRM and lpWDA on the CNC (a–d) and Avionics (e–h) task sets with various amounts of transition time overhead.

V. Summary

In this paper we presented a new algorithm called low power Limited Demand Analysis with Transition overhead (lpLDAT) that leverages the advantages of lpWDA by Kim in [5] while at the same time removing the inherent disadvantages of lpWDA. Additionally, time and energy transition overhead are accounted for in an intelligent manner that guarantees system deadlines, while at the same time keeping energy consumption down to a reasonable level. lpLDAT outperforms previous on-line DVS algorithms by as much as 40%.

Although lpLDAT does perform well, it is not optimal. The heuristic used to deal with time overhead is pessimistic in that it assumes no jobs can execute before the highest priority job is released, up to $2\Delta t$ from the current time, which is rarely the case. Additionally, lpLDAT could benefit from a more sophisticated off-line analysis of the task set, especially if more is information is known about each task. One example is the probability density of each task's execution cycles. Future work should conduct an in depth analysis of these issues.

References

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

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

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

[4] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design (ISPLED)*. New York, NY: IEEE, Aug. 2001, pp. 46–51.

[5] W. Kim, J. Kim, and S. L. Min, "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)*. New York, NY: ACM Press, Aug. 2003, pp. 396–401.

[6] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP)*. New York, NY: ACM Press, 2001, pp. 89–102.

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

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

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

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

[11] T. D. Burd, "Energy-efficient processor system design," Ph.D. dissertation, University of California, Berkeley, Berkeley, CA, May 2001.

[12] "Compaq ipaq h3600 hardware design specification - version 0.2f," online- http: //www.handhelds.org/ Compaq/ iPAQH3600/ iPAQ_H3600.html, Compaq Computer Corporation.

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

[14] B. Mochocki, X. S. Hu, and G. Quan, "A realistic variable voltage scheduling model for real-time applications," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-Aided design (ICCAD)*. New York, NY: ACM Press, Nov. 2002, pp. 726–731.

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

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

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

[18] Y. Zhang and K. Chakrabarty, "Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE)*. Washington DC: IEEE Computer Society, 2004, p. 21170.

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