Voltage-Clock-Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems

C. M. Krishna
Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
krishna@ecs.umass.edu

Yann-Hang Lee
Computer and Information Science
University of Florida
Gainesville, FL 32608
yhlee@cise.ufl.edu

Abstract

Many embedded systems operate under severe power and energy constraints. Voltage clock scaling is one mechanism by which energy consumption may be reduced: it is based on the fact that power consumption is a quadratic function of the voltage, while the speed is a linear function. In this paper, we show how voltage scaling can be scheduled to reduce energy usage while still meeting real-time deadlines.

1. Introduction

Many applications impose severe power and/or energy constraints on embedded systems. Examples include battery-powered devices and spacecraft relying on solar or nuclear power.

Voltage control is a powerful mechanism for reducing the energy consumption: the power consumption declines as the square of the voltage, while circuit delays increase linearly. Since the clock frequency is proportional to the inverse of the circuit delay, we have an obvious tradeoff between the power consumed and the speed of the circuit.

The idea of exploiting this tradeoff has attracted increasing attention since the first paper was published in 1994 [6]. In [1], a circuit may choose from among multiple voltage levels to reduce power consumption while satisfying latency constraints. In [8], the Dhrystone 1.1 benchmarks were run on an ARM7D processor at two voltage-frequency combinations: (5.0V, 33 MHz) and (3.3V, 20 MHz) yielding 185 MIPS/watt and 579 MIPS/watt, respectively. Yao *et al.* derived a voltage-control heuristic to reduce energy consumption, assuming that the power usage is a convex function of the clock rate [7]. A benchmark suite and simulation environment for voltage scaling are presented in [5].

In this paper, we focus on hard real-time systems, where meeting critical task deadlines is of paramount importance [4]. Such systems are to be found in, for example, fly-by-wire aircraft and spacecraft. We show how to schedule voltage settings so that energy consumption is reduced, while still guaranteeing that all task deadlines are met. Our algorithms consist of an *offline* phase, in which voltage settings are picked to reduce energy consumption assuming that tasks run to their worst-case execution times (WCET). However, many tasks finish well before their WCET, and we have an *online* phase which adjusts the voltage settings on-the-fly to reclaim any resources released by such tasks. Our numerical results indicate that substantial energy savings are attained.

The paper is organized as follows. In Section 2, we outline our system model. This is followed in Section 3 by a scheduling algorithm which works for the case where the tasks have a common period. In Section 4, arbitrary task periods are allowed, and a somewhat more complex algorithm is used. We also show how this algorithm handles task sets whose phasings are not known until run time. The paper concludes with a brief discussion in Section 5.

2. System Model

Most real-time systems used in critical embedded applications use periodic workloads. That is, each task, T_i, has a period, P_i, and an iteration of T_i is released each P_i time units. The deadline of a task is equal to the period. That is, a task iteration must be done by the time the next iteration of that task is released. The worst-case execution time of each task is assumed to be known.

There is a huge literature on the problem of allocation and scheduling of tasks in real-time systems; for a survey, see [4]. The typical approach is to carry out an allocation of tasks to processors and then to run a uniprocessor scheduling algorithm on each of the processors to decide when each task will execute.

In this paper, we focus on the problem of uniprocessor scheduling. The task-scheduling algorithms are Cyclic and Earliest Deadline First (EDF). Under a cyclic schedule, a subset of tasks will be selected for execution in a minor frame while a set of minor frames iterate periodically in a major frame [2]. As the term implies, EDF picks the task to run whose deadline is the earliest among all the ready tasks. Ties are broken arbitrarily. The EDF algorithm is preemptive, and it is assumed that

preemption costs are negligible compared to the task run times. It can be shown that under such assumptions, and for task sets whose deadlines equal their respective periods, EDF is an optimal uniprocessor scheduling algorithm. That is, if EDF cannot feasibly schedule some task set, no other algorithm can, either. A periodic task set with the deadline of each task equal to its period is EDF-schedulable iff the task set utilization does not exceed 1; this is a lightweight schedulability test for the EDF algorithm. Note that, for a cyclic schedule, it is feasible if the task set utilization during every minor frame does not exceed 1.

Our other assumptions are as follows:

A1 Voltage switching consumes negligible overhead.

A2 There is a time-of-day clock available to the system, with sufficient precision to time-stamp the completion of tasks and other significant events.

A3 Tasks are independent: no task depends on the output of any other task.

A4 The worst-case execution time of each task T_i, ω_i, is known. The actual execution time is not known, however, and may vary from one iteration to the next: it is a random variable with distribution $G_i(\cdot)$.

A5 The overhead of the scheduling algorithm is negligible when compared to the execution time of the application workload.

We now present two algorithms. The first deals with the tasks scheduled in a minor cycle under a cyclic algorithm. The second algorithm focuses in EDF algorithms.

3. Algorithm 1: Cyclic Scheduling

For each minor frame, all scheduled tasks are released at the beginning of the frame and must finish by the end of that frame. We assume that the tasks have a predefined order of execution.

3.1 Algorithm Description

The algorithm consists of two phases. In the offline (or pre-processing) phase, so-called because it is executed before the system is actually used, we simulate the task execution, using the worst-case execution times. The purpose of the offline phase is to come up with a labelling of each task as either a *high-voltage* (hv) or a *low-voltage* (lv) task. A task labelled hv (lv) will have all of its iterations executed at high voltage (low voltage). The offline phase finds, by a standard search algorithm, the labellings that minimize the total energy used over a task period, subject to the need to meet all deadlines. We record, for these labellings (i.e., voltage settings), the time at which each task executes. This information, together with the lv and hv labellings, allow us at any time to compute the total unfinished work remaining in the system at any time, t. Denote this unfinished work by $offline\_unf(t)$. Similarly, we can obtain the voltage setting at time t: denote this by $offline\_setting(t)$.

Let us now consider the online phase of the algorithm, i.e., the scheduling algorithm that is used during actual execution. The system keeps track of the worst-case unfinished work remaining in the system (i.e., total unfinished work assuming each unfinished task takes its worst-case execution time). Denote this by $online\_unf(t)$.

The online scheduling algorithm is as follows.

- The task to be executed is chosen based on the pre-defined order.
- At any time t, the processor is set at low voltage, unless each of the following conditions is satisfied:
  - $online\_unf(t) = offline\_unf(t)$.
  - The voltage setting of the processor in the offline phase at time t is *high*. (This does not have to be stored separately: an examination of the slope of the $offline\_unf(t)$ line at t provides this information).

Note that the algorithm does not need to keep track of whether the $online\_unf(t) < offline\_unf(t)$ condition is satisfied for every clock cycle (that would be impossible). Instead, when a processor takes up a task, it checks this condition. If the condition is true, and the low-voltage setting is used, the system computes the time τ at which $online\_unf(t) = offline\_unf(t)$. This can be done easily, since we know the rate of execution at high and low voltages, as also the rate at which $offline\_unf(t)$ declines with t. If the task is still executing at τ, and the voltage setting at that instant in the offline phase is *high*, the processor is switched at that epoch, to high voltage.

It is not difficult to show that the scheduling algorithm does not miss any deadlines if the original task set is feasible: the proof of the following statements is left to the reader.

Lemma 1 $online\_unf(t) \leq offline\_unf(t)$, *for all* t.

Theorem 1 *No deadlines are missed by the online algorithm.*

3.1.1 Example

We illustrate this algorithm by walking through a single simulation. The task set consists of three tasks, T_0, T_1, T_2, all with period 10, and with worst-case high-voltage execution times 1.933, 3.678, and 1.888, respectively. If all tasks are run at high voltage, the processor utilization would be 0.75. Let the actual task high-voltage execution times be uniformly distributed over the respective intervals: [WCET/2, WCET], and assume that the system works 50% slower at low voltage.

The offline algorithm determines that the best offline voltage assignment is *low* for tasks T_0 and T_2, and *high* for T_1. If tasks take their worst-case execution times, the processor utilization is 0.941. The reader should note how close this is to 1, which assures us that this simple algorithm would be close to optimal if all tasks consume their worst-case execution times.

By simulating this algorithm, it is easy to obtain $offline\_unf(t)$ (see Figure 1). $offline\_unf(t)$ consists of a set of straight-line segments: by storing the endpoints of these segments, the value of the function at any t can quickly be computed. The offline part of the algorithm is now over.

Consider now the operation of the online part. Suppose, in an execution, the actual high-voltage execution times of T_0, T_1, T_2 were $1.53, 2.57, 1.87$, respectively. The corresponding low-voltage times are $2.30, 3.86, 2.80$, respectively. Since the processor is not an oracle, it cannot know these execution times until after the respective tasks have completed execution. (This means that $online\_unf(t)$ has potential downward jumps at the epochs of task completion). Task T_0 starts executing, at low voltage, and completes at time 2.30. At this time, the algorithm knows that $online\_unf(t) < offline\_unf(t)$ for $t = 2.30$. As a result, it can execute T_1 at low voltage up to time τ, at which $offline\_unf(\tau) = offline\_unf(\tau)$. It is easy to see that $\tau = 4.10$. At that instant, the system switches from low to high voltage, and completes T_1 at time 5.47. At this time, T_2 can be run at low voltage to its own completion, at time 8.27. The execution trajectory is shown in Figure 1.

3.2 Performance Model

In this section, we derive models to compute the energy savings for our algorithm. First, we present a simple fluid approximation that provides us with a lower bound on the energy consumed. Then, we present a more exact analysis for a system consisting of a finite number of tasks.

We start by defining some notation. Note that all workloads are defined by the time taken to execute them at high voltage.

$\sigma_A^{off}(t)$	Voltage setting at time t, specified by the offline phase, for task set A.
$T_{off}^{worst}(A)$	Total execution time for task set A, under the schedule developed by the offline algorithm, if all tasks run to their worst-case execution times.
$T_{on}^{actual}(A)$	Total execution time for task set A, under the online algorithm.
$W^{worst}(A)$	Total workload due to task set A if all tasks run to their worst-case times.
$W^{actual}(A)$	Actual total workload due to task set A.
$E_{off}^{actual}(A), E_{off}^{worst}(A)$	Actual and worst-case energy consumed, respectively, if the tasks in set A run to the settings prescribed by the offline algorithm.
$E_{on}^{actual}(A), E_{on}^{worst}(A)$	Actual and worst-case energy consumed by tasks in set A.
$U_H^{worst}(A)$	Worst-case processor utilization if all tasks in task set A are run at high voltage.
P	Common period of all the tasks.
λ	$\frac{\text{Power consumption at low voltage}}{\text{Power consumption at high voltage}}$
ϕ	$\frac{\text{Clock rate at high voltage}}{\text{Clock rate at low voltage}}$

Where the task set is obvious from the context, no argument is provided to functions. For example, if we are talking about just one task set, U_H^{worst} would be the worst-case processor utilization for that task set.

Throughout, we assume that $\lambda/\phi < 1$; otherwise, there would be no point in running anything at low voltage!

3.2.1 Fluid Approximation

In this model, we assume that the workload consists of tasks whose execution times are independent and identically distributed, with worst-case execution time (at high voltage), μ. The number of tasks, $n_{tasks} \to \infty$ and $\mu \to 0$ in such a way that the total worst-case workload, $n_{tasks}\mu = W$, a constant. $U_H^{worst} = W/P$, where U_H^{worst} is the worst-case processor utilization if all tasks are run at high voltage. Let W^{actual} be the actual total workload.

If all tasks run to their worst-case execution times

and we use the schedule and voltage settings generated by the offline phase of the algorithm, the total busy time for the processor over the period is given by

$$T_{off}^{worst} = \min\{W^{worst}\phi, P\}.$$

Since we have an infinite number of tasks, with probability 1, the online algorithm will consume

$$T_{on}^{actual} = \min\{W^{actual}\phi, T_{off}^{worst}\}.$$

We now have two cases.

Case 1: $W^{actual}\phi \leq T_{off}^{worst}$: In this case, with probability 1, the online algorithm will keep the entire workload at low voltage, and so the energy consumed will be $\mathcal{E} = 100\lambda\phi\%$ of that at high voltage.

Case 2: $W^{actual}\phi > T_{off}^{worst}$: Some of the workload will have to be run at high voltage. Let t_h and t_ℓ be the time over which the processor is run at high and low voltage, respectively. Clearly, $t_h + t_\ell = T_{off}^{worst} = P$ when $W^{actual}\phi > T_{off}^{worst}$.

The total workload is W^{actual} seconds at high voltage. Since the processor runs ϕ times slower at low voltage, we must have

$$\begin{aligned} W^{actual} &= t_\ell/\phi + t_h \\ \Rightarrow t_\ell &= \frac{\phi}{\phi - 1}(P - W^{actual}) \text{ since } t_h + t_\ell = P \end{aligned}$$

The energy consumed, as a percentage of the all-high-voltage setting, is therefore given by

$$\begin{aligned} \mathcal{E} &= \frac{\lambda t_\ell + t_h}{W^{actual}} \times 100 \\ &= \frac{(\lambda\phi - 1)P + \phi(1-\lambda)W^{actual}}{\phi - 1} \times 100 \end{aligned}$$

This analysis provides us with a lower bound to the energy consumed under this algorithm for a total actual workload of W^{actual}. We now prove that this is the case.

Lemma 2 *Let A and B be two task sets such that* $W^{worst}(A) = W^{worst}(B)$ *and* $W^{actual}(A) = W^{actual}(B)$. *A has a finite number of tasks; B follows the infinite-task model. Then,*

$$T_{off}^{worst}(A) \leq T_{off}^{worst}(B).$$

Proof: We proceed by contradiction. Suppose the lemma is false, and there do exist task sets A and B which constitute a counter-example.

The offline scheduling algorithm picks the voltage settings per task to minimize the energy consumed, under the constraint that the entire task has to be executed at the prescribed voltage setting (e.g., one cannot execute half a task at *high* and the other half at *low* settings).

If $T_{off}^{worst}(A) > T_{off}^{worst}(B)$, then $E_{off}^{worst}(A) < E_{off}^{worst}(B)$ (since $W^{worst}(A) = W^{worst}(B)$).

Now, suppose we run task set B using setting $\sigma_A(t)$ at time t. We can do this because the tasks in B are infinitely short, and so the voltage setting can be switched at any time by the offline algorithm.

This will result in task set B taking exactly the same worst-case execution time as task set A. In such a case, B will use less energy than it did under the $\sigma_B(t)$ offline voltage setting. This contradicts the fact that the offline algorithm picks voltage settings to minimize the energy consumed. **QED**

Theorem 2 *Suppose task sets A and B are as defined in Lemma 2. Then,* $E_{on}^{actual}(A) \geq E_{on}^{actual}(B)$.

Proof: From the algorithm, $T_{on}^{actual}(A) \leq \min\{W^{actual}(A)\phi, T_{off}^{worst}(A)\}$ and $T_{on}^{actual}(B) = \min\{W^{actual}(B)\phi, T_{off}^{worst}(A)\}$. Since $W^{actual}(A) = W^{actual}(B)$ and $T_{off}^{worst}(A) \leq T_{off}^{worst}(B)$ (from Lemma 3), $E_{on}^{actual}(A) \geq E_{on}^{actual}(B)$. **QED**

3.2.2. Relaxing the Infinite-Task Assumption

Relaxing the infinite-task assumption complicates the analysis. We would then have a model with a finite number of tasks, each with a certain execution time distribution. One can construct a model which can then be solved numerically

The most practical approach is to construct, given the worst-case execution times, the offline schedule. This yields us a plot of the unfinished work over time that is used as a template by the online algorithm. Then, one conditions on the actual execution times of the online tasks: given these times, a numerical evaluation is carried out to determine the voltage settings over time for the online phase, and thus compute the energy consumed. That is, if F_i is the actual execution time of task T_i, we will obtain $E(F_1, \cdots, F_{n_{tasks}})$, the energy consumed under these conditions.

Then, we uncondition on the actual execution times, obtaining the overall energy consumed as:

$$\int_{F_1=0}^{\Omega_1} \cdots \int_{F_{n_{tasks}}=0}^{\Omega_{n_{tasks}}} E(F_1, \cdots, F_{n_{tasks}}) dG_{n_{tasks}}(F_{n_{tasks}}) \cdots dG_1(F_1)$$

This integral can be evaluated either through numerical means or simulation.

It only remains for us to show how to derive $E(F_1, \cdots, F_{n_{tasks}})$.

We will need some further notation for this.

$\Upsilon(i)$ slack time available at the end of the task T_i execution, $i = 1, \cdots, n_{tasks}$. That is, if task T_i ends at time τ_i, the slack is the time remaining to when T_i completes in the offline schedule, i.e., $\Upsilon_i = \Omega_1 + \cdots + \Omega_i - \tau_i$. For convenience, define $\Upsilon_0 = 0$.

Π_{low} Power consumption at low voltage

Π_{high} Power consumption at high voltage

Φ_i $\begin{cases} 1 & \text{if offline voltage setting is } \textit{low} \text{ for } T_i \\ \phi & \text{otherwise} \end{cases}$

τ_i Finishing time of T_i in the online schedule, $i = 1, \cdots, n_{tasks}$. For convenience, define $\tau_0 = 0$.

ε_i Energy consumed by T_i in the online schedule.

We have two cases.

Case 1: Offline setting of task T_i is *low*: In such a case, task T_i will also be executed at low voltage in the online schedule. Then we can immediately write:

$$\begin{aligned} \varepsilon_i &= F_i \phi \Pi_{low} \\ \tau_i &= \tau_{i-1} + F_i \phi \\ \Upsilon_i &= \sum_{i=1}^{i} \Omega_i \Phi_i - \tau_i \end{aligned}$$

Case 2: Offline setting of task T_i is *high*: The total time available to execute T_i is $\Omega_i + \Upsilon_{i-1}$. Since the task may involve up to Ω_i units of high-voltage work, we have to compute how much of the task can safely be done at low-voltage and still leave enough time for it to be completed, even if it runs to its worst-case time. Let ξ_0 be the maximum time that it can be run at low voltage without being in danger of missing its deadline. Now, define ξ_1 to satisfy the following equations:

$$\begin{aligned} \xi_0/\phi + \xi_1 &= \Omega_i \\ \xi_0 + \xi_1 &= \Omega_i + \Upsilon_{i-1} \end{aligned}$$

From these equations, the reader can easily recognise that $\max\{\xi_1, 0\}$ is the time available for high-voltage execution, should that prove necessary.

Solving these equations yields:

$$\begin{aligned} \xi_0 &= \Upsilon_{i-1} \frac{\phi}{\phi - 1} \\ \xi_1 &= \Omega_i - \frac{\Upsilon_{i-1}}{\phi - 1} \end{aligned}$$

We will run T_i for up to ξ_0 at low voltage: if it still hasn't finished, we will run it to completion at high voltage. We have two subcases:

Case 2a. $F_i \leq \xi_0/\phi$: In this case, the entire execution of T_i can be done at low voltage. We can therefore write:

$$\begin{aligned} \tau_i &= \tau_{i-1} + F_i \phi \\ \varepsilon_i &= F_i \phi \Pi_{low} \\ \Upsilon_i &= \Omega_1 + \cdots + \Omega_i - \tau_i \end{aligned}$$

Case 2b. $F_i > \xi_0/\phi$: In such a case, we first execute T_i for ξ_0 seconds at low voltage, and then switch to high voltage for the rest of the execution. We therefore have:

$$\begin{aligned} \tau_i &= \tau_{i-1} + \xi_0 + F_i - \xi_0/\phi \\ &= \tau_{i-1} + \xi_0 \frac{\phi - 1}{\phi} + F_i \\ \varepsilon_i &= \xi_0 \Pi_{low} + (F_i - \xi_0/\phi) \Pi_{high} \\ \Upsilon_i &= \Omega_1 + \cdots + \Omega_i - \tau_i \end{aligned}$$

The total energy consumed is then given by

$$E(F_1, \cdots, F_{n_{tasks}}) = \varepsilon_1 + \cdots \varepsilon_{n_{tasks}}$$

3.3 Simulation Results

We present here results of a simulation written from first principles. In our experiments, we assumed that at high voltage, the power consumption was 0.165 watts and at low voltage, it was 0.033 watts. The clock rate at high voltage is 50% higher than at low voltage. All execution times are specified in terms of the high-voltage setting. We assumed that the actual execution time of task T_i varies uniformly in the interval $[a\omega_i, \omega_i]$, where a is a constant and ω_i is the worst-case execution time of T_i. The common period was set to 10.

Given the processor utilization at high-voltage (i.e., the utilization if all the workload was executed at high voltage), the task execution times were generated randomly to meet this requirement.

Figure 2 shows the energy consumption for an 8-task system for various processor utilizations at high voltage, U_H. By "percentage online consumption" we mean the energy consumed by the online algorithm as a percentage of the consumption of the processor if everything were run at high voltage.

Table 1 shows some experimental results on the average processor utilization, U, that would result from using the voltage settings generated by the offline algorithm, if each task ran to its worst-case time. Except when all tasks can be run at low voltage, U is extremely close to 1 even for small task sets. U is a measure of how close the offline algorithm is to optimal: in the optimal case, we would have the processor utilized 100% at worst-case execution times.

Figure 3a is a plot of the energy consumption of the online schedule as a percentage of that obtained by using just the settings of the offline phase of the algorithm. It indicates the gains that are possible when the scheduler reclaims resources after a task has completed before its worst-case execution time would predict. When the utilization of the task set is small, everything can be executed at low voltage, and there is nothing to be gained from the online phase. As the utilization increases beyond this region, the savings of the online phase steadily increase. Resource reclaiming is greatest when $a = 0$, and decreases as a increases. Clearly, when $a = 1$, there is no resource reclaiming possible and the online energy consumption is the same as that using just the offline settings.

We next consider the impact of the size of the task set. As the number of tasks increases, two things happen. First, the offline algorithm has more flexibility in making its powersettings, and consequently is able to get the worst-caseprocessor utilization with its power settings closer to 1. We have alrady seen this in Table 1. Also, the resource reclaiming opportunities increase with the number of tasks. (To take an extreme example, if the entire task set consists of just one task, there can be no reclaiming. If it consists of two tasks, the reclaimed time from just one task can be used.) As a result, the online energy consumption as a percentage of the corresponding offline energy consumption decreases with the number of tasks. This is shown in Figure 3(b).

4. Algorithm 2: EDF Scheduling

In this section, we discuss voltage-clock scheduling for the EDF algorithm. In addition to Assumptions A1 to A5, we have:

A6 Task phasings are known in advance.

This extra assumption can be relaxed as we show at the end of the section.

Algorithm 2 is very similar to Algorithm 1, except in the data that are collected. It consists of offline and online parts.

The offline part consists of selecting the voltage settings that will minimize the total energy used over the LCM of the periods, while still maintaining EDF-schedulability. Following this, the schedule, using the EDF algorithm and the worst-case execution times, is generated, and the functions $\mathrm{offline\_unf}(i,t)$ are computed. $\mathrm{offline\_unf}(i,t)$ denotes the unfinished work under the offline schedule of task i at time t. $\mathrm{offline\_unf}(i,t)$ consists of straigh t-linesegments for each task i, and so only the end-points of these segments must be stored. Also stored is $\mathrm{offline\_task}(t)$, which is the task which is executing at time t. When these functions have been obtained up to the LCM of the task periods, the offline phase ends.

The online part also uses the EDF scheduling algorithm. At any time t, the voltage setting is at *low* unless each of the following conditions is satisfied (i is the online executing task):

- $i = \mathrm{offline\_task}(t)$.
- The unfinished work of the executing task (based on the worst-case execution times) at time t is equal to that of $\mathrm{offline\_unf}(i,t)$.
- $\mathrm{offline\_setting}(i) =$ *high*.

If each of these conditions is true, the voltage setting is *high* at time t.

4.1 Proof of Correctness

Denote the online executing task at time t by $\mathrm{online\_task}(t)$. Define $\mathrm{iter}_i(t) = \lfloor t/P_i \rfloor$, where P_i is the period of task T_i. Define $T'_{i,m}$ as the m' iteration of task T_i.

Lemma 3 *If* $\mathrm{online\_task}(t) \neq \mathrm{offline\_task}(t)$, *then the online schedule has already completed the* $\mathrm{iter}_{\mathrm{offline\_task}(t)}(t)'$*th execution of task* $\mathrm{offline\_task}(t)$.

Proof: Suppose this lemma is not true. We have, from the definition of the model, that $\mathrm{online\_task}(0) = \mathrm{offline\_task}(0)$, so if the lemma is untrue, there exists some $t > 0$ which the earliest time at which $\mathrm{online\_task}(t) \neq \mathrm{offline\_task}(t)$ but the online schedule has not yet finished the $\mathrm{iter}_{\mathrm{offline_task}(t)}(t)'$th execution of task $\mathrm{offline\_task}(t)$.

Let $\mathrm{offline\_task}(t) = T'_{j,n}$ and $\mathrm{online\_task}(t) = T'_{i,m}$. By definition of t, $T'_{j,n}$ is not yet done in the online schedule at time t. Since $T'_{i,m}$ is being executed instead by the online schedule at t, we must have $T'_{i,m} \succ T'_{j,n}$ ($A \succ B$ means that A has higher priority than B).

By definition of t, $\forall \xi < t$, if $online\_task(\xi) \neq offline\_task(\xi)$, then the online schedule has already finished $offline\_task(\xi)$ by time ξ.

Note that we cannot have $online\_task(x) = offline\_task(x)\ \forall x < t$. If this were to happen, then the offline and online schedules would both be exactly parallel until time t. In particular, task $T'_{i,m}$ would execute at precisely the same intervals in both the offline and online schedules prior to t. But, since $T'_{i,m}$ is done by the offline schedule before t, it follows from the voltage-selection rule in Algorithm 2 that it would also be done in the offline schedule before t, which contradicts the assumption that $online\_task(t) = T'_{i,m}$.

The assumption that the lemma is false therefore requires that there must be some time $y < t$ such that $online\_task(y) \neq offline\_task(y)$. But, from the definition of t, we must have for every $z < t$, $online\_task(z) = T'_{i,m}$ whenever $offline\_task(z) = T'_{i,m}$: otherwise, by the definition of t, $T'_{i,m}$ would have been completed before t in the online schedule. Let us now consider two cases:

Case 1. The offline voltage setting of T_i is *low*.
In this case, since the offline schedule finishes executing $T'_{i,m}$ by time t, so must the online schedule, since the offline schedule assumes worst-case execution times. So, Case 1 cannot happen.

Case 2. The offline voltage setting of T_i is *high*.
During times when both the offline and online schedules are executing $T'_{i,m}$, the online schedule will only use a low-voltage setting at some time u when $online\_unf(i,u) < offline\_unf(i,u)$. From this, and the fact that $T'_{i,m}$ is executed in the online schedule whenever it is executing in the offline schedule, it follows that $T'_{i,m}$ must have finished in the online schedule before t.

We therefore have a contradiction: no such t exists, and so the proof is complete. **QED**

Lemma 4 *Every iteration is completed in the online schedule no later than when it is completed in the offline schedule.*

Proof: Suppose this is not true, i.e., that there exists some iteration $T'_{i,m}$ which completes in the offline schedule before it has completed in the offline schedule.

Let t be the time at which $T'_{i,m}$ completes in the offline schedule. By the preceding Lemma, $T'_{i,m}$ will execute in the online schedule whenever it does so in the offline schedule (since otherwise it would be done in the online schedule ahead of t). The result follows immediately from this and the voltage-setting rule of the algorithm. **QED**

From Lemmas 3 and 4, we have the following theorem:

Theorem 3 *All task deadlines are met by the online algorithm.*

4.2 Analysis

An analysis of Algorithm 2 can be done along the same lines as for Algorithm 1. However, since task periods can be different, the number of special cases that have to be considered is very large. Analysis is only useful when it either produces a compact expression that offers insight into performance, or when it allows for faster performance evaluation than simulation. The analysis for Algorithm 2 would be so complex that it would likely satisfy neither requirement. Accordingly, we have restricted ourselves to simulation for studying the performance of Algorithm 2.

4.3 Numerical Results

The experimental setup for these runs has been briefly described earlier. The only difference is that the task periods are chosen randomly to be integers between 1 and 11. Figure 4 shows the energy consumption for an 8-task system for various processor utilizations at high voltage, U_H.

Figure 5 mirrors Figure 3 of the previous section, and has similar characteristics.

4.4 Relaxing Assumption A6

Let us now relax A6, and assume that task phasings are not known in advance. As before, we can compute the offline voltage settings, since these depend only on the need to keep worst-case execution times so that the task set utilization does not exceed 1. However, we cannot precompute the offline schedule. Instead, the offline schedule must be generated on-the-fly, as tasks arrive. In other words, the system builds up the offline schedule as tasks arrive, assuming that the offline voltage settings are used and that each task runs to its worst-case time. As the offline schedule is generated, the system can follow Algorithm 2 to pick the appropriate voltage setting.

To combine the simulation of an offline on-the-fly schedule and the voltage-clock schedule, we can adopt a slack-time queue (ST-QUEUE) to track the slack times resulting from early task completions. Note that a task can execute during the slack time of a finished task or during the period assigned to it in the offline schedule. In the normal EDF task queue (TK-QUEUE), we use t-

wo variables to keep track of the computation times for each task. The first one, ct_i, specifies the computation time that task T_i has consumed during its scheduled period of the offline schedule. This allows us to compute a task's slack time when it finishes. The second variable, cst_i, indicates how long a hv-mode task can stay in lv-mode execution after it steals slack time. As in the TK-QUEUE, the slac k times of the completed tasks are ordered according to a task's deadlines in the slack-time queue.

The steps to perform voltage-clock scheduling are as follows:

S1 When task T_i arriv es, it is inserted into TK-QUEUE. The variables cst_i and ct_i are set to 0.

S2 When task T_i completes, a slack time $st_i = \omega_i - ct_i$ is inserted into ST-QUEUE if the difference is greater than 0.

S3 When the processor is idle (i.e. TK-QUEUE is empty), the slack time at the head of ST-QUEUE decreases ev ery unit of time. Once it reac hes zero, the slac k time is deleted fromST-QUEUE.

S4 When a task T_i is dispatched (under EDF), it can consume slack time st_j at the head of ST-QUEUE, if task T_i has a deadline greater than task T_j. If $offline\_setting(i) = high$, we can switch the setting to low for an additional period $st_j \frac{\phi}{\phi - 1}$ (to be accumulated in cst_i).

S5 When a task T_i cannot find any available slack time for its execution, it is executed at the voltage-clock mode $offline\_setting(i)$ if $cst_i = 0$ or at lv-mode if $cst_i > 0$. Also, the time used in its computation is then accumulated in ct_i.

It can be shown that, at step S5, $i = offline\_task(t)$ when a task T_i cannot find any available slack time for its execution. Thus, the slack time due to an early completion can be computed correctly by $st_i = \omega_i - ct_i$. Also, if $cst_i = 0$ at time t and $offline\_setting(i) = high$,the unfinished work of the executing task T_i (based on the w orst-case execution times) at timet is equal to that of $offline\_unf(i,t)$.

5. Conclusion

In this paper, w e ha ve described simple algorithms for voltage scaling in real-time systems. These algorithms exploit the fact that pow er consumption tends to drop quadratically with voltage, while circuit delays (and thus the clock period) increase only linearly. Our algorithms have offline and online components. The offline component assumes that the tasks run to their worst-case execution times, and computes the voltage settings to minimize energy consumption. The online component starts with the offline voltage settings as a base, and then reclaims any time resources that are released b y tasks which finish ahead of their predicted worst-case execution times, thus making for a further round of energy savings. Our results indicate that significant energy savings are made possible, while guaranteeing that all tasks will contin ue to meet their deadlines.

References

[1] J.-M. Chang and M. P edram, "Energy minimization using multiple supply voltages," *IEEE Trans. VLSI Systems*, V ol. 5, No. 4, December 1997, pp. 436–443.

[2] C. D. Locke, "Software Architecture for Hard Real-time Applications: Cyclic Executives vs. Fixed Priorit y Executives," *Journal of Real-Time Systems*, Vol. 4, 1992, pp. 37-53.

[3] M. M. Khellah and M. I. Elmasry, "P ow erminimization of high-performance submicron CMOS circuits using a dual-V_{dd} dual-V_{th} (DVDV) approach, *Proc. 1999 International Symp. Low-Power Electronics and Design*, pp. 106–108, 1998.

[4] C. M. Krishna and K. G. Shin, *R eal-Time Systems*, New York: McGraw-Hill, 1997.

[5] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," *Proc. 1998 International Symp. Low-Power Electronics and Design*, pp. 76–81, 1998.

[6] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Sc heduling for reduced CPU energy," *Pr oc. USENIX Symp. Operating Systems Design and Implementation*, pp. 13–23, 1994.

[7] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *Proc. 36th IEEE Symp. F oundations of Computer Science*, 1995, pp. 374–382.

[8] *Intr oduction to Thumb*, ARM Documentation, Advanced RISC Machines, Ltd.

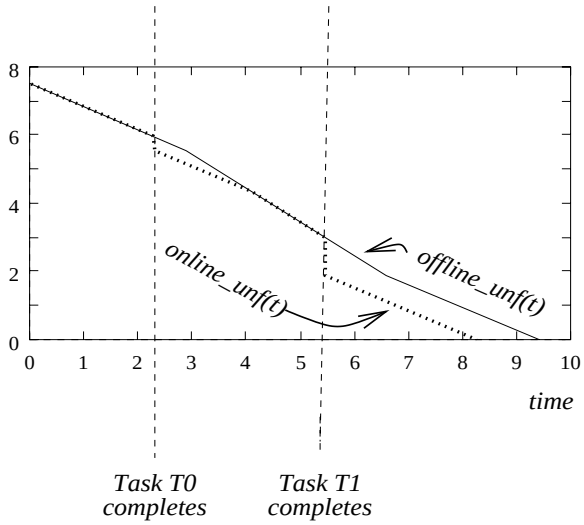


Figure 1. Offline and Online Trajectories

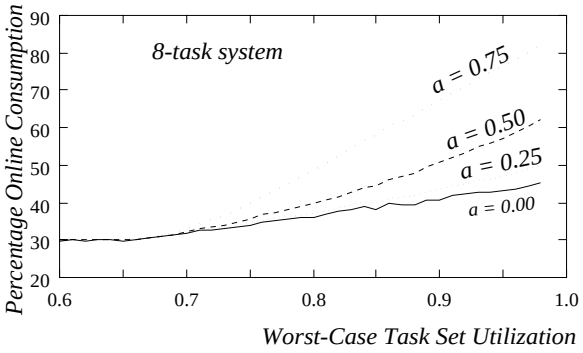
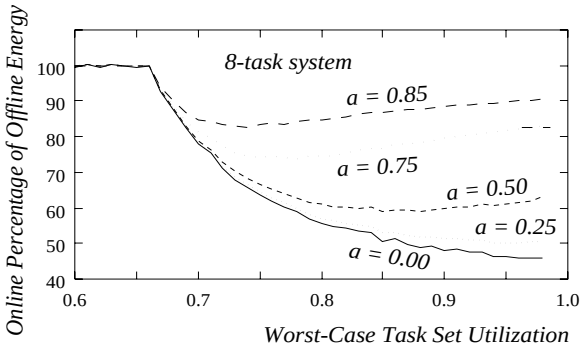
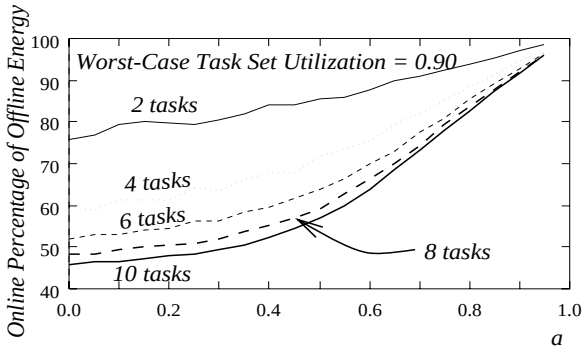


Figure 2. Percentage Energy Consumption



(a) Effect of Reclaiming



(b) Effect of Number of Tasks

$$\text{Online Percentage of OfflineEnergy} = \frac{\text{Energy Consumption of Online Algorithm} \times 100}{\text{Energy Consumption of Offline Algorithm}}$$

Figure 3: Online as a Percentage of Offline Energy Consumption

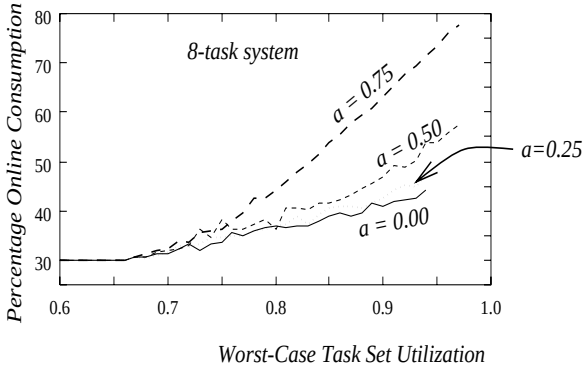
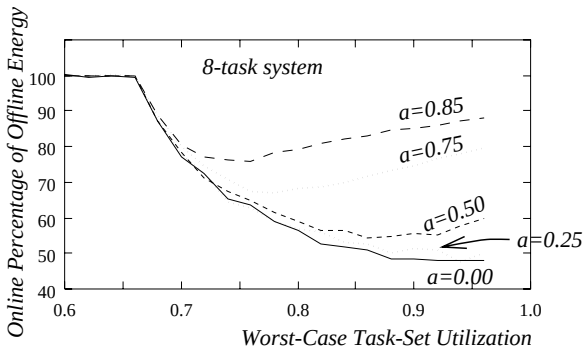
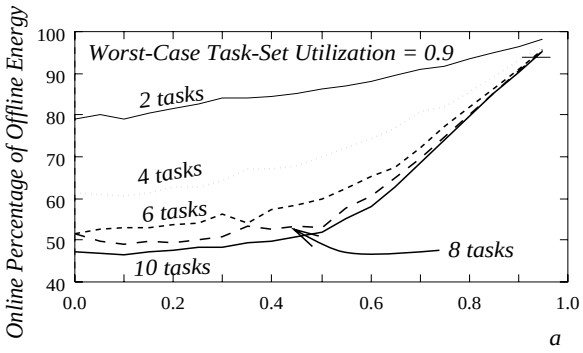


Figure 4: Percentage Energy Consumption



(a) Effect of Reclaiming



(b) Effect of Number of Tasks

$$\text{Online Percentage of OfflineEnergy} = \frac{\text{Energy Consumption of Online Algorithm} \times 100}{\text{Energy Consumption of Offline Algorithm}}$$

Figure 5: Online as a Percentage of Offline Energy Consumption

U_H	No of Tasks			
	2	4	6	8
0.60	0.900	0.900	0.900	0.900
0.65	0.975	0.975	0.975	0.975
0.70	0.893	0.975	0.992	0.997
0.75	0.892	0.979	0.995	0.999
0.80	0.924	0.982	0.996	0.999
0.85	0.898	0.974	0.994	0.999
0.90	0.916	0.968	0.991	0.998
0.95	0.953	0.967	0.983	0.993

Note: All utilizations are for worst-case task run times.

Table 1. Average Processor Utilization with Offline Settings