Shortest Path Algorithms for Real-Time Scheduling of FIFO tasks with Minimal Energy Use

Bruno Gaujal and Nicolas Navet
INRIA - Trio Team
and
Cormac Walsh
INRIA - Maxplus Team

We present an algorithm for scheduling a set of non-recurrent tasks (or jobs) with FIFO real-time constraints so as to minimize the total energy consumed when the tasks are performed on a dynamically variable voltage processor. Our algorithm runs in linear time and is thus an improvement over the classical algorithm of Yao et al. in this case. It was inspired by considering the problem as a shortest path problem. We also propose an algorithm to deal with the case where the processor has only a limited number of clock frequencies. This algorithm gives the optimum schedule with the minimum number of speed changes, which is important when the speed switching overhead cannot be neglected. All our algorithms are linear in the number of tasks if the arrivals and deadlines are sorted and need $O(N \log N)$ time otherwise. These complexities are shown to be the best possible. Finally, we extend our results to fluid tasks and to non-convex cost functions.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Real-time systems, low-power design, scheduling, voltage scaling

1. INTRODUCTION

To provide more functionality and better performance, embedded systems have an increasing need for computation power. This requires the use of high frequency electronic components that consume much electrical power. Currently, battery technology is not progressing sufficiently quickly to keep up with demand. All battery operated systems, such as PDAs, laptops and mobile phones, would benefit from better energy efficiency. Reducing energy consumption will not only lead to a longer operating time but also to a decrease of the weight and space devoted to

Authors'adresses: B. Gaujal, Laboratoire ID, Ensimag - Zirst 51, avenue Jean Kuntzmann, 38330 Montbonnot, France; email: {bruno.gaujal@imag.fr}; N. Navet, LORIA - TRIO Team, Campus Scientifique - B.P. 239, 54506 Vandoeuvre-lès-Nancy, France; email: {nnavet@loria.fr}; C. Walsh, INRIA Rocquencourt, Maxplus Team, Domaine de Voluceau-Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France; email: {walsh@inria.fr}.

the battery.

Amongst the hardware and software techniques aimed at reducing energy consumption, supply voltage reduction is particularly effective. This is because the power dissipated in CMOS circuits is proportional to the square of the supply voltage. Of course, the reduction in voltage results in a diminution of the processor speed. In the last few years, variable voltage processors have become available and much research has been conducted in the field of dynamic voltage scaling. When real-time constraints are matter of concern, the extent to which the processor speed can be reduced depends on the tasks' characteristics (execution time, arrival time, deadline ...) and on the underlying scheduling policy.

Power conscious versions of the two classical real-time scheduling policies, namely EDF (Earliest Deadline First) and FPP (Fixed Priority Pre-emptive), have been proposed. For FPP, Shin and Shoi in [1999] have presented a simple run-time strategy that reduces energy consumption. Quan and Hu, in [2001], proposed a more complex algorithm that was more efficient in their experiments. More recently, Yun and Kim in [2003] prove that computing the optimal voltage schedule of a set of tasks under FPP is NP-Hard and they present an approximation scheme running in polynomial time that gives a schedule as close to optimal as desired.

Yao *et al.* in [1995] proposed an off-line algorithm, based on EDF, for finding the optimal voltage schedule of a set of independent tasks. They also presented some on-line heuristics and gave lower bounds on their efficiency. Other on-line heuristics based on EDF have been proposed, for instance in [Hong et al. 1998] for the problem of scheduling both periodic and aperiodic requests.

Other directions of research concern the difference between worst-case execution times (WCET) and actual execution times. One class of algorithms, known as "stochastic scheduling" [Lorch and Smith 2001; Gruian 2001; 2002] try to finding a feasible speed schedule that minimizes the expected energy consumption. A second class of techniques [Mossè et al. 2000; Shin et al. 2001] is known as "compiler-assisted scheduling". Here, tasks are divided into sections for which the WCET is known and the processor speed is re-computed at the end of each section according to the difference between the WCET and the time that was actually needed to execute the task. Other strategies dynamically collect the unused computation time at the end of each task and share it among the remaining active tasks. Examples of this approach, called "dynamic reclaiming" include [Aydin et al. 2001] and [Zhang and Chanson 2002]. Numerous other studies have been conducted on dynamic voltage scheduling; the reader may refer to [Gruian 2002] for a survey.

The study published in [Yao et al. 1995] remains one of the most important in the field because it provides, for independent tasks with deadlines, the schedule that minimizes energy usage while ensuring that deadlines are met. The algorithm works by identifying the time interval, called the critical interval, over which the highest processing speed is required. The tasks belonging to this interval (those with arrival date and deadline inside the interval) are then removed and a sub-problem is constructed with the remaining tasks.

Our contribution is complimentary to [Yao et al. 1995] in the sense that we first address the particular case of tasks having FIFO constraints and then propose several extensions that work in all cases (non-FIFO). More precisely, the problem of

finding the minimal voltage schedule is here reduced to a shortest path problem when the tasks constraints are FIFO. This enables us to provide linear-time algorithms for minimizing the energy consumption of a set of non-recurrent tasks in the following situations:

(1) when the processor speed range is continuous (the processor can have an arbitrary clock frequency up to a certain maximum),
(2) when the number of speeds is discrete,
(3) when the number of speeds is discrete and there is the additional objective of minimizing the number of speed changes. To the best of our knowledge, in our context, this problem has not been addressed before.
(4) To show that minimizing the number of speed change actually minimize the total energy consumption up to a small factor.

Furthermore, extensions 2 and 3 are still valid without the FIFO assumption. Another contribution of the paper is a proof that, when there are only a finite number of speeds, the optimal solution uses at each instant the two speeds that bracket the ideal speed. This result was established in [Ishihara and Yasuura 1998] for a single task considered alone but, to the best of our knowledge, was not known for a global optimization over a set of tasks. We also consider the case of fluid tasks and non-convex cost functions.

Section 2 describes the system model and studies the problem in the case where the processor has a continuous range of speeds available. In Section 3 an algorithm giving the optimal schedule is described and its complexity is shown to be linear. In Section 4, the problem where the processor has a finite number of speeds is investigated in three steps: first without minimizing the number of speed changes, then with this additional objective, and finally by showing that the overhead induced by speed changes is upper bounded by a small constant. Section 5 is concerned with fluid tasks and Section 6 with non-convex cost functions.

2. STATEMENT OF THE PROBLEM

We consider a system consisting of a single processor which must execute some work under real-time constraints. We denote by $A(t)$ the amount of work that has arrived up to time t and by $D(t)$ the amount that the processor must have executed by time t. The latter is determined by the deadlines of the tasks. By definition, the functions A and D are non-decreasing and $A(t) \geq D(t)$ for all t. With no loss of generality, one can assume that $A(0) = 0$ and $D(0) = 0$. For simplicity, we shall assume that A and D are piece-wise continuous.

The tasks are characterized by the set $\{(a_n, s_n, d_n)\}_{n=1\cdots N}$ where the quantities a_n, s_n, d_n respectively denote the arrival time, the size (i.e. execution time at maximum speed) and the deadline of task n. Note that such non-recurrent tasks are sometimes called "jobs" or even "aperiodic tasks" in the literature. Nevertheless, we point out that the results presented in this paper can also be applied to periodic tasks by computing the schedule over a time interval equal to the least common multiple of all task periods.

For the remainder of this section, we assume that tasks have FIFO real-time constraints, in other words, $a_i \leq a_j \Rightarrow d_i \leq d_j, \forall i, j$. FIFO constraints occur

for instance in data processing in mobile phones or multimedia devices. Also, many systems do not actually have real-time constraints and deadlines are often used merely as a convenient way to specify a minimum level of performance in power-aware systems (the best strategy to minimize energy consumption without performance requirements would be to stay idle). For such systems, the FIFO assumption seems quite natural.

The functions $A(t)$ and $D(t)$ are staircase functions (*i.e.* piece-wise constant, with a finite number of pieces) :

$$A(t) = \sum_{i=1}^{N} s_i \cdot \mathbf{1}_{[a_i<t]},$$

$$D(t) = \sum_{i=1}^{N} s_i \cdot \mathbf{1}_{[d_i\leq t]},$$

Note that the function A is left-continuous and the function D is right-continuous.

The processor speed u can vary with time over a continuous range from 0 to 1 (after a possible re-scaling). The case where there are only a finite number of speeds $\{v_1 \cdots v_\ell\}$ will be investigated in section 4.

The scheduling problem is to choose at each time t the speed $u(t)$ in such a way as to complete all tasks before their deadlines while minimizing the total energy consumption between time 0 and time T, where T is the time horizon of the problem. The energy consumption of the processor at time t, denoted $e(t)$, is a function of its speed $u(t)$. In the following we assume that $e(t) = g(u(t))$ where g is an arbitrary increasing convex function[1] over $\mathbb{R}_+$. One can express the problem in mathematical terms:

PROBLEM 2.1. *Find an integrable function* $u : [0, T] \to \mathbb{R}$ *such that*

$$\int_0^T g(u(s))ds \text{ is minimized,} \tag{1}$$

under the constraints

$$u(t) \geq 0 \quad \forall t \in [0, T], \tag{2}$$

$$\int_0^t u(s)ds \leq A(t) \quad \forall t \in [0, T], \tag{3}$$

$$\int_0^t u(s)ds \geq D(t) \quad \forall t \in [0, T]. \tag{4}$$

A function u satisfying the constraints is called an *admissible* solution. An admissible u minimizing $\int_0^T g(u(s))ds$ is called an *optimal* solution.

THEOREM 2.2. *There exists a schedule meeting all timing constraints if and only of the speed of the processor* $u(t)$ *satisfies constraints* (2), (3) *and* (4).

[1]with CMOS technology, typically $e(t) \approx \alpha C u(t)^{1+2/(\gamma-1)}$, where $1 \leq \gamma \leq 3, \alpha \geq 0, C \geq 0$. See [Gruian 2002] for more details.

PROOF. Tasks are sorted according to their arrival dates and ties are broken according to the deadlines. We first prove that if $u(t)$ satisfies (2), (3), and (4) then it is possible to find a schedule meeting the FIFO real-time constraints. Let $U(t) \stackrel{\text{def}}{=} \int_0^t u(s)ds$. We prove that the FIFO policy leads to a feasible schedule. The proof works by induction on the number of tasks. The property is obvious for $N = 1$. We consider the nth task. Let $S_k = \sum_{i=1..k} S_i$.

We define $t_{n-1} = U^{-1}(S_{n-1})$ where $U^{-1}(y) = \sup\{x \,|\, U(x) \leq y\}$. By induction, tasks $1, 2, \ldots, n-1$ have been successfully scheduled up to time t_{n-1}. One has $a_n \leq t_{n-1}$ since $A(a_n) = S_{n-1}$, $U(t_{n-1}) = S_{n-1}$, and, by constraint (3), $A(t) \geq U(t)$.

We define $t_{n-1} = U^*(S_{n-1})$ where $U^*(y) = \inf\{x \,|\, U(x) \geq y\}$. One has $U(t_n) = S_n$ (by definition) and $D(d_n) = S_n$ (by the FIFO assumption). This implies that $t_n \leq d_n$ using constraint (4).

We schedule task n between times t_{n-1} and t_n (this is possible because $t_{n-1} \geq a_n$). In this time interval, the processor can execute s_n units of work and meets the real-time constraints since $t_n \leq d_n$.

For the second part of the proof, we assume that tasks are scheduled using the EDF policy. Between time t_{n-1} and t_n, the task with the earliest deadline is task n. The scheduling is feasible if and only if $t_n \leq d_n$. Since $D()$ is the smallest non-decreasing function such that $D(d_n) = S_n$ (FIFO property) we have that $U()$ is larger than $D()$. Since EDF is optimal for feasibility (see [Jackson 1955] quoted in [Stankovic et al. 1998]) this means that constraint (4) must be satisfied. Constraint (2) just means that the speed is necessarily non-negative. As for constraint (3), if $U(t)$ is larger than $A(t)$, this would mean that the processor has executed more work than has arrived at time t, which is impossible whatever the scheduling policy. □

Note that the problem statement only uses the integral $U(t) \stackrel{\text{def}}{=} \int_0^t u(s)ds$ of u. Therefore, the function u need only be defined almost everywhere (a.e.). In the following, we will identify all functions which are equal a.e. .

The system (A, D) is said to be *feasible* if, when setting $u(t) = 1$ (i.e. using the processor at maximal speed) and when scheduling under EDF, no time constraint is violated. Actually, in order to take into account the fact that the speed of the processor cannot exceed 1, one must consider a more constrained problem:

PROBLEM 2.3. *Find an integrable function $u : [0, T] \to \mathbb{R}_+$ such that*

$$\int_0^T g(u(s))ds \text{ is minimized,}$$

under the constraints (2), (3), (4) *and*

$$u(t) \leq 1 \quad \forall t \in [0, T]. \tag{5}$$

In the rest of the paper, we focus on the problem of determining the optimal speed using the equivalence proved in theorem 2.2. We assume that tasks are scheduled under EDF as in the proof of theorem 2.2.

2.1 Main result

In this section, we characterize the speed functions u that are solutions to Problem 2.1.

LEMMA 2.4. *If the function g is strictly convex, then the optimal solution of Problem 2.1 is unique (up to a set of measure zero).*

PROOF. Consider the set of all integrable functions satisfying constraints (2), (3) and (4). This set is obviously convex. Assume that two functions u_1 and u_2, differing over a set S of positive measure, both minimize the energy: $\int_0^T g(u_1(s))ds = \int_0^T g(u_2(s))ds$. Then for all $0 < \alpha < 1$ and all $t \in S$, $g(\alpha u_1(t) + (1-\alpha)u_2(t)) < \alpha g(u_1(t)) + (1-\alpha)g(u_2(t))$ by strict convexity of g. Therefore,

$$\begin{aligned}\int_0^T g(\alpha u_1 + (1-\alpha)u_2)ds \;&<\; \int_0^T \alpha g(u_1)ds + \int_0^T (1-\alpha)g(u_2)ds \\ &= \int_0^T g(u_1)ds.\end{aligned}$$

This clearly contradicts the optimality of u_1. □

THEOREM 2.5. *If u^* is the optimal solution of Problem 2.1 where g is strictly convex and non-decreasing, then u^* is also an optimal solution of Problem 2.1 for any other non-decreasing convex function.*

PROOF. We first consider the case when g is strictly convex. Consider the problem of minimizing

$$\int_{t_0}^{t_1} g(u(t))dt$$

under $U(t_0) = U_0$ and $U(t_1) = U_1$ in the absence of any other constraints. This is an easy problem in the Calculus of Variations. The solution given by Euler's formula is $d/dt\, g'(u) = 0$, which implies that u is constant. Thus, if $(t_0, U^*(t_0))$ and $(t_1, U^*(t_1))$ are two points on the optimal path $U^* \stackrel{\text{def}}{=} \int_0^t u^*(s)ds$, then U^* has constant slope between these two points if this is feasible. We conclude that U^* only changes slope at the arrival times $\{a_n\}_{1\le n\le N}$ or deadline times $\{d_n\}_{1\le n\le N}$. Moreover, when the slope of U^* decreases we must have $U^*(t) = D(t)$ and $t = d_n$ for some $n \in \{1, \ldots, N\}$. Otherwise, in the neighborhood of t, U^* should be a straight line if it were feasible. Likewise, when the slope of U^* increases we must have $U^*(t) = A(t)$ and $t = a_n$ for some $n \in \{1, \ldots, N\}$.

We will show that these properties, together with $U^*(0) = 0$ and $U^*(T) = U_T$, completely determine U^*. Suppose that there are two functions U_1 and U_2 meeting the constraints with the properties above such that $U_1(0) = U_2(0) = 0$. Let τ be the first time that U_1 and U_2 differ. Then the right derivative of U_1 (say) at τ is strictly greater than that of U_2. Therefore U_1 is strictly greater than U_2 at the next event τ_1, be it arrival or deadline. We can have neither $U_1(\tau_1) = A(\tau_1)$ nor $U_2(\tau_1) = D(\tau_1)$. Therefore the slope of U_1 cannot decrease at τ_1 and that of U_2 cannot increase. Proceeding by induction, we get that $U_1(t) > U_2(t)$ for all $t > \tau$. Thus $U_1(T)$ and $U_2(T)$ must differ. □

In the following, the solution u^* defined above will be called "the optimum solution of Problem 2.1". As will be seen in Section 4, when g is not strictly convex there may be other solutions equally as good.

Note that $g(x) = \sqrt{1+x^2}$ is strictly convex and increasing, and that $\int_0^T \sqrt{1+u^2(s)}ds$

is the length of the curve of the function $U(t) \stackrel{\text{def}}{=} \int_0^t u(s)ds$ from $t = 0$ to $t = T$. Hence, the optimal solution can be interpreted in the following way. Consider a road bounded by two concrete walls (the functions A and D). Find the shortest path from the beginning to the end. This gives U^*.

COROLLARY 2.6. *The optimal solution u^* of Problem 2.1 satisfies the following inequality*[2]*:*
$\sup_{0\le t\le T} u^*(t) \le \sup_{0\le t\le T} u(t)$, *for all functions u satisfying constraints* (2), (3) *and* (4).

PROOF. Consider the function $g_n(x) = x^n$. The function g_n is increasing and convex over $[0, T]$. Applying Theorem 2.5, we see that the optimal solution u^* of Problem 2.1 with $g = g_n$ does not depend on n. Now consider the limit when n goes to infinity: over the interval $[0, T]$, for all functions u,

$$\lim_{n\to\infty} \left(\int_0^T g_n(u(s))ds \right)^{1/n} = \sup_{0\le t\le T} u(t).$$

This shows that $\sup_{0\le t\le T} u^*(t) \le \sup_{0\le t\le T} u(t)$. □

COROLLARY 2.7. *If the system (A, D) is feasible, then the optimal solution u^* of Problem 2.1 satisfies $u^*(t) \le 1$, and therefore problems 2.1 and 2.3 are equivalent.*

PROOF. Consider the case where the system (A, D) is feasible. This means that their exists a solution u to the set of constraints of Problem 2.3. This solution is also a solution to the set of constraints of Problem 2.1. Using Corollary 2.6, the optimal solution of Problem 2.1 satisfies $\sup_{0\le t\le T} u^*(t) \le \sup_{0\le t\le T} u(t) \le 1$. Therefore u^* is also an optimal solution to Problem 2.3. □

An example illustrating Problem 2.1 is given in Figure 1.

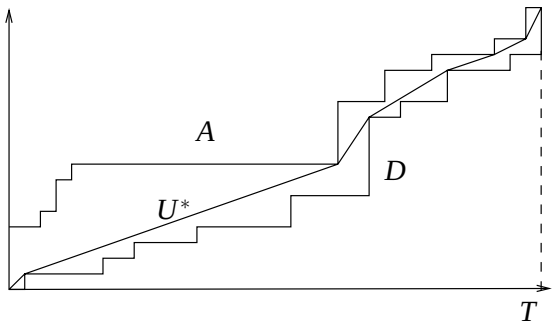

Fig. 1. The function U^* is the shortest path from point 0 to T.

[2]here, the sup operator stands for the essential supremum, since all functions are only defined almost everywhere.

Remark 2.8. Conversely, if the system is not feasible then the optimal solution of Problem 2.1 will not satisfy $u^* \leq 1$. This provides a feasibility test for a set of non-recurrent independent tasks with FIFO real-time constraints under EDF.

Actually more precise results can be stated. Since u^* is piece-wise constant, one need only focus on the discontinuity points of u^*. We denote these points by $P_1, \cdots, P_n$. They either belong to the graph of A or to the graph of D. The first point is $P_1 = (0, 0)$ and belongs to both A and D. This sequence of points can be split into sub-sequences of consecutive points belonging to A or to D.

COROLLARY 2.9. *Consider a sub-sequence $P_r, \cdots, P_s$ of consecutive discontinuity points of u^* such that $P_r \in A, P_s \in A$ and all other points in between belong to D. Then, between points P_r and P_s, U^* is the upper concave envelope of the points $P_r, \cdots, P_s$.*

Dually, consider a sub-sequence $P_i, \cdots, P_j$ of consecutive discontinuity points of u^* such that $P_i \in D, P_j \in D$ and all other points in between belong to A. Then, between points P_i and P_j, U^* is the lower convex envelope of the points $P_i, \cdots, P_j$.

PROOF. (Sketch) This is a direct consequence of the fact that U^* is the shortest path going through the points $P_1, \cdots, P_n$. □

The function u^* can be computed according to the method described in [Yao et al. 1995] which is based on the computation of critical intervals. This algorithm is, in the worst case, cubic in the number of tasks N: there are at most N successive critical intervals and finding a single interval requires $O(N^2)$ since for each arrival there are at most N deadlines to consider. In [Yao et al. 1995], the authors claim that using "a suitable data structure such as the segment tree" the running time can be reduced to $O(N \log^2(N))$. However, this was never really achieved, as mentioned in [Yao 2003]. We actually do not know how to obtain an implementation with their algorithm in less than $O(N^3)$ and we are not aware of any paper in the literature that has addressed the problem with a complexity lower than $O(N^3)$.

An interesting consequence of the construction of u^* as given in [Yao et al. 1995] is that there exists a schedule with minimal energy consumption such that each task is executed at a constant speed. This is not obvious with our approach. However, as seen in the following sections, our approach has other important advantages. In particular, we will see in section 3 that there exists a linear time algorithm to compute u^*.

3. A LINEAR-TIME ALGORITHM TO COMPUTE U^*

While Theorem 2.5 characterizes the function u^*, it is not constructive. This section shows how to construct u^*. We assume that the function A (respectively D) is given in the form of an ordered list $L_A := [(a_1, A(a_1)), \cdots (a_N, A(a_N))]$ (respectively $L_D := [(d_1, D(d_1)), \cdots, (d_N, D(d_N))]$) with $a_1 < \cdots < a_N$ (respectively $d_1 < \cdots < d_N$).

We describe an algorithm that constructs the function $U^*(t)$ (as well as $u^*(t)$) in the form of an ordered list $(x_1, y_1), \cdots, (x_K, y_K)$ with $x_1 < \cdots < x_K$, the function U^* being the linear interpolation between these points. This algorithm is similar to the linear time algorithm to compute the convex hull of a set of ordered points in the plane (see for example [Boissonnat and Yvinec 1995]).

The main idea of the algorithm is to construct two piece-wise affine functions V and W inductively by introducing the points of A and D one by one. Both functions start with the same initial value at the initial point: $V(x_0) = W(x_0)$. The function U^* and V or W share a common prefix, determined by the algorithm.

step 0. Merge the lists L_A and L_D into a single list L ordered by the first coordinate.

step 1. Set $x_0 := (0,0)$, $V := [(0,0)]$, $W := [(0,0)]$.

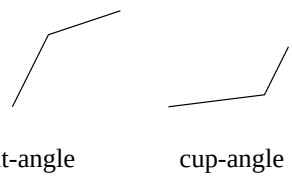

Fig. 2. A hat-angle and a cup-angle

step 2. Sweep the list L. We keep track of the following data. $V = [(V_0^1, V_0^2), \cdots (V_k^1, V_k^2)]$ is the lower convex hull of the function A from point x_0 to $(a_i, A(a_i))$ and $W = [(W_0^1, W_0^2), \cdots, (W_m^1, W_m^2)]$ is the upper concave hull of the function D from point x_0 to $(d_j, D(d_j))$. Here i and j are the number of points in A and D, respectively, that have been processed so far.

—If the next point in L belongs to A (*i.e.* it is $(a_{i+1}, A(a_{i+1}))$),
1. Add it in the last position $(k+1)$ to the list $V := V \cdot (a_{i+1}, D(a_{i+1}))$.
2. Update the list V by removing points starting from (V_k^1, V_k^2) and going backwards as long they form a hat angle (figure 2).
3. If all intermediate points are removed (the first point (V_0^1, V_0^2) cannot be removed), update everything as follows:

$$\begin{aligned} \ell &:= 0, \quad \text{while } V \text{ is below } W \text{ at point } W_{\ell+1} \text{ do } \ell := \ell + 1 \text{ od} \\ U^* &:= U^* \cdot [(W_0^1 W_0^2) \cdots (W_\ell^1, W_\ell^2)] \\ W &:= [(W_\ell^1, W_\ell^2), \cdots, (W_m^1, W_m^2)] \\ V &:= [(W_\ell^1, W_\ell^2), (V_{k+1}^1, V_{k+1}^2)] \\ x_0 &:= (W_\ell^1, W_\ell^2). \end{aligned}$$

—If the next point belongs to D (*i.e.* $(d_{j+1}, D(d_{j+1}))$), do the same as above switching the role of V and W, replacing the hat-angle test with a cup-angle test and testing if W is above V instead of V below W.

step 3. Once the last point in L has been swept, update for the last time the list U^* by concatenating U^* and W, in other words $U^* := U^* \cdot W$.

Notice that the algorithm constructs U^* rather than u^*. However, since U^* is piece-wise affine, it is easy to retrieve u^* from U^*. An example of the algorithm in action is shown Figures 3, 4, and 5. Figure 3 gives the current position where V and W have been constructed up to the current point. As shown in Figure 4, the next point W_4 belongs to D, so we update W. The angle at point W_3 is a cup-angle

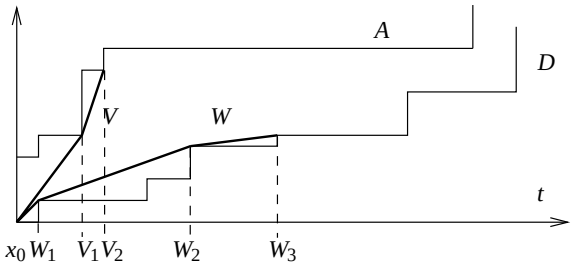

Fig. 3. The current position.

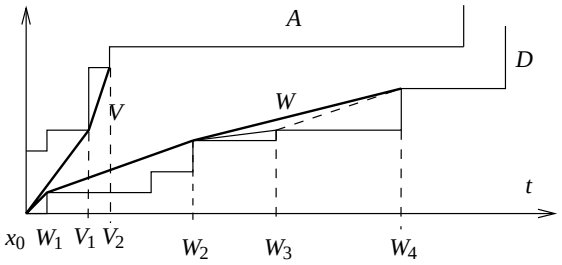

Fig. 4. A simple case where only W is updated.

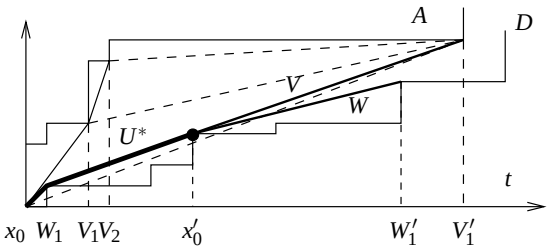

Fig. 5. A case where x_0 is updated.

and so it is removed from the list. The point W_2 does not form a cup-angle in the new W function so no more points need to be removed. In Figure 5, we add yet another point, V_3. We update V. Point V_2 forms a hat-angle and is removed. Now point V_1 forms a hat-angle and it too is removed. Since all intermediate points in V have been removed, we must check whether V remains above W. This is not the case since both W_1 and W_2 are above V. Point W_3 is below V. This means that we update the starting points of both V and W to the new starting point $x_0' = W_2$. The new V consists of the two points x_0' and V_3, while the new W consists of the two points x_0' and W_4. The function U^* is extended from 0 to the new starting point x_0' (thicker line in the figure).

THEOREM 3.1. *The algorithm described above constructs the function u^* in time $O(N)$, using a memory size $O(N)$.*

PROOF. That the function constructed by the algorithm is actually u^*, the optimal solution of Problem 2.1, is a direct consequence of the proof of Theorem 2.5 and Corollary 2.9.

The proof that the algorithm runs in linear time with linear memory size is similar to the proof for the algorithm computing the convex hull of a set of ordered points ([Boissonnat and Yvinec 1995]). A simple way of looking at it is to notice that the total time needed to construct V, W and u^* is proportional to the number of changes occurring in the lists V, W and u^*. Since each point in these lists is eliminated at most once, the number of changes is proportional to N. □

Note that if the original data is in the form of a set of tasks rather than staircase functions A and D, then one needs, as the first step, to create the lists L_A and L_D, which may take $O(N \log(N))$ time if the tasks are not already sorted. In any case, our algorithm is an improvement over the best previously known algorithm, which was given in [Yao et al. 1995] since this algorithm requires $O(N^3)$ time.

3.1 Optimal complexity

In the previous section, we showed that the construction of the function u^* (in the form of an ordered list of its points of discontinuity, called L^* in the following) requires $O(N)$ elementary operations (comparisons, additions and multiplications) when the data lists L_A and L_D are already sorted.

This complexity is optimal because any algorithm would have to at least examine all the points in the data lists to construct u^*. When the data lists L_A and L_D are not already sorted, our algorithm needs a pre-processing phase to sort them before constructing u^*. The total complexity jumps to $O(N \log N)$.

Next, we show that the computation of the list L^* is at least as complex as sorting $L_A \cup L_D$. Consider the following problem:
Input: the set of the arrivals (unsorted): $\{a_i, i = 1 \cdots N\}$ and deadlines (unsorted) $\{d_i, i = 1 \cdots N\}$.
Output: the sorted list L^*.

If the size of the jumps of the cumulative functions A and D are chosen appropriately, then the list L^* will contain all the discontinuity points of A and D, so that it is actually equivalent to a sorted list of the set $\{a_i, i = 1 \cdots N\} \cup \{d_i, i = 1 \cdots N\}$. Here is a way to choose the jumps of both A and D. The choice is made iteratively. Assume that the first $i-1$ discontinuity points have been constructed already. We now look at the ith point p_i, which may be a discontinuity of either A or D. There are two cases.

If the previous point belongs to D (say $(d_k, D(d_k))$, then choose the height of the current point (regardless of whether it belongs to A or to D) to be in the interval $[D(d_k), D(d_k) + (p_i - d_k)u^*(d_k)]$.

If the previous point belongs to A (say $(a_j, A(a_j))$, then choose the height of the current point (regardless of the fact that it belongs to A or D) to be greater than $(p_i - a_j)u^*(a_j)]$.

By choosing the functions A and D in this way, we ensure that all the discontinuity points of A and D are also discontinuity points of u^*, so that they will all be listed in L^*.

The arithmetic complexity of the computation of U^* is $O(N \log(N))$ operations

(counting additions, multiplications and comparisons). This also allows us to sort the initial list $L_A \cup L_D$ in the same amount of time. To our knowledge, no algorithm sorting a list of numbers with arithmetic complexity (number of additions, multiplications and comparisons, regardless of the size of the numbers) lower than $O(N \log(N))$ has been found so far. This provides strong evidence that our algorithm has the lowest possible arithmetic complexity.

3.2 Experimental results

To evaluate the gain that one can expect in practice with our proposal (which we call the "shortest path algorithm") with respect to the Yao et al's algorithm, experiments were conducted with both algorithms on the same random sets of FIFO jobs.

The sets of jobs are created according to the following procedure, which ensures the FIFO property:

—in a time interval of arbitrary length, choose randomly (uniform distribution) $2N$ points where N is the number of jobs,

—the list of points is swept and each point is set at random to be either an arrival date or a deadline date (care is taken that at each step there are at most as many deadlines as arrivals and that at the end the number of arrivals is equal to the number of deadlines),

—the ith arrival date and the ith deadline date form the ith job where the execution time is a random positive value.

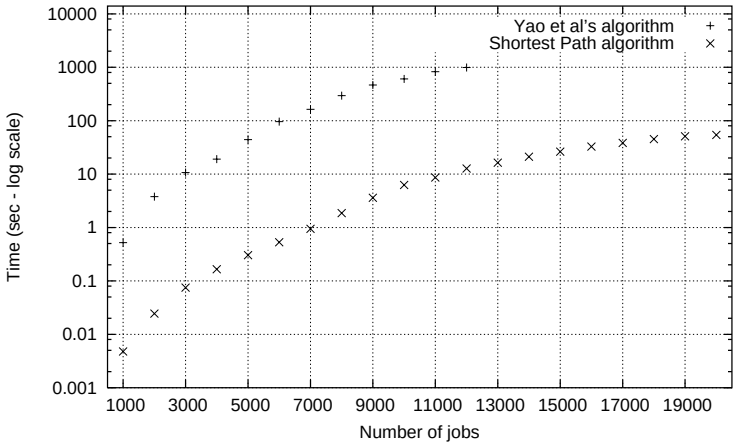

Fig. 6. Computation times of the optimal voltage schedule for the Shortest Path and Yao et al algorithms on the same random sets of tasks. For each problem size, the result is the average of 20 experiments. Computations were stopped when at least one experiment took more that one hour of CPU time.

The computation times for each algorithm are shown in Figure 6 for a number of jobs varying between 1000 and 20000. Each point is the average value of twenty experiments. Computations were stopped when at least one experiment took more

than one hour of CPU time on a 2Ghz CPU and no point is drawn in this case. When the number of jobs becomes greater than 12000, some solutions cannot be found within one hour of CPU time with Yao et al's algorithm while problems with several millions of jobs can be solved in the same amount of time using the Shortest Path algorithm. Globally, from figure 6, one observes a speedup of around 100 using our algorithm. It has to be noted that in practice, it is sometimes necessary to handle very large sets of jobs since the computing of the voltage schedule for periodic tasks with arbitrary deadlines has to be done for the least common multiple of all task periods.

4. FINITE NUMBER OF SPEEDS

We now consider the case where the clock frequency of the processor can only take a finite number of values $v_1 \leq \cdots \leq v_\ell$. As explained in [Gruian 2002], this is necessarily the case with today's technology.

PROBLEM 4.1. *Find an integrable function* $z : [0, T] \to \mathbb{R}_+$ *such that*

$$\int_0^T g(z(s))ds \text{ is minimized,}$$

under Constraints (3) *and* (4) *and the additional constraint*

$$z(t) \in \{v_1, \cdots, v_\ell\} \quad \forall t \in [0, T]. \tag{6}$$

Let $u^*(t)$ be the solution of Problem 2.1. We assume that $v_1 \leq u^*(t) \leq v_\ell$ for all $0 \leq t \leq T$, so that the range of speeds which are available cover the speeds needed by the processor. This assumption will be satisfied in the typical situation where $v_1 = 0$ (the processor can idle) and $v_\ell = 1$ (the processor can use its maximal speed), and the set of tasks is feasible.

We now describe how to construct an optimal solution z^* to Problem 4.1. Partition $[0, T]$ into contiguous intervals in such a way that the boundaries between intervals are the discontinuity points of u^*. There will be $M < 2N$ intervals. On any one of these intervals, say $I_k = [b_k, b_{k+1})$, u^* is constant as seen in the proof of Theorem 2.5; denote its value by u_k^*. Then, u_k^* falls between two possible speeds for the processor, v_i and v_{i+1}. Let α_k be defined by $u_k^* = \alpha_k v_i + (1 - \alpha_k)v_{i+1}$ Now, construct a function over I_k: $z^*(t) = v_i$ over $[b_k, c_k)$ and $z^*(t) = v_{i+1}$ over $[c_k, b_{k+1})$, where $c_k = (1 - \alpha_k)b_k + \alpha_k b_{k+1}$. It is clear that the function $z^*(t)$ is an admissible solution for Problem 4.1 since it satisfies all the constraints. Furthermore, as shown in the following theorem, it is an optimal solution.

THEOREM 4.2. *Under the foregoing assumptions, the function* $z^*(t)$ *is an optimal solution to Problem 4.1.*

PROOF. Using the assumption on the range of the v_i's, for any u such that $v_1 \leq u < v_\ell$, there exist $i(u)$ such that $v_{i(u)} \leq u < v_{i(u)+1}$. We introduce the coefficient α_u such that $u = \alpha_u v_{i(u)} + (1 - \alpha_u)v_{i(u)+1}$ and consider the real function $\tilde{g}(u) \stackrel{\text{def}}{=} \alpha_u g(v_{i(u)}) + (1 - \alpha_u)g(v_{i(u)+1})$.

First, note that $\tilde{g}$ is the linear interpolation of g over the points $v_1, \cdots v_\ell$. Since g is convex and non-decreasing, $\tilde{g}$ is also convex and non-decreasing.

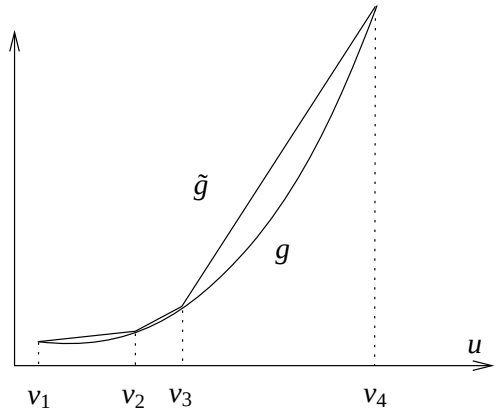

Fig. 7. The functions g and its linear interpolation $\tilde{g}$.

The second part of the proof consists of showing that

$$\int_0^T g(z^*(s))ds = \int_0^T \tilde{g}(u^*(s))ds. \tag{7}$$

Indeed, using the definition of z^*,

$$\int_0^T g(z^*(s))ds = \sum_{k=1}^{M} \int_{I_k} g(z^*(s))ds = \sum_{k=1}^{M} \left(\int_{b_k}^{c_k} g(v_i)ds + \int_{c_k}^{b_{k+1}} g(v_{i+1})ds \right)$$
$$= \sum_{k=1}^{M} ((c_k - b_k)g(v_i) + (b_{k+1} - c_k)g(v_{i+1})) = \sum_{k=1}^{M} (b_{k+1} - b_k)(\alpha_k g(v_i) + (1 - \alpha_k)g(v_{i+1}))$$
$$= \sum_{k=1}^{M} (b_{k+1} - b_k)\tilde{g}(u_k^*) = \int_0^T \tilde{g}(u^*(s))ds.$$

Now, let z be any admissible solution to Problem 4.1. So, $z(t) \in \{v_1 \cdots, v_\ell\}$. Since the function $\tilde{g}$ coincides with g over $\{v_1 \cdots, v_\ell\}$, one has $\int_0^T g(z(s))ds = \int_0^T \tilde{g}(z(s))ds$. Now, using the fact that $\tilde{g}$ is increasing and convex, $\int_0^T \tilde{g}(z(s))ds \geq \int_0^T \tilde{g}(u^*(s))ds$. Applying equality (7), we see that $\int_0^T g(z(s))ds \geq \int_0^T g(z^*(s))ds$. This means that the energy use of any admissible solution z is larger than the energy use of z^*. □

Note that z^* can be constructed in linear time, once the function u^* is given. The construction of z^* is illustrated in Figure 8.

It would be interesting to study the difference in energy consumption between the continuous case where the speed can range over the whole interval $[0, 1]$ and the case where it can take only finitely many values. By uniform convergence arguments, it should be obvious that they coincide in the limit when the maximal gap between two consecutive admissible speeds goes to zero.

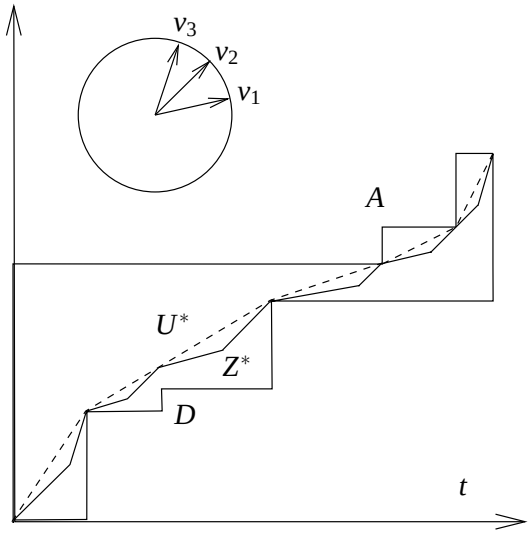

Fig. 8. The function Z^* is the integral of an optimal solution z^* when using 3 speeds, v_1, v_2 and v_3.

4.1 Minimal number of speed changes

Because the modified cost function $\tilde{g}$ is not strictly convex, there will be many different optimal solutions to Problem 4.1. Amongst them will be z^*. However, it may be possible to find an optimal solution with fewer speed changes than z^*. We now present an algorithm for doing this.

The main idea of the construction is to switch between speeds only when absolutely necessary. Suppose we are at a point $(\tau, M(\tau))$ in an interval I in which z^* uses only two speeds v_h and v_{h+1}. If the current processor speed is v_h, then the latest time we can switch to speed v_{h+1} while still meeting the constraints is $\sup\{t : M(\tau) + (t-\tau)v_h \geq \tilde{D}(t)\}$, where

$$\tilde{D}(t) := \max_{d_i \geq t}[D(d_i) - (d_i - t)v_{h+1}].$$

Similarly, if the current processor speed is v_{h+1}, then the latest time we can switch to speed v_h and still be guaranteed not to run of out work is $\sup\{t : M(\tau) + (t-\tau)v_{h+1} \leq \tilde{A}(t)\}$, where

$$\tilde{A}(t) := \min_{a_i \geq t}[A(a_i) - (a_i - t)v_h].$$

The "latest switching" algorithm in the interval I consists of alternating between the speeds v_h and v_{h+1} in the above manner. We use the following Viterbi algorithm to construct an optimal function with a minimum number of speed changes:

(1) Partition $[0, T]$ into intervals $I_1, \cdots, I_k$ such that in each I_i, the function z^* only uses the same two speeds, say v_h, v_{h+1}. We also require that the partition is the coarsest possible in the sense that the pairs of speeds used by z^* in neighboring intervals are different.

(2) In each interval $I_i = [a_i, b_i]$, calculate the following two functions $\underline{m}_i$ and $\overline{m}_i$ using the "latest switching" algorithm above. The first starts at $(a_i, z^*(a_i))$ with initial speed v_h and finishes at $(b_i, z^*(b_i))$. The second has the same start and finish points but has initial speed v_{h+1}. Record the terminal speed of both functions and their number of speed changes.

(3) Initialize $\overline{m}$ and $\underline{m}$ to be empty. Go through the intervals in reverse order, applying the following recursive procedure. At each stage, append to $\overline{m}_i$ either $\overline{m}$ or $\underline{m}$ so as to minimize the total number of speed changes, including the possible speed change at the interface. All the necessary information was calculated during the previous step. The resulting function is the new $\overline{m}$. Similarly, we append either the old $\overline{m}$ or $\underline{m}$ to $\underline{m}_i$ to form the new $\underline{m}$.

(4) We end up with two functions $\overline{m}$ and $\underline{m}$. Choose the one with fewer speed changes and call it m^*.

Remark 4.3. It is straightforward to see that the computation time of m^* given z^* is linear in the number of tasks.

THEOREM 4.4. *The function m^* constructed by this algorithm is an optimal solution to Problem 4.1 and any other optimal solution has at least as many speed changes.*

PROOF. First note that M^* and Z^* agree at the end points and use the same two speeds in each interval, I_i. Therefore, they use each speed for the same amount of time and hence use the same amount of energy. This shows that m^* is an optimal solution to Problem 4.1.

Now let m be any feasible function that uses the speeds $\{v_h : 1 \le h \le l\}$ in the same proportions as z^*. Let the maximum speed of u^* lie between v_h and v_{h+1} and consider the set of those intervals $\{I_k : k \in K\}$ in the partition where u^* lies between v_h and v_{h+1}. Let $I_i = [a_i, b_i]$ be one of these intervals. In the neighboring intervals I_{i-1} and I_{i+1}, we have $z^* < v_h$. Therefore, $Z^*(a_i) = A(a_i)$ and $Z^*(b_i) = D(b_i)$. Thus

$$\int_{a_i}^{b_i} m\,dt \ge Z^*(b_i) - Z^*(a_i). \tag{8}$$

Let τ_i' and τ_i be, respectively, the amount of time m and z^* spend at speed v_{h+1} in the interval I_i. Since the total time spend at speed v_{h+1} is the same for both m and z^*, if $\tau_i' > \tau_i$ for some $i \in K$ then to compensate there must be some $j \in K$ such that $\tau_j' < \tau_j$. However, since $\int_{a_i}^{b_i} m\,dt \le \tau_i' v_{h+1} + (b_j - a_j - \tau_j')v_h$, this would contradict (8). We conclude that $\tau_i' = \tau_i$ for all $i \in K$ and, moreover, that in each of the intervals $\{I_k : k \in K\}$ the function m uses the same two speeds as z^* for the same amount of time. By now removing these intervals and applying an inductive argument, we may extend the same conclusion to all the intervals in the partition. This provides a justification for considering each interval separately.

If m is not equal to the function m^* constructed above, then at some point m switches between speeds earlier than necessary. If delaying both this switch and the following switch in the opposite direction, then we obtain another optimal solution to Problem (4.1) which has the same or fewer speed changes as m. By doing this each time m switches too early, and choosing the delays appropriately, m may be

transformed into m^*. Therefore m^* has the minimum possible number of speed changes. □

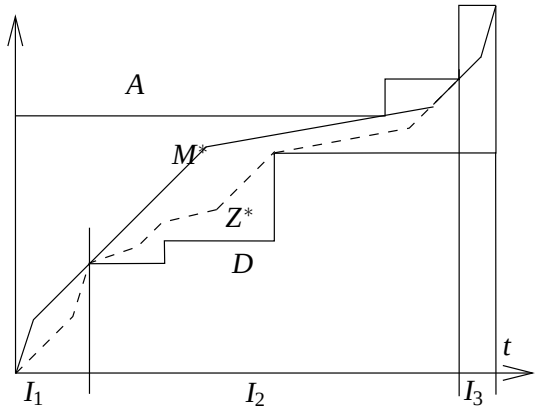

Fig. 9. An optimal solution with a minimal number of speed changes.

Figure 9 shows both integrals Z^* and M^* in the example given in Figure 8. In this case, m^* has four speed changes while z^* has ten speed changes. The function m^* was constructed by using the "latest switching" algorithm. In the interval I_3, the two speeds are v_2 and v_3, and m^* uses v_2 first. In I_2, the two speeds are v_1 and v_2. The best solution uses v_2 first. Finally in I_1, the two speeds are v_2 and v_3, and it is best is to start with v_3.

Remark 4.5. The construction of z^* and of m^* is still valid in the non-FIFO case if the optimal solution in the continuous speed case is provided (for example, using the algorithm proposed by Yao et al in [1995]). Indeed, the construction of z^* and m^* can be performed unchanged on each "critical interval".

4.2 Speed change overheads

The previous section shows how to minimize the number of speed changes while minimizing the energy consumption. This is applicable as long as the overhead for each speed change is the same. However, this is typically not the case for real circuits. The actual dynamics of speed changes depends heavily on the technology used. Here, we will consider two important cases, synchronous and asynchronous circuits, and, for each, we will provide upper and lower bounds on the optimal energy consumption as well as a speed function that remains admissible with the transition overheads.

4.2.1 *Asynchronous circuits.* For asynchronous circuits, the speed is changed by changing the voltage. The time δ required depends only on the DC-DC converter. Therefore, to switch from v_1 to v_2, the time required is $\delta = \kappa|v_1 - v_2|$, where κ is some fixed parameter (very small in all cases). In the following, we use δ_{max} to denote the maximum length of time required for any speed change: $\delta_{max} = \kappa(v_{max} - v_{min})$. Within the time interval δ, the speed $v(t)$ changes continuously

from v_1 to v_2 in a smooth manner: $v(t)$ is differentiable with a bounded derivative $v'(t) \leq C$ (the changing rate is smaller than some constant C). An example is shown on Figure 10.

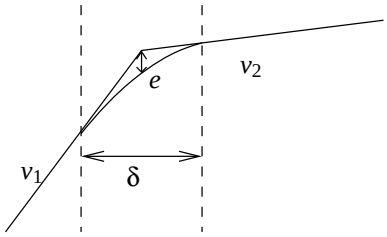

Fig. 10. Speed changes for asynchronous circuits.

Moreover in the asynchronous case, work continues (at rate $v(t)$) while the speed is changing. As for the energy spent during the change, the instantaneous consumption is of the form $g(v(t)) + \omega$. A more precise description of speed changes in asynchronous circuits can be found for instance in [Es Salhiene et al. 2003].

Call G^* the optimal total energy consumption taking into account the precise model for speed changes given above. We denote by n the number of speed changes of some admissible z and by $G(z) = \int_0^T g(z(t))dt$ the total energy consumption of z with the ideal model for speed changes (immediate changes with no overheads).

Then, with the above notation, one has the following result:

THEOREM 4.6. *For asynchronous circuits, there exists a small gap e such that if $A(t) - D(t) \geq e$ for all t then there exists an admissible speed function y such that*

$$G(m^*) \leq G^* \leq G(y) + n^*(\omega\delta_{max}) \leq G(m^*) + n^*(\omega\delta_{max}) + h\delta_{max}^2 \qquad (9)$$

where h is a constant independent of m^* computed in the proof below and n^* is the number of speed changes of m^*.

This theorem deserves several comments. First, since δ_{max} is small, this means that the ideal computations (z^* as well as m^*) are very good approximations of the actual power consumption. Second, if we go to second order, since δ_{max} is small, the term $h\delta_{max}^2$ can be neglected compared with $n^*(\omega\delta_{max})$. This means that among all ideal candidates, the one with the minimal number of speed changes (m^*) is the best up to first order. This justifies *a posteriori* the construction of m^*.

PROOF. The first inequality in (9) simply says that the ideal model for speed changes spends less energy than the realistic one because of the overheads ω.

Now, let us construct a smooth version of m^* (called $s(m^*)$) where all speed changes are smoothed as in Figure 10. The total energy consumption of $s(m^*)$ is less than that of m^* (shorter path) if one discards the overheads ω. This means that the energy consumption due to speed changes can be overlooked as long as it does not jeopardize the feasibility of the solution. In Figure 10, one can see that there exists a small quantity called e that may prevent the smooth version $s(m^*)$ from being feasible.

The computation of e goes along the following lines: since the curvature of $s(m^*)$ is always larger than some parameter C, using the intermediate value theorem twice, $e \leq C\delta^2/2$ for each speed change. Therefore, $e \leq C\delta_{max}^2/2$. The goal now is to construct a new version of m^*, called y, that stays away from the feasibility constraints by a margin at least equal to e so that its smooth version $s(y)$ will be feasible. Then, a bound on the consumption of this new speed function will be derived.

So m^* can be considered to have two types of speed changes. *Critical* speed changes are those occurring close to the borders ($M^*(t) \leq D(t) + e$ (type D) or $M^*(t) \geq A(t) - e$ (type A)). Other speed changes occur far from the feasibility borders and are not considered in the following. For example, the function m^* of Figure 9 has no critical speed changes.

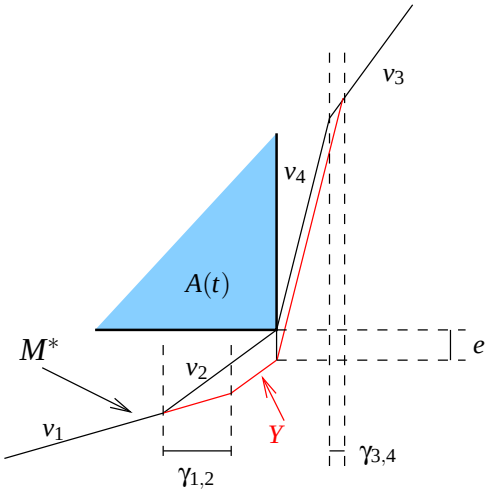

Fig. 11. The construction of $Y(t) \stackrel{\text{def}}{=} \int_0^t y(s)ds$ for asynchronous circuits at one critical speed change of type A. The construction preserves feasibility if the distance between $A(t)$ and $D(t)$ is greater than or equal to e for all t.

Figure 11 shows the construction of y around one critical speed change. We focus on a critical speed change close to the A border. Type D changes are similar and will be discussed further in the following. This speed change involves four speeds: $v_1 \leq v_2$ are the speeds of z^* before the change and $v_3 \leq v_4$ are the speeds after the change. Note that since the change occurs next to an A border, $v_1 \leq v_2 \leq v_3 \leq v_4$. The figure shows the case where v_2 is used last (just before the change) and v_4 is used first (just after the change). All other cases (v_1 used last and/or v_3 used first) are equivalent. This construction modifies the speed durations in the following way: v_1 is used longer (γ_{12} more), v_2 is used less (γ_{12} less), v_4 is used longer (γ_{34} more) and v_3 is used less (γ_{34} less). If for all t, $A(t) - D(t) \geq e$ then $Y(t)$ cannot cross $A(t)$ or $D(t)$ and the feasibility is ensured.

Direct computation shows that $\gamma_{ij} = \frac{e}{v_j - v_i}$. Therefore, the energy surplus caused

by one critical change when y is used instead of m^* is

$$\delta^2 C/2 \frac{g(v_4) - g(v_3)}{v_4 - v_3} - \frac{g(v_2) - g(v_1)}{v_2 - v_1}.$$

If one sums over all critical speed changes, one sees that terms cancel as long as critical changes are of type A. Doing the same for critical jumps of type D we are left with only two types of terms (both appearing twice): those from one change of type A to one of type D and those from one change of type D and one of type A. Note that those changes do not depend on z^* but on u^*. The first set (denoted P) corresponds to time intervals $[a, b]$ such that $U^*(a) = A(a)$ and $U^*(b) = D(b)$, while the second one (denoted Q) corresponds to time intervals where $U^*(a) = D(a)$ and $U^*(b) = A(b)$. Denote by v_{max} and v_{max-1} the maximum two speeds used by m^* and by v_{min} and v_{min+1} the minimum two speeds used by m^*. By construction, both sets have the same size (up to one) $|P - Q| \leq 1$. We get that

$$G(y) - G(m^*) \leq \delta_{max}^2 \left(|P| C \left(\frac{g(v_{max}) - g(v_{max-1})}{v_{max} - v_{max-1}} - \frac{g(v_{min+1}) - g(v_{min})}{v_{min+1} - v_{min}} \right) \right).$$

The conclusion follows when one observes that $G^* \leq G(s(y)) \leq G(y) + n^*(\omega \delta_{max})$. □

4.2.2 *The synchronous case.* In the synchronous case, the speed changes are not as simple as in the asynchronous case. The duration of a speed change depends on the voltage converter as well as on the PLL and is of the form $\delta = \kappa |v_2 - v_1| + \phi$ where ϕ may be constant or may depend on the speeds. In general κ is much larger than in the asynchronous case and ϕ is of the same order as δ.

Another difference with the asynchronous case is that, for most CPUs, no work can be done during a speed change (see Figure 12). The energy consumption can be more complicated to measure but is almost the same as if the processor was still working at the initial speed, that is, the total energy use is $g(v_1)\delta$. Again, see [Es Salhiene et al. 2003] for more hardware details.

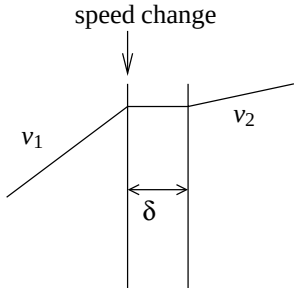

Fig. 12. The amount of work done during a speed change in a typical synchronous circuit is nil.

THEOREM 4.7. *For synchronous systems, the optimal energy consumption is bounded by*

$$G(m^*) \leq G^* \leq G(m^*) + \sum_{i \in I} \delta_i v_1^i \frac{g(v_1^i) - g(v_2^i)}{v_1^i - v_2^i},$$

where I is the set of all speed changes of m^*, and $v_1^i \geq v_2^i$ are the two speeds of the change.

Again, we make several comments. One can see that if δ_i is small, then the ideal case remains close to the real case, although the difference is larger than in the asynchronous case. Also, the construction of m^* for the minimization of the number of speed changes has the following additional property : each speed change in m^* can be mapped one to one to a subset of the speed changes in u^* with the same speeds. Therefore, the quantity $\sum_{i\in I} \delta_i v_1^i \frac{g(v_1^i)-g(v_2^i)}{v_1^i-v_2^i}$, is smaller for m^* than for u^*. This is again a justification of using m^* instead of z^* (admittedly, a weaker one than for the asynchronous case).

PROOF. Here, since speed changes do not conserve work, one has to consider all the speed changes and not just the critical ones. For each speed change i, Figure 13 shows how to choose a speed schedule y conforming to the dynamics of speed changes.

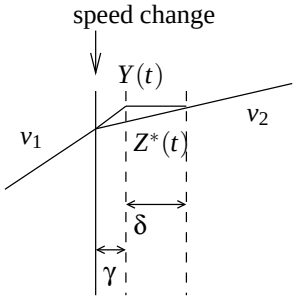

Fig. 13. Changing speeds with the constraints of synchronous circuits.

The increase in power consumption is $\gamma g(v_1^i)-(\gamma+\delta_i)g(v_2^i)$ where $\gamma = \delta_i v_1^i/(v_1^i - v_2^i)$. Now, taking into account the fact that during the change, the actual power consumption also includes $\delta_i g(v_1^i)$, one gets a total increase of $\delta_i v_1^i \frac{g(v_1^i)-g(v_2^i)}{v_1^i-v_2^i}$. □

5. EXTENSION 1: FLUID TASKS

In this section, we generalize to arbitrary functions A and D, without assuming that they are staircase functions with a finite number of discontinuities.

The fluid task model can be used to specify a minimum performance level for systems in which it is not possible to identify precisely job characteristics. Fluid tasks can also be used to approximate the curves $A(t)$ and $D(t)$ using a limited number of parameters when the number of tasks is very large but there are some regularities. For large sets of tasks, the main problem is not really the computation time of the voltage schedule which is done off-line (see paragraph 3.2) but more the amount of memory needed to store the list of successive voltages. Thus, only approximations of $A(t)$ and $D(t)$ are needed and can be very effective (by providing small set of successive voltages and a limited increase in energy consumption) when

the system possesses a limited number of work arrival patterns, each corresponding for instance to a particular functioning mode of the system.

In this section, we consider functions A and D with the following properties:

(F_1) A is a non-decreasing left-continuous function, with right and left derivatives in $\mathbb{R} \cup \{-\infty, +\infty\}$, and $A(0) = 0$.

(F_2) D is a non-decreasing right-continuous function, with right and left derivatives in $\mathbb{R} \cup \{-\infty, +\infty\}$, $D(0) = 0$, and $D \geq A$.

Problems 2.1 and 2.3 remain unchanged: find an integrable u that minimizes the energy expended between time 0 and time T, while satisfying constraints (3) and (4), and respectively (5).

The solution is also the same: the optimal speed schedule for the processor is given by the shortest path bounded by A and D from point $(0, 0)$ to point $(T, D(T))$. This can be seen from the proof of Theorem 2.5, which also works for arbitrary functions A and D. However, now the optimal solution U^* is not necessarily piecewise affine, as shown by the example of Figure 14.

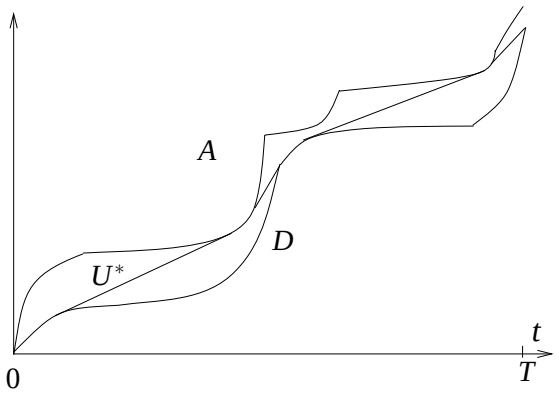

Fig. 14. the optimal solution with arbitrary function A and D.

The computational issues become a real concern here because the functions A and D can be arbitrarily difficult to code in a computer program. However, if both A and D are piecewise polynomial (of degree k), then the computation of U^* only involves solving polynomial equations of degree k. This can be done with arbitrary precision using symbolic computation tools based on Schur polynomials and Gröbner basis. Here, the complexity of the algorithm is at least exponential in k.

5.1 Finite number of speeds

The solution of Problem 4.1 in this more general framework requires taking some precautions. We need to add technical assumptions on the functions A and D to ensure that there are admissible solutions. One way of doing this is to assume the following:

(F_3) $\exists\, \delta > 0, \quad \forall\, 0 \le a \le b \le T, \quad \int_a^b A(s) - D(s)ds \ge \delta(b-a)$,

(F_4) there exists a finite number of points x between 0 and T such that $A(x) = D(x)$, and $\forall\ x$ s.t. $A(x) = D(x), \exists\ v, w \in \{v_1, \cdots v_\ell\}$ s.t. $\frac{dA}{dt_+}(x) \ge v, \frac{dD}{dt_+}(x) \le v$, and $\frac{dA}{dt_-}(x) \le w, \frac{dD}{dt_-}(x) \ge w$.

It should be obvious that, if assumption (F_4) is not satisfied, then Problem 4.1 does not have any admissible solution since at time 0 any choice of the initial speed would break either constraint 3 or 4. Assumption (F_3) adds the requirement that, whenever A is strictly above D, there is still enough space between A and D for some admissible solution using a finite number of speeds.

To find the optimal solution to Problem 4.1, one must construct a finite sequence of functions, (y_n) using the following procedure.

Let $x_1 = 0, x_2, \cdots, x_h = T$ be the points where A and D meet. Partition each interval $[x_i, x_{i+1}]$ of length $T_i \stackrel{\text{def}}{=} x_{i+1} - x_i$ into n sub-intervals of the same size (T_i/n). Here is a way to construct the function y_n.

At step k, we construct the function $y_n(t)$ over the kth interval, namely $I_k \stackrel{\text{def}}{=} (x_i + kT_i/n, x_i + kT_i/n + T_i/n]$. There exists h such that

$$v_h T_i/n \le \int_{x_i+kT_i/n}^{x_i+(k+1)T_i/n} u^*(s)ds \le v_{h+1} T_i/n.$$

There exists $\alpha_k \in (0, 1]$ such that

$$u_k^* \stackrel{\text{def}}{=} n/T \int_{x_i+kT_i/n}^{x_i+(k+1)T_i/n} u^*(s)ds = \Big(\alpha_k v_h + (1-\alpha_k)v_{h+1}\Big)$$

At this point, we have two options in defining the function y_n over the interval $I_k = (kT_i/n, kT_i/n+T_i/n]$. At least one of them will be admissible when n becomes large enough.

First alternative: $y_n(t) = v_h$ over the interval $I_k = (x_i + kT_i/n, x_i + \alpha_k T_i/n + (1-\alpha_k)(kT_i/n + T_i/n)]$ and $y_n(t) = v_{h+1}$ over the interval $(x_i + \alpha_k T_i/n + (1-\alpha_k)(kT_i/n + T_i/n), x_i + kT_i/n + T_i/n]$.

Second alternative: $y_n(t) = v_{h+1}$ over the interval $(x_i + kT_i/n, x_i + \alpha_k(kT_i/n + T_i/n) + (1-\alpha_k)T_i/n]$ and $y_n(t) = v_h$ over the interval $(x_i + \alpha_k(kT_i/n + T_i/n) + (1-\alpha_k)T_i/n, x_i + kT_i/n + T_i/n]$.

Note that because of assumption F_4, this is locally admissible at the extreme points of the intervals.

THEOREM 5.1. *The function $y^* = y_n$ is an optimal solution of Problem 4.1 for the smallest n such that y_n is admissible.*

PROOF. First note that the integral of y_n converges to U^* pointwise as n goes to infinity. Using assumption F_3, this implies that the function y_n is admissible if n is large enough. Second, using the assumption on the range of the u_i's, for any $v_1 \le u < v_\ell$, there exists $i(u)$ such that $v_{i(u)} \le u < v_{i(u)+1}$. We then define the coefficient α_u such that $u = \alpha_u v_{i(u)} + (1-\alpha_u)v_{i(u)+1}$. We use the real function $\tilde{g}(u)$ as in Section 4.

The next part of the proof consists of showing that

$$\int_0^T g(y^*(s))ds = \int_0^T \tilde{g}(u^*)ds. \tag{10}$$

Using the definition of y^*,

$$\begin{aligned}\int_0^T g(y^*(s))ds &= \sum_{k=0}^{N} \int_{I_k} g(y_n(s))ds \\ &= \sum_{k=0}^{N} \int_{I_k} \tilde{g}(u_k^*)ds.\end{aligned}$$

Since the function $\tilde{g}$ is affine on all the intervals I_k, we have $\sum_{k=0}^{N} \int_{I_k} \tilde{g}(u_k^*)ds = \int_0^T \tilde{g}(u^*)ds$.

The last part of the proof resembles the proof in the case of discrete tasks. Let u be any admissible solution of Problem 4.1. Then, $u(t) \in \{v_1 \cdots, v_\ell\}$. Since the function f coincides with g over $\{v_1 \cdots, v_\ell\}$, one has $\int_0^T g(u(s))ds = \int_0^T \tilde{g}(u(s))ds$. Now using the fact that f is increasing and convex, $\int_0^T \tilde{g}(u(s))ds \geq \int_0^T \tilde{g}(u^*(s))ds$. Finally, using Equality (10) shows that $\int_0^T g(u(s))ds \geq \int_0^T g(y_n(s))ds$. This means that the cost of any admissible solution is larger than the cost of y_n. □

6. EXTENSION 2: NON-CONVEX ENERGY FUNCTIONS

Here, we consider the case where the function g, which gives the instantaneous energy consumption, is not convex and increasing. This is occurs for example when the static power dissipated by the processor is not negligible. In this case, the typical behavior of g is displayed in Figure 15.

To be as general as possible, we keep the assumption that A and D are fluid functions satisfying F_1, F_2, F_3, and F_4. Everything that follows is also valid for staircase functions.

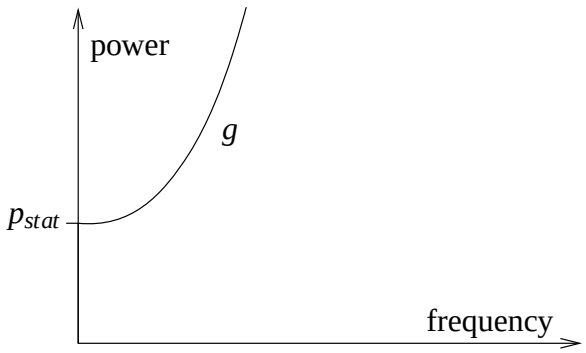

Fig. 15. Example of a non-convex energy consumption function. The energy consumed at frequency 0 is not nil due to the static power.

We assume that the function g is semi-continuous but not necessarily convex or increasing. For technical reasons, we will further assume that g has a finite number of inflexion points.

In this case, the optimal solution v^* of Problem 2.1 depends on g. Here is a way to construct v^*.

The first step is to construct the convex hull h of g. Since $g(0) = 0$ and $g(x) \geq 0, \forall x \geq 0$, we have that h is an increasing convex function. Let $\mathcal{C}$ be the set of points where g and h coincide: $\mathcal{C} \stackrel{\text{def}}{=} \{x \in \mathbb{R}_+ \, s.t. \, h(x) = g(x)\}$. Let $\mathcal{B}$ be the complement: $\mathcal{B} \stackrel{\text{def}}{=} \{x \in \mathbb{R}_+ \, s.t. \, h(x) \neq g(x)\}$. Using the assumption on the inflexion points of g, the set $\mathcal{B}$ is composed of a finite number of intervals. Note that the function h is affine on $\mathcal{B}$. For each $x \in \mathcal{B}$, we define two points in C surrounding x: $\overline{m}(x) \stackrel{\text{def}}{=} \inf\{s \in \mathcal{C} \, s.t. \, s \geq x\}$ and $\underline{m}(x) \stackrel{\text{def}}{=} \sup\{s \in \mathcal{C} s.t. s \leq x\}$.

The second step is to solve Problem 2.1 using h instead of g as the instantaneous cost. Since h is convex and increasing, we get, as before, the shortest path U^* bounded by A and D.

The third step is to construct a set of functions, $v_n, n \in \mathbb{N}$ as follows.
If $u^*(t) \in \mathcal{C}$, then $v_n(t) = u^*(t)$.
If $u^*(t) \in \mathcal{B}$, then there exists an interval I, containing t and maximal for inclusion, over which $u^* \in [\underline{m}(u^*(t)), \overline{m}(u^*(t))]$. We partition the interval I into n sub-intervals, each of size $|I|/n$. In each such interval, say $[t_1, t_2)$, the average value of u^* over this interval is $\mu \stackrel{\text{def}}{=} \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} u^*(t)dt$ and the coefficient $\alpha \stackrel{\text{def}}{=} \frac{\mu - \underline{m}(\mu)}{\overline{m}(\mu) - \underline{m}(\mu)}$. Now, $v_n(t) = \underline{m}(\mu)$ over $[t_1, t_1 + (1-\alpha)(t_2 - t_1))$ and $v_n(t) = \overline{m}(\mu)$ over $[t_1 + (1 - \alpha)(t_2 - t_1), t_2)$.

The final step is to take $v^*(t) = v_n(t) \quad \forall t \in [0, T]$, for some n large enough that $v^*(t)$ is admissible.

THEOREM 6.1. *The function v^* is the optimal solution to Problem 2.1.*

PROOF. (sketch) The proof is similar to the proof given in Section 5.1. The first thing to notice is that, since the intervals used to define the functions v_n get smaller and smaller, the integrals of these functions converge point-wise towards $U^*(t)$ as n grows. Therefore since $A > D$, there exists a finite n such that v_n satisfies constraints 3 and 4 and so is admissible.

The second key point in the proof is to notice that , since h is affine over $\mathcal{B}$, $\int_{t_1}^{t_2} h(v_n)dt = h(\mu)(t_2 - t_1)$. By integrating over the entire time range, we see that

$$\int_0^T h(v^*)dt = \int_0^T h(u^*)dt. \tag{11}$$

To finish the proof, take u any admissible solution for problem 2.1.

$$\int_0^T g(u(t))dt \geq \int_0^T h(u(t))dt, \tag{12}$$

$$\geq \int_0^T h(u^*(t))dt, \tag{13}$$

$$= \int_0^T h(v^*(t))dt, \tag{14}$$

$$= \int_0^T g(v^*(t))dt. \tag{15}$$

Inequality (12) comes from the fact that $g \geq h$; inequality (13) comes from the fact that u^* is the optimal solution for the convex cost h; equality (14) is the same as (11); and Equality (15) comes from the fact that $v^*(t) \in \mathcal{C}$ for all t and that $g = h$ over $\mathcal{C}$. □

6.1 Finite set of speeds

The case where the processor can only take a finite number of speeds $\{v_1, \cdots, v_\ell\}$ is much easier to handle. First, construct the convex hull h of the finite set of points $\{(v_1, g(v_1)), \cdots, (v_\ell, g(v_\ell))\}$. Then remove all the speeds which do not belong to the convex hull from the set of allowed speeds. Finally, solve Problem 4.1 as in Section 4 with the reduced set of speeds. This gives the optimal solution.

7. CONCLUSION

In this study, we presented a new approach to determine the optimal speed schedule of a set of independent tasks subject to FIFO real-time constraints. The immediate advantage of our proposal is that it can be implemented in linear time if the functions A and D are given, in $O(N \log(N))$ otherwise. The algorithm works both when the range of possible speeds is continuous and when it is discrete. In the latter case, we provide an algorithm that ensures the minimum number of speed changes and thus minimizes the speed changing overhead. The results have been extended to fluid tasks and non-convex cost functions.

It has been shown that, in the context of this study, the problem of minimizing energy consumption is equivalent to a shortest path problem. This observation may lead to some new advances in the field of dynamic voltage scaling.

We are currently investigating the on-line case with probabilistic assumptions on the workload arrival. Two distinct objectives are considered: minimizing the expected energy consumption and minimizing the worst-case energy consumption.

REFERENCES

Aydin, H., R., M., Mossè, D., and P., M.-A. 2001. Dynamic and agressive scheduling techniques for power aware real-time systems. In *Real-Time Systems Symposium*. 95–105.

Boissonnat, J. and Yvinec, M. 1995. *Géométrie Algorithmique*. Ediscience International.

Es Salhiene, M., Fesquet, L., and Renaudin, M. 2003. Adaptation dynamique de la puissance des systèmes embarqués: les systèmes asynchrones surclassent les systèmes synchrones. In *journées d'études Faible Tension - Faible Consommation (FTFC'03)*. 51–58.

Gruian, F. 2001. On energy reduction in hard real-time systems containing tasks with stochastic execution times. In *IEEE Workshop on Power Management for Real-Time and Embedded Systems*. 11–16.

Gruian, F. 2002. Energy-centric scheduling for real-time systems. Ph.D. thesis, Lund Institute of Technology, Sweden.

Hong, I., Potkonjak, M., and Srivastava, M. 1998. On-line scheduling of hard real-time tasks on variable voltage processor. In *International Conference on Computer Design*. 653–656.

Ishihara, T. and Yasuura, H. 1998. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design*. 197–202.

Jackson, J. 1955. Scheduling a production line to minimize maximum tardiness. Tech. rep., University of California. Report 43.

Lorch, J. and Smith, A. 2001. Improving dynamic voltage scaling algorithms with pace. In *ACM SIGMETRICS 2001 Conference*. 50–61.

Mossè, D., Aydin, H., Childers, B., and Melhem, R. 2000. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compiler and Operating Systems for Low-Power*.

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

Shin, D., Kim, J., and Lee, S. 2001. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design & Test of Computers 18,* 2, 20–30.

Shin, Y. and Choi, K. 1999. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference*. 134–139.

Stankovic, J., Spuri, M., Ramamritham, K., and Buttazo, G. 1998. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publisher.

Yao, F. 2003. Complexity of the Yao Demers Shenker algorithm. Private communication.

Yao, F., Demers, A., and Shenker, S. 1995. A scheduling model for reduced CPU energy. In *Proceedings of lEEE Annual Foundations of Computer Science*. 374–382.

Yun, H.-S. and Kim, J. 2003. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Transactions on Embedded Computing Systems 2,* 3 (Aug.), 393–430.

Zhang, F. and Chanson, S. 2002. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *Real-Time Systems Symposium*. 235–245.

Received November 2003; revised April 2004; accepted September 2004