

# Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors

Flavius Gruian

Department of Computer Science, Lund University

Box 118

S-221 00 Lund, Sweden

Tel.: +46 046 2224673

e-mail: [Flavius.Gruian@cs.lth.se](mailto:Flavius.Gruian@cs.lth.se)

## ABSTRACT

The work presented in this paper addresses scheduling for reduced energy of hard real-time tasks with fixed priorities assigned in a rate monotonic or deadline monotonic manner. The approach we describe can be exclusively implemented in the RTOS. It targets energy consumption reduction by using both on-line and off-line decisions, taken both at task level and at task-set level. We consider sets of independent tasks running on processors with dynamic voltage supplies (DVS). Taking into account the real behavior of a real-time system, which is often better than the worst case, our methods employ stochastic data to derive energy efficient schedules. The experimental results show that our approach achieves more important energy reductions than other policies from the same class.

## Keywords

Low-energy, hard real-time, RTOS, scheduling

## 1. INTRODUCTION

Low energy consumption is today an increasingly important design requirement for digital systems, with impact on operating time, on system cost, and, of no lesser importance, on the environment. Reducing power and energy dissipation has long been addressed by several research groups, at different abstraction levels. We focus here on methods applicable at system-level, where the system to be designed is specified as an abstract set of tasks. Selecting the right architecture has been shown to have a great influence on the system energy consumption [4,5]. Recently, with the advent of dynamic voltage supply (DVS) processors [2,22,25], highly flexible systems can be designed, while still taking advantage of supply voltage scaling to reduce the energy consumption. Since the supply voltage has a direct impact on processor speed, classic task scheduling and supply voltage selection have to be addressed together. Scheduling offers thus yet another level of possibilities for achieving energy/power efficient systems, especially when the system architecture is fixed or the system exhibits a very dynamic behavior. For such dynamic systems, various power management techniques exist and are reviewed for example in [1,17]. Yet, these mainly target soft

real-time systems, where deadlines can be missed if the Quality of Service is kept. Several scheduling techniques for soft real-time tasks, running on DVS processors have already been described [3,18,19,23]. Energy reductions can be achieved even in hard real-time systems, where no deadline can be missed, as shown in [6,7,10,20,24]. In this paper, we also focus on hard real-time scheduling techniques, where every deadline has to be met.

Task level voltage scheduling decisions can reduce even further the energy consumption. Some of these intra-task scheduling methods use several re-scheduling points inside a task, and are usually compiler assisted [11,16,21]. Alternatively, fixing the schedule before the task starts executing as in [6,7,8] eliminates the internal scheduling overhead, but with possible affects on energy reduction. Statistics can be used to take full advantage of the dynamic behavior of the system, both at task level [16] and at task-set level [24]. In our approach we employ stochastic data to derive efficient voltage schedules without the overhead of intra-task re-scheduling.

The rest of the paper is organized as follows. In section 2 we describe our hard real-time scheduling strategy, pointing out the related work for each decision we make. Section 3 contains several experimental results conducted both on real life examples and on randomly generated, large task sets. Finally, we present our conclusions in section 4.

## 2. RT SCHEDULING FOR LOW-ENERGY

In the work described here, we address independent tasks running on a single processor. The processor has variable speed (supply voltage and energy) adjustable at runtime. The tasks arrive with given periods and have to be executed before certain deadlines. The priorities are fixed, assigned in a rate-monotonic (RM) or deadline monotonic (DM) manner [14]. The runtime scheduling also operates as in RM/DM scheduling with the difference that each task instance is assigned a maximum allowed execution time. The scheduling strategies we adopt at task-level are presented in sub-section 2.1. The allowed execution time are influenced by task group level decisions, taken both off-line and on-line. The off-line phase is presented in sub-section 2.2 and the on-line phase in sub-section 2.3. Sub-section 2.3 also contains a proof that our scheduling method keeps the response times from the original RM/DM scheduling, and thus does not affect the feasibility of the schedule.

### 2.1 Task-level Scheduling Decisions

Task-level voltage scheduling has captured the attention of the research community rather recently [8]. Fine grain scheduling, where several re-scheduling points are used inside a task were pre-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or fee.

*ISLPED'01*, August 6-7, 2001, Huntington Beach, California, USA.

Copyright 2001 ACM 1-58113-371-5/01/0008...\$5.00.

sented in [11,16]. In [16] statistical data is used to improve the task level schedule, by slowing down different regions of a task according to their average execution time. Our approach produces voltage schedules only when a task starts executing, while using stochastic data more aggressively both at task level and task-set level. At task level we generate voltage schedules that are correlated with the task execution length probability distribution. For task-set level scheduling decisions see sub-section 2.3.

In our model a task  $\tau_i$  can be executed in phases, at different available voltages, depending on its allowed execution time  $A_i$ . The ideal case states that the most energy is saved when the processor uses the voltage for which the task exactly covers its allowed execution time. This corresponds to an ideal voltage which may not overlap with the available voltages. A close to optimal solution is to execute the task in two phases at two of the available voltages. These two voltages are the ones bounding the ideal voltage [6,8].

An important observation is that tasks may finish, and in many cases do finish, before their worst case execution time (WCET). Therefore it makes sense to execute first at a low voltage and accelerate the execution, instead of executing at high voltage first and decelerate. In this manner, if a task instance is not the worst case, one skips executing high voltage (and power eager) regions.

In the following we will distinguish between three modes of execution for a task, as depicted in Figure 1. The ideal case (mode 1) is when the actual execution pattern (the number of clock cycles) becomes known when the task arrives. We can stretch then the actual execution time of the task to exactly fill the allowed time. This mode requires rather accurate execution pattern estimates, depending on the input data, and therefore is rarely achievable in practice. The second mode (mode 2) is the WCE stretching - the voltage schedule for the task is determined as if the task will exhibit its worst case behavior. These two modes use at most two voltage regions, and therefore at most one DC-DC switch. The third mode (mode 3), described in more detail next, uses stochastic data to build a multiple voltage schedule. The purpose for using stochastic data is to minimize the average case energy consumption. Note that the voltage schedules in all these three modes are decided at a task instance arrival. Unlike in [11,21] no rescheduling is done while the task is executing. The only overhead during task execution is the one given by the changes in the supply voltage. For instance, the lpARM processor [2] needs at most 70 $\mu$ s to switch from 1.2 to 3.8V. For closer voltage levels, the switch occurs faster. Depending on the actual task execution time, this delay may have some impact on the schedule. The same goes for the energy lost during the DC-DC switch. Although our discussion does not cover these, the methods

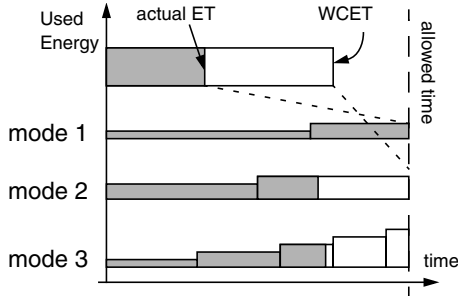


Figure 1. Voltage scheduling modes for tasks: 1) ideal schedule, 2) WCET oriented schedule, 3) stochastic schedule.

presented here can be adapted to accommodate both the DC-DC delay and energy loss whenever the actual processor requires it.

The stochastic voltage schedule (mode 3 in Figure 1) for a task is obtained using the probability distribution of the execution pattern for a task (the number of clock cycles used). This probability distribution can be obtained off-line, via simulation, or built and improved at runtime. Let us denote by  $X$  the random variable associated with the number of clock cycles used by a task instance. We will use the cumulative density of probability function,  $cdf_x$ , associated with the variable  $X$ ,  $cdf_x = P(X \leq x)$ . This function reflects the probability that a task instance finishes before a certain number of clock cycles. If  $WX$  is the worst case number of clock cycles,  $cdf_{WX} = 1$ . Deciding a voltage schedule for a task, means that for every clock cycle up to  $WX$  we decide a specific voltage level (and processor speed). Each cycle  $y$ , depending on the voltage adopted, will consume a specific energy,  $e_y$ . But each of these cycles are executed with a certain probability, so in average the energy consumed by cycle  $y$  can be computed as  $(1 - cdf_y) \cdot e_y$ . To obtain the average energy for the whole task, we have to consider all the cycles up to  $WX$ :

$$\bar{E} = \sum_{0 < y \leq WX} (1 - cdf_y) \cdot e_y \quad (1)$$

This is the value we want to minimize by choosing appropriate voltage levels for each cycle. Since  $WX$  may be a large number in practice, in our implementation we group several consecutive clock cycles into equal size groups. For the sake of brevity and clarity we describe here only the simpler case, when the voltage levels are decided clock cycle by clock cycle.

A task has to complete its execution during an allowed execution time,  $A$ . If we denote the clock length associated to clock cycle  $y$  by  $k_y$ , this constraint can be written as:

$$\sum_{0 < y \leq WX} k_y \leq A \quad (2)$$

The clock cycle length  $k$  dependency on the supply voltage  $V$  and threshold voltage  $V_T$  is according to:  $k \sim V / (V - V_T)^\beta$  where  $\beta$  is the velocity saturation index. If  $V_T$  is small enough or we use a variable threshold technology [22], this dependency is simplified to:  $k \sim V^{(1-\beta)}$ . The clock cycle energy  $e$  is directly dependent on the square of the supply voltage as in:  $e \sim V^2$  [6]. Eliminating  $V$  from the last two expressions we obtain the dependency between the clock cycle energy and length:

$$e \sim 1/k^{\left(\frac{2}{\beta-1}\right)} \quad (3)$$

For clarity we will bound now  $\beta = 2$ , but the rest of the calculus can be carried out for any other reasonable value of  $\beta$ . If we substitute (3) in (1), we obtain:

$$\bar{E} \sim \sum_{0 < y \leq WX} \frac{(1 - cdf_y)}{k_y^2} \quad (4)$$

which is the value to be minimized. By mathematical induction one can prove that the right hand side of (4) has a lower bound (using also (2)):

$$LB = \left( \frac{\sum_{0 < y \leq WX} \sqrt{1 - cdf_y}}{\sum_{0 < y \leq WX} k_y} \right)^2 \geq \frac{1}{A^2} \cdot \left( \sum_{0 < y \leq WX} \sqrt{1 - cdf_y} \right)^2 \quad (5)$$

This lower bound can only be obtained if and only if:

$$k_y = A \cdot (\sqrt{1 - cdf_y}) / \left( \sum_{0 < y \leq WX} \sqrt{1 - cdf_y} \right) \quad (6)$$

These are the optimal values for the clock cycle length in each clock cycle up to  $WX$ . In practice these values may not overlap with the available clock lengths so they have to be converted to real clock cycles. This conversion is done in a similar way to deriving a dual level voltage schedule from an ideal one [6,8]. We find the two bounding available clock cycles  $CK_i < k_y \leq CK_{i+1}$  and distribute the work of the ideal cycle in two such that  $k_y = \Delta w_i \cdot CK_i + (1 - \Delta w_i) \cdot CK_{i+1}$ , where  $\Delta w_i$  is the work given to  $CK_i$  and the rest is the work given to  $CK_{i+1}$ . Thus, each cycle in the task will distribute its work between two of the several available clock lengths. Finally, the accumulated work loads for each available clock cycle is rounded to integers, since one can only execute full clock cycles.

Note that the coefficient of  $A$  in (6) can be computed off-line or, if the probability distribution is built at runtime, on-line from time to time. Therefore, the on-line computational complexity for obtaining the stochastic voltage schedule is given by the steps subsequent to (6). One has to compute the ideal clock cycle for each of the  $WX$  clock cycles. Finding the bounding clock cycles takes logarithmic time of the number of voltage levels,  $N_v$ . This gives a complexity of  $O(WX \cdot \log N_v)$ .

Two examples of stochastic voltage schedules are given in Figure 2. We assumed a normal probability distribution with the mean of 70 cycles, and standard deviation of 10.  $WX$  is 100. Assuming we only have four available clock frequencies  $f, f/2, f/3,$  and  $f/4$ , we give two voltage schedules obtained for two different values of the allowed execution time. The schedules are given in number of clock cycles executed at each available frequency. The allowed execution time is reported in percentage of the time needed for executing the worst case behavior ( $WX$ ) at the highest clock frequency ( $f$ ). Some experimental results on how stochastic voltage schedule contribute at saving energy are presented in section 3.

## 2.2 Off-line Task Stretching

The scheduling condition proposed by Liu and Layland [14] is a sufficient one and covers the worst possible case for the task group characteristics. Yet, an exact analysis as proposed in [13] may reveal possibilities for stretching tasks and still keeping the deadlines. Based on this, [20] describes a method to compute the maximum required frequency for a task set (or the minimum stretching factor). In similar way, we go further and compute minimal stretching factors  $\{\alpha_i\}_{1 \leq i \leq n}$  for each task  $\tau_i$  in the task group  $\{\tau_i\}_{1 \leq i \leq n}$ . A task is defined by the triple  $\tau_i = (C_i, T_i, D_i)$  composed of the WCET, period and deadline for task  $\tau_i$ . Note that throughout the paper  $C_i$  refers to the worst case execution pattern  $WX$  running at the fastest clock frequency. We

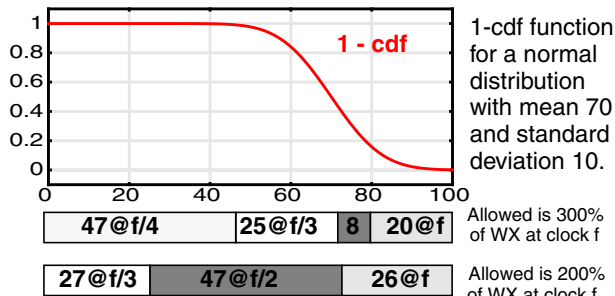


Figure 2. Two stochastic voltage schedules for a task with normal distribution execution time and worst case behavior of 100 cycles

consider that the tasks in the group are indexed according to their priority, computed as in RMS.

We compute the stretching factors in an iterative manner, from the higher to the lower priority tasks. An index  $q$  points to the latest task which has been assigned a stretching factor. Initially,  $q = 0$ . Each of the tasks  $\tau_i, q < i \leq n$  has to be executed before one of its scheduling points  $S_i$  as defined in [13]:  $S_i = \{kT_j | 1 \leq j \leq i; 1 \leq k \leq \lfloor T_i/T_j \rfloor\}$ , if  $T_i = D_i$ . If  $T_i \neq D_i$ , we only need to change the set of scheduling points according to  $S_i' = \{t | (t \in S_i) \wedge (t < D_i)\} \cup \{D_i\}$ . For each of this scheduling points  $S_{ij} \in S_i$ , task  $\tau_i$  exactly meets its deadline if:

$$\sum_{1 \leq r \leq q} \alpha_r C_r \cdot \left\lceil \frac{S_{ij}}{T_r} \right\rceil + \alpha_{ij} \cdot \sum_{q < p \leq i} C_p \cdot \left\lceil \frac{S_{ij}}{T_p} \right\rceil = S_{ij} \quad (7)$$

Note that for the tasks which already have assigned a stretching factor we used that one,  $\alpha_r$ , while for the rest of the tasks we assumed they will all use the same and yet to be computed stretching factor,  $\alpha_{ij}$ , which is dependent on the scheduling point. For the task  $\tau_i$  the best scheduling choice, from the energy point of view, is the largest of its  $\alpha_{ij}$ . At the same time, from (7), this has to be the equal for all tasks  $\tau_i, q < i \leq n$ . There is a task with index  $m$  for which its best stretching factor is the smallest among all other tasks:  $\max(\alpha_{mj}) = \min(\max(\alpha_{ij}))$ . Note that this is not necessarily the last task,  $n$ . If  $q^i = 0^j$ , this task sets the minimal clock frequency as computed in [20]. Having the index  $m$ , all tasks between  $q$  and  $m$  can be at most stretched (equally) by the stretching factor of  $m$ . Thus, we assign them stretching factors as  $\alpha_r = \max(\alpha_{mj}), q < r \leq m$ . With this an iteration of the algorithm for finding the stretching factors is complete. The next iteration then proceeds for  $q = m$ . Finally the process ends when  $q$  reaches  $n$ , meaning all tasks have been given their own off-line stretching factors.

An example is given in Table 1. Note that tasks 3 and 4 can be stretched off-line more than 1 and 2, while 5 has the largest stretching factor. The processor utilization changes from 0.687 to 0.994. We use the utilization after off-line stretching in computing the energy reduction upper bound in our experiments. For  $T_i > D_i$ , the difference between the stretching factors grows.

Table 1: Numerical Example for Off-line Stretching

No.	Task $\tau$		Off-line Stretching factor $\alpha$	
	WCET (C)	Period (T)	value	iterations needed
1	1	5	1.428	1
2	5	11	1.428	1
3	1	45	1.785	2
4	1	130	1.785	2
5	1	370	2.357	3

## 2.3 On-line Slack Distribution

At runtime it is important to use the variations in execution length of the various task instances to be able to stretch other tasks and thus consume less energy. In [20] the only situation when a task is stretched is when it is the only one running and has enough time until the next task arrives. In all other situations tasks are executed at the speed dictated by the off-line analysis. In [11] tasks are

stretch at their WCET at runtime, independent of other tasks, using several checking/re-scheduling points during a task instance. The work in [10] uses only two voltage levels. The slack produced by finishing a task early is entirely used to run the processor at the low voltage. As soon as this slack is consumed, the task starts running at high voltage. Our method is perhaps most resemblant to the optimal scheduling method OPASTS presented in [7]. Yet, OPASTS performs analysis over task hyperperiods, which may lead to working on a huge number of task instances for certain task sets. Our method keeps a low and the same computational complexity, regardless of the task set characteristics.

We describe next our strategy for slack distribution. In short, an early finishing task may pass on its unused processor time for any of the tasks executing next. But this time slack can not be used by any task at any time since deadlines have to be met. We solve this by considering several levels of slacks, with different priorities, as in the slack stealing algorithm [12]. If the tasks in the task set  $\{\tau_i = (C_i, T_i, D_i)\}_{1 \leq i \leq n}$  have  $m$  different priorities, we use  $m$  levels of slacks  $\{S_j\}_{1 \leq j \leq m}$ . Without great loss of generality consider that the tasks have different priorities,  $m = n$ . The slack in each level is a cumulative value, the sum of the unused processor times remaining from the tasks with higher priority. The invariant describing the state of the slacks in every level, at any time is given by (10). Initially, all level slacks  $S_j$  are set to zero. To maintain the relation between slack levels, the levels are managed at runtime as follows:

- whenever an instance  $k$  of a task  $\tau_i$  with priority  $i$  starts executing, it can use an arbitrary part  $\Delta C_i^k$  of the slack available at level  $i$ ,  $S_i$ . So the allowed execution time for task  $\tau_i$  will be:  $A_i^k = C_i + \Delta C_i^k$ . The remaining slack from level  $i$  will degrade into level  $i+1$  slack. Each level slack will be updated according to:

$$S_j' = \begin{cases} 0, & j \leq i \\ S_j - \Delta C_i^k, & j > i \end{cases} \quad (8)$$

- whenever a task instance finishes its execution, it will generate some slack if it finishes before its allowed time. If  $E_{ik}^k$  is the actual execution time, the generated slack is  $\Delta A_i^k = A_i^k - E_{ik}^k$ . This slack can be used by the lower priority tasks. In this case the level slacks are updated according to:

$$S_j'' = \begin{cases} S_j, & j \leq i \\ S_j + \Delta A_i^k, & j > i \end{cases} \quad (9)$$

- idle processor times are subtracted for all slacks. This ensures that the critical instance from the classic RM analysis remains the same.

The computational complexity required by the on-line method is linearly dependent to the number of slack levels.

Note that task instances can only use slack generated from higher priority tasks and produce low priority slack. We call this slack degradation. Whenever the lowest priority task starts executing, all level slacks are reset. Note also that not necessarily all slack at one level is used by a single task. Various methods can be used, but we mention here only the two we used in our experiments:

- **Greedy:** the task gets all the slack available for its level:  $\Delta C_i^k = S_i$
- **Mean proportional:** we consider the mean execution time  $\mu_i$  for each task instances waiting to execute (in the ready queue). The slack is proportionally distributed according to these:  $\Delta C_i^k = S_i \cdot \mu_i / \left( \sum_{j \in \text{ReadyQ}} \mu_j \right)$

The strategy of managing the slack we just described allows us to keep the critical instance response time for all tasks, as we prove next. The response time  $R_i(t)$  for task  $\tau_i$  is computed as  $R_i(t) = A_i + I_i(t)$ , where  $A_i$  is its allowed execution time, as before, and  $I_i(t)$  is the interference from the other tasks. From the managing strategy given before, the cumulated slack on each level, at a certain time  $t$  is of form:

$$S_i(t) = S_{i-1}(t) - \sum_k \Delta C_{i-1}^k + \sum_k \Delta A_{i-1}^k, \quad k = \left\lceil \frac{t}{T_{i-1}} \right\rceil \quad (10)$$

The slack of level  $i$  is composed of all slack from level  $i-1$ , less the slack used by the instances of tasks with priority  $i-1$  but plus all the slack generated by these. The number of instances executed,  $k$ , is determined by the task period. Note that  $S_j$  is always zero. Eliminating the iteration in the previous formula:

$$S_i(t) = \sum_{j=1}^{j < i} \left( \sum_k \Delta A_j^k - \sum_k \Delta C_j^k \right) \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (11)$$

The task with the highest priority will never receive slack and therefore,  $\Delta C_1^k = 0$ .

The interference from the high priority tasks is the time used to execute all arrived instances of these high priority tasks:

$$I_i(t) = \sum_{j=1}^{j < i} \sum_k E_j^k \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (12)$$

With the notations from the slack managing algorithm  $E_j^k = A_j^k - \Delta A_j^k = C_j + \Delta C_j^k - \Delta A_j^k$ . Introducing this in (12):

$$I_i(t) = \sum_{j=1}^{j < i} \sum_k (C_j + \Delta C_j^k - \Delta A_j^k) \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (13)$$

The last two terms in the sum are actually giving the slack of level  $i$ , as in (11), so we can re-write (13) as:

$$I_i(t) = \sum_{j=1}^{j < i} k C_j - S_i(t) \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (14)$$

Note that the maximal response time for a task is obtained when it uses all the slack available at its level:  $R_i(t) = C_i + I_i(t) + S_i(t)$ . From the last two equations:

$$R_i(t) = C_i + \sum_{j=1}^{j < i} \left\lceil \frac{t}{T_j} \right\rceil C_j \quad (15)$$

which is exactly the response time when all tasks execute at WCET. Thus, if the RM analysis decides that a task set is schedulable, it remains valid when using our on-line policy.

In our implementation we additionally used a method similar to the on-line method presented in [20]. Namely, whenever there are no tasks in the Ready queue, the currently executing task can stretch until the closest arrival time of a task instance. We will refer to this in our experiments as the *Istretch* method.

### 3. EXPERIMENTAL RESULTS

The first experiment examines the energy gains of using a stochastic voltage schedule at task level. For this we considered a single task with execution time varying between a best case (BCE) and a worst case (WCE) according to a normal distribution. All distributions have the mean (BCE+WCE)/2 and standard deviation (WCE-BCE)/6. For a several cases ranging from highly flexible execution time (BCE/WCE is 0.1) to almost fixed (BCE/WCE is 0.9) we built stochastic schedules for a range of allowed execution times (from

WCE to 3x WCE). We assumed that our processor has 9 different voltage levels, equally distributed between  $f$  and  $f/3$ . For a large number of task instances generated according to the given distribution we computed both the energy of the stochastic schedule (mode 3 in Figure 1) and the WCE-stretch schedule (mode 2 in Figure 1). We depict in Figure 3 the average energy consumption of the stochastic schedule as a part of the WCE-stretch schedule. Note that when the allowed time approaches either WCE or 3-times WCE, the energy consumptions become equal. The lowest possible clock frequency is  $f/3$  which anyway means 3-times WCE, so there is no better schedule for these cases. On the other hand when the allowed time closes WCE, there is no other way but to use the fastest clock. Somewhere between the slowest and the fastest frequencies (Allowed/WCE = 2) is the largest energy gain since the stochastic schedule can use the whole spectrum of available frequencies. Note that the energy gains become more important when the task execution time varies much (BCE/WCE closes 0.1). It is important to notice that WCE-stretch already gains very much energy compared to the non-scaling case. For example when the allowed time is twice the WCE, the WCE-stretch energy is around 25% of the no-scaling energy. But a stochastic approach contributes even more to these gains, as the figure shows.

Next we took two real-life hard-RT applications [9, 15] and applied several energy reduction strategies. The results are depicted in Figure 4. We assumed tasks with normal distributions, with the same characteristics as in the previous experiment. The 100% energy is the energy obtained by running all tasks as fast as possible and executing NOPs when no tasks are supposed to run. We assumed that the NOP instruction consumes only 20% of the average power, as in [20]. The virtual processor used for these experiments has 14 voltage levels, with clock frequencies varying between  $f=100\text{MHz}$  and  $11\text{MHz}$ . A power-down mode is also available, in which the processor consumes 5% of the highest frequency average energy.

The curves named “Upper Bound” depict the upper bound of the energy reduction possibilities. These were obtained in a post-execution analysis, by considering that the tasks are uniformly stretched up to maximum processor utilization as computed in sub-section 2.2.2. This limit is hardly achievable in practice, since the actual execution patterns for all task instances are never available beforehand. Moreover, this optimum obtained by uniformly stretching all instances may violate some deadlines, being therefore useless in practice. A more realistic bound is given by the “Ideal stretch.”

The curves named “Offline+Istretch” were obtained by using only the off-line stretching method and the *Istretch* method mentioned in sub-section 2.2.3. The “All” labeled curves were obtained by

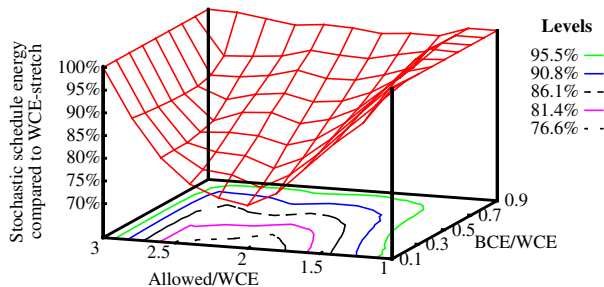


Figure 3. The average energy consumption of a stochastic voltage schedule vs. the energy consumption of a WCE-stretch schedule.

using the off-line strategy, the on-line strategy with “mean proportional” slack distribution (sub-section 2.3), plus the stochastic execution task model (mode 3 in Figure 1). The curves labeled “Ideal stretch” were obtained by using the same method as the “All” curves, except using an ideal-stretch task execution model (mode 1 in Figure 1). Note that this method implies knowing the actual execution time at a task arrival, which is unlikely in reality. For the last three methods, “Offline+Istretch,” “All,” and “Ideal-stretch,” whenever the processor is idle, it goes to a power down mode.

We also tested our scheduling policy on randomly generated task sets of 50 and 100 tasks. The task sets were generated as follows. For each set, the task periods (and deadlines) were selected using a uniform distribution in  $100..5000$  and  $100..10000$  respectively. The worst case execution times were then randomly generated such that the task set would yield approximately 0.67 processor utilization, for the fastest clock. The average utilization after off-line stretching turned out to be 0.92 for the sets of 50 tasks, and 0.85 for the sets of 100 tasks. Using the same processor type as in the previous experiment, we simulated the runtime behavior of several scheduling methods. We also used post-simulation data to obtain the upper bounds, as in the previous experiment. The values depicted in Figure 5 are averages over one hundred sets of tasks. As results from these experiments, our policy (“All”) performs best, when little information on task execution is available.

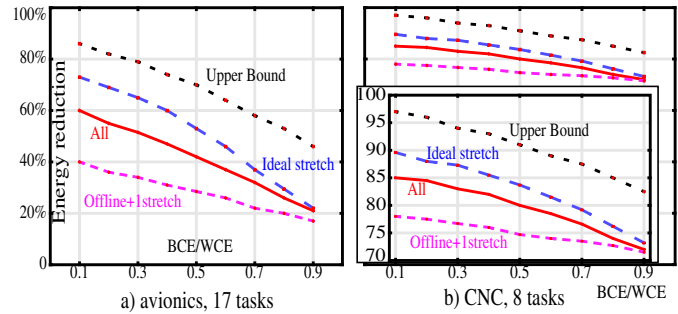


Figure 4. The energy reduction for an a) avionics application [15] and b) a controller CNC [9]. In b) the area between 70-100% is enlarged.

## 4. CONCLUSIONS

We presented and analyzed a scheduling policy for hard real-time tasks running on a dynamic voltage supply processor, with the final purpose of reducing the energy consumption. The policy is designed for sets of tasks with fixed priorities assigned in a rate/deadline monotonic manner. It consists of both off-line and on-line scheduling decisions, taken both at task and task set levels. The off-line decisions use exact timing analysis to derive off-line voltage scaling factors for each task. The on-line policy distributes available processor time on priority basis, using slack levels and statistics. Task-level voltage schedules are built using stochastic data, with the goal of minimizing the average case energy consumption. The paper also contains a proof that our scheduling policy meets all deadlines. Our method can be fully implemented in the RTOS, without appealing to special compilers or changing the software. Yet, combined with the afore mentioned methods, our approach may yield even greater energy reductions. The experimental results show that our policy can be successfully used to reduce the energy consumption in a hard real-time system.

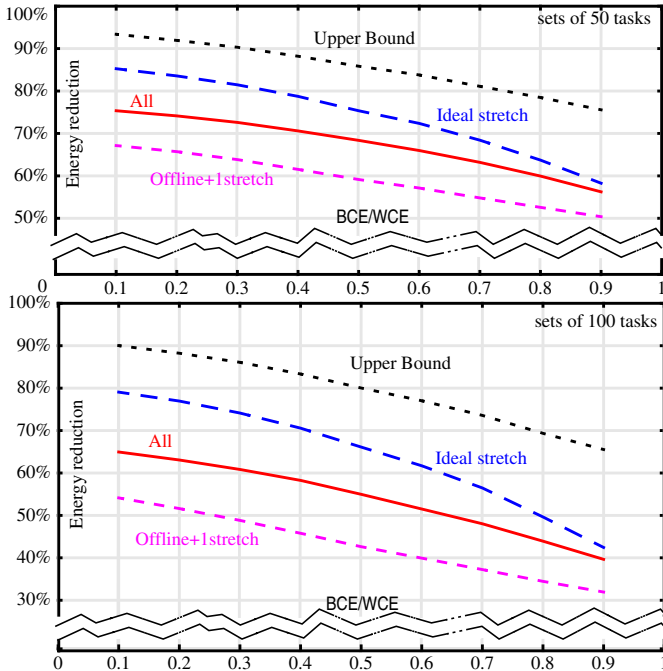


Figure 5. The energy reduction using different strategies for sets of 50 tasks above and sets of 100 tasks below. The values are averages over a hundred task sets.

## 5. ACKNOWLEDGMENTS

This work was funded by ARTES - A network for Real-Time research and graduate Education in Sweden<sup>1</sup>. The author would like to thank Petru Eles, Kris Kuchcinski, and Per Larsson-Edefors for their helpful comments.

## 6. REFERENCES

- [1] Benini, L. and DeMicheli, G. System-level power optimization: techniques and tools, in *ACM Trans. on Design Automation of Electronic Systems*, No. 2, Vol. 5, April 2000, 115-192.
- [2] Burd, T., Pering, T., Stratakos, A., and Brodersen, W. A dynamic voltage scaled microprocessor system in *IEEE Journal of Solid-State Circuits*, No. 11, Vol. 35, November 2000, 1571-1580.
- [3] Chnadrakasan, A., Gutnik, V., and Xanthopoulos, T. Data driven signal processing: an approach for energy efficient computing in *Proceedings of ISLPED'96*, 347-352.
- [4] Dave, B.P., Lakshminarayana, G., and Jha, N.K. COSYN: hardware-software co-synthesis of embedded systems in *Proceedings of the 34th DAC 1997*, 703-708.
- [5] Gruian, F., and Kuchcinski, K. Low-energy directed architecture selection and task scheduling for system-level design in *Proceedings of the 25th Euromicro Conference, 1999*, pp. 296-302.
- [6] Gruian, F., and Kuchcinski, K. LEneS: task scheduling for low-energy systems using variable voltage processors in *Proceedings of ASP-DAC2001*, 449-455.
- [7] Hong, I., Potkonjak, M., and Srivastava, M.B. On-line scheduling of hard real-time tasks on variable voltage

processor in *Digest of Technical Papers of ICCAD'98*, 653-656.

- [8] Ishihara, T., and Yasuura, H. Voltage scheduling problem for dynamically variable voltage processors in *Proceedings of ISLPED'98*, 197-202.
- [9] Kim, N., Ryu, M., Hong, S., Saksena, M., Choi, C.-H., and Shin, H. Visual assessment of a real-time system design: a case study on a CNC controller, *The 17th IEEE Real-Time Systems Symposium, 1996*, 300-310.
- [10] Lee, Y.-H., and Krishna, C.M. Voltage-clock scaling for low energy consumption in real-time embedded systems in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, 1999*, 272-279.
- [11] Lee, S., and Sakurai, T. Run-time voltage hopping for low-power real-time systems in *Proceedings of the 37th DAC, 2000*, 806-809.
- [12] Lehoczy, J., and Ramos-Thuel, S. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems in *Proceedings of RTSS'92*, 110-123.
- [13] Lehoczy, J., Sha, L., and Ding, Y. The rate monotonic scheduling algorithm: exact characterization and average case behavior in *Proceedings of RTSS'89*, 166-171.
- [14] Liu, C.L., and Layland, J.W. Scheduling algorithms for multiprogramming in a hard real time environment in *JACM* 20 (1), 1973, 46-61.
- [15] Locke, C.D., Vogel, D.R., and Mesler, T.J. Building a predictable avionics platform in Ada: a case study in *Proceedings of RTSS'91*, 181-189.
- [16] Mossé, D., Aydin, H., Childers, B., and Melhem, R., Compiler-assisted dynamic power-aware scheduling for real-time applications. *Workshop on Compilers and Operating Systems for Low-Power, October 2000*.
- [17] Pedram, M. Power optimization and management in embedded systems, *Proceedings of ASP-DAC 2001*, 239-244.
- [18] Pering, T., Burd, T., and Brodersen, R., The simulation and evaluation of dynamic voltage scaling algorithms in *Proceedings of ISLPED'98*, 76-81.
- [19] Pering, T., Burd, T., and Brodersen, R., Voltage scheduling in the lpARM microprocessor system in *Proceedings of ISLPED'00*, 96-101.
- [20] Shin, Y., and Choi, K. Power conscious fixed priority scheduling for hard real-time systems in *Proceedings of the 36th DAC, 1999*, 134-139.
- [21] Shin, D., Kim, J., and Lee, S. Intra-task voltage scheduling for low-energy hard real-time applications, *Special Issue of IEEE Design and Test of Computers, October 2000*.
- [22] Suzuki, K., Mita, S., Fujita, T., Yamane, F., Sano, F., Chiba, A., Watanabe, Y., Matsuda, K., Maeda, T., and Kuroda, T. A 300MIPS/W RISC core processor with variable supply-voltage scheme in variable threshold-voltage CMOS, *Proceedings of the ICC'97*, 587-590.
- [23] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced CPU energy in *Proceedings of the First Symposium on Operating Systems Design and Implementation, November 1994*.
- [24] Yao, F., Demers, A., and Shenker, S. A scheduling model for reduced CPU energy in *Proceedings of the 36th Symposium on Foundations of Computer Science, 1995*, 374-382.
- [25] <http://www.transmeta.com>

<sup>1</sup><http://www.artes.uu.se/>