On energy reduction in hard real-time systems containing tasks with stochastic execution times

Gruian, Flavius

Published in:
[Host publication title missing]

2001

Link to publication

Citation for published version (APA):
Gruian, F. (2001). On energy reduction in hard real-time systems containing tasks with stochastic execution times. In *[Host publication title missing]* (pp. 11-16). IEEE - Institute of Electrical and Electronics Engineers Inc..

Total number of authors:
1

On Energy Reduction in Hard Real-Time Systems Containing Tasks with Stochastic Execution Times

Flavius Gruian
Department of Computer Science, Lund University,
P.O. Box 118,
S-221 00 Lund, Sweden
<Flavius.Gruian@cs.lth.se>

Abstract — This paper addresses low energy scheduling in hard real-time systems, containing independent tasks running on dynamic voltage supply (DVS) processors. Since tasks with probabilistic run times are more realistic, we focus on systems containing such tasks. We show that the energy consumption can be reduced if the scheduling takes into account the probabilistic behavior of the tasks. Moreover, stochastic data can be used in deriving schedules both for task sets and individual tasks. First we address task-set level scheduling for two cases. Then we analyze two task-level voltage scheduling approaches: intra-task scheduling and our method called stochastic scheduling. The examples and experimental results presented throughout the paper support our conclusion.

1. Introduction

Today an important design requirement for digital systems is low energy consumption. It has impact on operating time, on system cost, and, of no lesser importance, on the environment. Methods for reducing the power and energy dissipation in digital systems have long been sought for by several research groups. With the arrival of dynamic voltage supply processors [3,17], highly flexible systems can be designed, while taking advantage of supply voltage scaling to reduce 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 another niche for reducing the energy consumption, 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 [1,2]. 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 [4,5,6]. Even for hard real-time systems, where all deadlines have to be met, energy reductions can be achieved as shown in [7,8,9,10,11]. Yet most of the work carried out in the hard real-time systems area does not make use of the stochastic properties of a real-life application. In this paper, we also focus on hard real-time scheduling techniques, but we emphasize the impact of considering the probabilistic behavior of the hard real-time tasks. For this reason we consider tasks with stochastic execution delays.

Task level voltage scheduling decisions can reduce even further the energy consumption. Some of these, the intra-task scheduling methods, use several re-scheduling points inside a task, and are usually compiler assisted [13,14,15]. Alternatively, fixing the schedule before the task starts executing as in [8,11,16] eliminates the internal scheduling overhead, but with possible affects on energy reduction. Statistics can be used to take advantage of the dynamic behavior of the system, both at task level [15] and at task-set level [7]. In this paper, we show the importance of employing stochastic data in deriving efficient voltage schedules at both task and task-set levels.

The rest of the paper is organized as follows. Section 2 presents the scheduling alternatives available at task-set level, for two cases, increasing in generality. Task-level scheduling methods for low average energy are addressed in section 3. We give examples and some experimental results throughout the paper, in specific sub-sections, hoping to make our point even clearer. Finally, our conclusions make the subject of section 4.

2. Task-Set Level Scheduling Decisions

In this paper, we address only independent tasks running on a single dynamic supply voltage processor. Yet the ideas can be applied to more general cases. Each of the N tasks in the task set is defined by a 4-tuple $\tau_i = (X_i, C_i, T_i, D_i)$ composed of the execution pattern, worst case execution time, period, and deadline for task τ_i. The execution pattern X_i is a random variable describing the number of clock cycles required to execute the task. We will denote by $\overline{X_i}$ the expected value of the random variable X_i. At the fastest processor speed (highest clock frequency), the maximal value for X_i is C_i.

The focus of this discussion is on the influence of considering tasks with probabilistic execution pattern on the task-set level scheduling strategies for low energy consumption. We show that scheduling policies considering only the tasks' worst case execution time (WCET) can be improved when stochastic data is used. We compute therefore the following measures. The worst case execution speed, s^{WCE}, which does not consider the stochastic behavior of the tasks. The expected ideal speed, s^{ideal}, which assumes that all tasks will always run at their expected time and can take full advantage of this knowledge. And, finally, the average real speed, s, which is a measure of the realistic behavior, when the actual execution times are not known a priori. We can use these three measures to compute the energy consumptions for the worst, ideal, and average cases. The energy consumption for a given time interval [$t0$, $t1$] is defined as the integral of power over time:

$$E = \int_{t0}^{t1} P(s(t))dt \cong \int_{t0}^{t1} (s(t))^{\alpha} dt \qquad (1)$$

Note that $\alpha \geq 2$, value influenced by the delay-supply voltage dependency. All the examples presented here use $\alpha = 3$, but all the formulae are given for the general case. To get the exact value of the energy consumption, one has to find out the processor speed at every time moment.

Because of the dynamic behavior of the system, we need to address not only off-line scheduling methods, but also on-line re-scheduling policies. Depending on the generality of the problem, the complexity of the off and on-line policies varies.

We address two cases, with increasing generality. First we consider task sets with tasks having the same period and deadline. Next we analyze tasks having the same period, but different deadlines. The influence of stochastic data on scheduling sets of task which have different periods and deadlines makes the subject of another of our papers [12], and will not be discussed here.

2.1 Unique Period, Unique Deadline

This is the simplest possible case, where the set of tasks has to finish before a certain deadline, in no particular order. Formally we consider $\forall(\tau_i, \tau_j), T_i = D_i = T_j = D_j = A$. When the actual execution time of each task is known beforehand, the optimal schedule from the energy point of view is given by uniformly stretching the tasks to exactly meet the deadline [11]. The worst case processor speed is then computed as:

$$s^{WCE} = \left(\sum_{1 \leq i \leq N} C_i\right) / A \tag{2}$$

while the ideal expected speed:

$$s^{ideal} = \left(\sum_{1 \leq i \leq N} \overline{X}_i\right) / A . \tag{3}$$

Using only off-line decisions, one can always guarantee the deadlines only by using the worst case speed. Yet, an on-line re-scheduling algorithm is required for energy efficiency. The on-line re-scheduling algorithm is used every time a task completes its execution. In each of these scheduling points, a new expected optimal processor speed is computed. If task τ_j is about to start executing, the new optimal processor speed is:

$$s_j = \left(\sum_{j \leq i \leq N} C_i\right) \Big/ \left(A - \sum_{1 \leq k < j} (X_k / s_k)\right) \tag{4}$$

Here X_k represents the actual execution time for task *k* in the current iteration. Note that the execution order of the tasks in this case is important from the energy consumption point of view. A simplistic approach would be to adopt a random execution order for each new task-set instance. This will yield in the long run an energy consumption which is somewhere between the ones produced by the worst and the best order.

Yet, better orders can be found at a deeper analysis of task set. In principle it is good eliminate the biggest uncertainties early, so that we can settle for a close to optimal speed as soon as possible. Tasks exhibiting a very dynamic behavior (large discrepancy between their WCET and expected execution time) seem like good candidates for executing first. For example one could order tasks according to the following priorities (low numbers mean high priority):

$$p_i = 1 / (C_i - \overline{X}_i) \tag{5}$$

On the other hand, task lengths play an important role. Short tasks should be executed first, since the left-over time might be used early on. Finally, according to our computations, the best ordering is given by using the following priorities:

$$p_j = \overline{X}_j \Big/ \left(1 - \left(\frac{\sum_{j < i \leq N} C_i}{\sum_{j \leq i \leq N} C_i - \overline{X}_j}\right)^{\alpha - 1}\right) \tag{5'}$$

Note that when using relations of form (5) to compute an order, an algorithm of complexity $O(N \cdot \log(N))$ will do. For the optimal order (5'), we need an algorithm with computational complexity of $O(N^2 \cdot \log(N))$, since the priorities depend on the already ordered tasks.

Having decided on the schedule, we can now compute the energy consumption for each of the cases presented here. Note that the processor speed is constant on intervals, and for the WCE and ideal case there is a unique speed:

$$\begin{aligned} E^{WCE} &= (s^{WCE})^{\alpha} \sum_{1 \leq i \leq N} (\overline{X}_i / s^{WCE}) \\ &= \left(\left(\sum_{1 \leq i \leq N} C_i\right) / A\right)^{\alpha - 1} \cdot \left(\sum_{1 \leq i \leq N} \overline{X}_i\right) \end{aligned} \tag{6}$$

$$E^{ideal} = A (s^{ideal})^{\alpha} = \left(\sum_{1 \leq i \leq N} \overline{X}_i\right)^{\alpha} / A^{\alpha - 1} \tag{7}$$

$$E^{real} = \sum_{1 \leq i \leq N} \frac{X_i}{s_i} \cdot s_i^{\alpha} = \sum_{1 \leq i \leq N} X_i \cdot s_i^{\alpha - 1} \tag{8}$$

Note that the last energy value is dependent on the task ordering. This is where one can reduce the energy consumption by using an optimal ordering. Also note that in the long run we can talk about a mean value for E^{real}. In fact, this mean value can be estimated by replacing X_i with $\overline{X}_i$ in (4) and (8). For the next example we used this kind of estimates, while for the following experiments we computed the actual energy consumption for each iteration and report the average value.

Consider a simple task set, composed of three tasks $\{\tau_1 = (X_1, 20, 100, 100)$, $\overline{X}_1 = 16$; $\tau_2 = (X_2, 30, 100, 100)$, $\overline{X}_2 = 20$; $\tau_3 = (X_3, 40, 100, 100)$, $\overline{X}_3 = 32\}$. The processor speeds for four cases, plus the energy consumptions are gathered in Table 1. The first two columns refer to the worst and the ideal cases. The "worst case" decides an off-line speed such that the tasks will always meet their deadlines, and uses only that speed at runtime. The "ideal case" assumes that the exact execution times are known beforehand and decides an optimal processor speed for each period. The real case implies using a runtime voltage re-scheduling policy. The processor speed is recomputed each time a task finishes execution, that is why we give three speed values for the real case. The "best order" in Table 1 is the order given by the priorities computed as in (5'). The "reverse best" is when the task assume an inverse order. From the table we can see that the on-line strategy reduces the energy consumption, as expected. More importantly, a good ordering also contributes to further energy reductions.

We also performed experiments for larger task sets, containing 10, 30, 60 and 100 tasks with stochastic behavior, using normal distributions with mean C/2 and standard deviation C/6. For each of these set sizes, we examined one thousand ran-

Table 1: An example of case 2.1 ($\alpha = 3$)

$\{\tau_1, \tau_2, \tau_3\}$	WCE	ideal	real expected (estimates)	
			best order: 2, 3, 1	reverse best: 1, 3, 2
speed, s	0.90	0.68	0.90, 0.77, 0.55	0.90, 0.85, 0.67
normalized energy, E	1.752	1	1.273	1.433

dom sets and extract the energy consumption via simulation for three different situations. We ordered the tasks according to (5') and also using the inverse order. We compared the energy consumption for these two to the ideal case, defined as for Table 1. Again, note that the ideal case is impossible to be reached in practice. The results are depicted in Fig. 1.

2.2 Unique Period, Different Deadlines

The problem becomes more complex when the deadlines for the tasks differ: $\forall(\tau_i, \tau_j)$, $T_i = T_j = A$, $D_i \neq D_j \leq A$. A deadline monotonic scheduling strategy would, in this case, guarantee feasible schedules up to full processor utilization. Considering the tasks ordered according to their deadlines, the processor speed for each task can be computed as in [7]:

$$s_j^{WCE} = \sum_{j \leq i \leq N} \left(C_i / \left(D_i - \sum_{1 \leq k < j} (C_k / s_k^{WCE}) \right) \right) \quad (9)$$

The ideal speed for each task j is obtained when one knows a priori the execution pattern of all tasks:

$$s_j^{ideal} = \sum_{j \leq i \leq N} \left(\overline{X}_i / \left(D_i - \sum_{1 \leq k < j} (\overline{X}_k / s_k^{ideal}) \right) \right) \quad (10)$$

For the real case, an on-line scheduling method is needed. The processor speed must be recomputed whenever a task finishes execution. The time moment when task j finishes is:

$$t_j = \sum_{1 \leq k \leq j} (X_k / s_k) \quad (11)$$

The processor speed for the next task is re-computed (see (9)):

$$s_{j+1} = \sum_{j < i \leq N} (C_i / (D_i - t_j)) \quad (12)$$

As in the case described in the previous section, the execution order influences the energy consumption. On the other hand, we have less freedom to reorder tasks since the deadlines have to be met. With this in mind, we present here an off-line preemptive pre-scheduling strategy, which attempts to obtain a

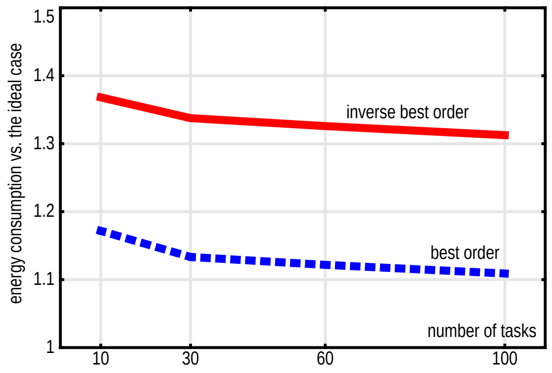

Fig. 1. The energy consumption obtained from simulation for case A: the dotted line was obtained for order given by (5'), the continuous line was obtained for the reverse order. All values are relative to the ideal case energy

```
begin procedure OfflinePreemptiveSchedule
empty WorkList
  forall deadlines $D_i$, longest..shortest do
    time := $D_i$, empty NextWorkList
    * put all tasks with deadline $D_i$ in WorkList
    * order WorkList descending
      using priorities $p_j$ from (5')
    forall tasks $\tau_k$ in WorkList do
      if $C_k/s_k$ > (time - $D_{i-1}$) then
        * split $\tau_k$:
          NextWorkList.add(task with
            C := $C_k$ - (time - $D_{i-1}$)*$s_k$,
            D := $D_{i-1}$, X := min(C, $X_k$))
          Also update $C_k$ := time - $D_{i-1}$
      end if
      * schedule $\tau_k$ between [time - $C_k/s_k$, time)
      time := time - $C_k/s_k$
    end forall
    * make NextWorkList the new WorkList
  end forall
end procedure
```

Fig. 2. Off-line scheduling algorithm for reduced energy in the case of task sets with unique period and different deadlines

low average energy consumption. The algorithm, presented in Fig. 2, uses an initial step in which WCE processor speeds are computed. It examines then the time intervals between two consecutive deadlines, starting from the longest deadline. For these "scheduling intervals" we can use the scheduling method presented in sub-section 2.1, but this time we schedule tasks from the end backwards. Whenever a task does not fit entirely in the remaining time slot, it is split in two parts with specific stochastic behavior. The end part is scheduled in the current scheduling interval. The other part is treated as a new task, which has to be completed before the current scheduling interval. The algorithm finishes when we scheduled all tasks and we reached moment zero. The on-line re-scheduling algorithm is the same as the one described previously, by equation (12).

Having decided on the schedule, we can now compute the expected energy consumption for all the cases presented here. As in the previous case, the processor speed is constant on intervals. For brevity we will not expand the expressions: $E^{WCE} = \sum_{1 \leq i \leq N} \overline{X}_i \cdot s_i^{WCE}$, $E^{ideal} = \sum_{1 \leq i \leq N} \overline{X}_i \cdot s_i^{ideal}$,

$E^{real} = \sum_{1 \leq i \leq M} X_i \cdot s_i$.

Note that in the last case the number of intervals with constant speed can increase to $M \geq N$ as a result of preemption. As an example, consider the following task set: $\{\tau_1 = (X_1, 10, 16, 15)$, $\overline{X}_1 = 9$; $\tau_2 = (X_2, 6, 16, 16)$, $\overline{X}_2 = 3\}$. A classic deadline monotonic strategy would schedule τ_1 first followed by τ_2. Using our algorithm described in Fig. 2, the energy is reduced by splitting τ_2 in two parts and the schedule looks differently. The first part of task τ_2, of length 5 (C_2 is 6) is executed first, then τ_1 gets executed, and finally the remaining part of τ_2. The differences in energy consumption for the average case are given in Table 2. The "WCE" and "ideal" columns have the same meaning as in Table 1. The "real expected" contains the expected or average value taken over a large number of task set instances. The "classic order" corresponds to the situation when one uses the classic deadline monotonic ordering. The "preemptive order" is the schedule found by our algorithm. Note that in the long run the second part of task 2 will not execute, hence the expected value of 0.

Table 2: An example of case B ($\alpha = 3$)

$\{\tau_1, \tau_2\}$	WCE	ideal	real expected	
			classic order 1, 2	preemptive order part of 2, 1, rest of 2
speeds, s	1, 1	0.79, 0.66	1, 0.86	1, 0.833, (0)
normalized energy, E	2.328	1	1.630	1.346

3. Task Level Scheduling Decisions

Task-level voltage scheduling has captured the attention of the research community rather recently [16]. One of the main reasons behind using task-level scheduling techniques is that these can lower the energy consumption of a certain task without requiring strong knowledge about the other tasks in the system. From the system point of view these methods are "self-centered," in the sense that they are concerned with doing their best for the current task and the given processor time. They try to fill-in their allocated time as best as possible. At the system level this means that uncertainties are dealt with at individual task level whenever this is possible. In this case, one can use classic scheduling techniques at system level and still get energy efficient systems. Even more efficient is having both the task-set level and the task-level scheduling techniques aware of the temporal non-determinism existent in the system, as presented for example in [12]. In the following we will describe two approaches to task-level scheduling and then compare these in different aspects.

3.1 The Intra-task, Compiler Assisted Schedule

The current trend in task level scheduling is that of using rescheduling points inside a task [13,14,15]. In [13, 15] the task is split into several sequential time slots, each with its own WCET. Every time a new slot begins, the processor speed is recomputed and adjusted to exactly meet the deadline. Additionally, [15] uses statistical data to improve the task level schedule, by slowing down different regions of a task according to their average execution time. In [14] the time slots are actually basic blocks extracted at assembly language level. Note that all of these approaches require compile time support. The compiler or a post-processor has to estimate the WCET for each region or basic block. It should also insert the necessary code for re-computing the new processor speed and the code for shifting the supply voltage in each re-scheduling point (Fig. 3.a). One of the advantages of this method seems to be that the operating system (OS) does not need to be aware of the DVS characteristics of the hardware architecture, since the OS can be by-passed using compiler generated re-scaling code. Thus, one can use an "old" OS without modifications.

Analyzing the energy reduction capabilities of such an approach one must take into consideration the "predictability"

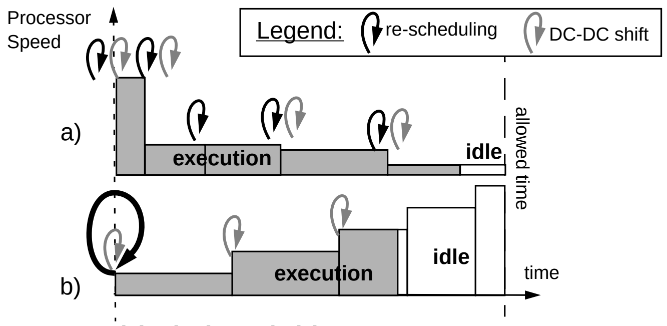

Fig. 3. Task-level voltage schedule strategies:
a) intra-task re-scheduling points b) stochastic schedule

of a task. This means how much information about the actual finishing time one can gather during the task execution. If, for example, the actual finishing time is known right at the beginning of the task, the scheduler can make the best decisions. If the actual finishing time can not be known until the very end, nothing can be gained. In this light, the conclusions from section 2 apply also here: the best decisions can be made if the uncertainties are eliminated early in the execution.

Without great loss of generality we will consider that there is a point in a task before which nothing is known about the finishing time. At this point, the task becomes aware of the exact value of the finishing time. It is interesting to see how the energy consumption of an intra-task schedule depends on the location of this point. If we denote the ratio of the unaware part of the task by *y*, it is not hard to show (using section 2) that the normalized energy expression for the intra-task schedule is:

$$E(y) = y + (1-y)\left(\frac{1-y}{C/\bar{X} - y}\right)^{\alpha - 1} \qquad (14)$$

where C, $\bar{X}$, and α have the same meaning as introduced in section 2. Note that the unit energy is that obtained by running the whole task ($\bar{X}$ cycles in average) at the clock frequency computed at the beginning of the task. This frequency has to accommodate the whole task even in the worst case. Thus, the unit energy is the same as the WCE-stretch energy from subsection 3.2. This formula assumes that the re-scheduling slots are very small and the re-scheduling overhead negligible. An example of such dependency is depicted in Fig. 4, where we instantiated $\bar{X} = C/2$ and $\alpha = 3$. We will return to this in subsection 3.3.

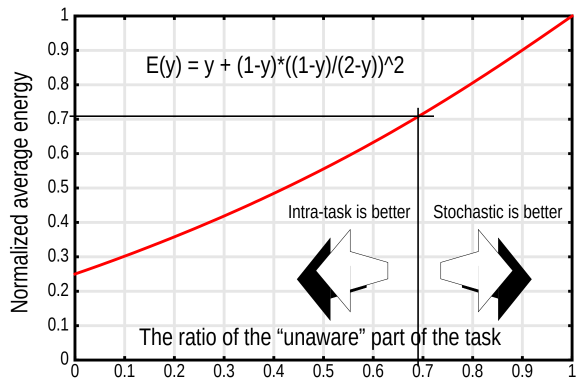

Fig. 4. The energy consumption of a intra-task schedule depending on the task predictability.

3.2 The Stochastic Schedule

The second task-level voltage scheduling approach is what we call stochastic scheduling. It can be completely implemented inside the RT-OS and it does not require special compiler support. Consequently, it interferes less with the actual task than the compiler based methods. This also means that tasks do not need to be re-compiled when the architecture changes. In principle our approach computes a voltage schedule only once, when the task starts executing. During task execution no re-scheduling is done, but the supply voltage is changed at well established intervals (Fig. 3.b).

First, note that in many cases tasks with variable execution time 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 regions. Our approach uses stochastic data to build a multiple voltage schedule, in which the processor speed increases towards the deadline. The purpose for using stochastic data is to minimize the average case energy consumption.

The stochastic voltage schedule 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. Unlike the compiler oriented methods, this approach can start with very little information about the task (only WCET) and gradually acquire more data at runtime. The actual voltage schedule is obtained by minimizing the expected value of the energy consumption. The on-line algorithm for deciding the schedule has a low computational complexity, linear with the number of available voltages. A detailed description of the algorithm is presented in [12].

Two examples of stochastic voltage schedules are given in Fig. 5. We assumed a normal probability distribution with the mean $\overline{X}$ of 70 cycles, and standard deviation of 10. *C* 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 at the highest clock frequency (*f*).

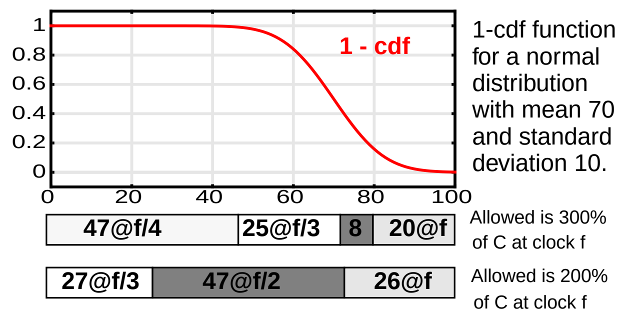

Fig. 5. Examples of stochastic voltage schedules for a task with normal distribution execution time and worst case behavior of 100 cycles

Next we present an experiment that examines the energy gains of using a stochastic voltage schedule at task level. For this we considered a single task with variable execution pattern, identical with the one used for the example in sub-section 3.1, Fig. 4. Assuming that the worst case behavior uses *C* cycles, we used a normal distribution centered on $\overline{X} = C/2$, with a standard deviation of *C/6*. We considered that our processor has 9 different voltage clock speeds, equally distributed between *f* and *f*/3. With this settings, we built stochastic schedules for a range of allowed execution times (from C at *f* to 3xC at *f*). For a large number of task instances generated according to the given distribution we computed both the energy of the stochastic schedule and a WCE-stretch schedule. The WCE-stretch schedule is the one obtained by assuming a unique frequency, for which the task will always meet the deadline, even the for WCE pattern (*C* cycles). We compared then the energy consumption for these two schedules to the non-scaling case, where the processor executes at its maximal frequency, *f*, all the cycles needed and then shuts down (0 power). The results are depicted in Fig. 6. Note that when the allowed time approaches either *C* at *f* or 3-times *C* at *f*, the energy consumptions for the two schedules become equal. The lowest available clock frequency is *f*/3, which means 3-times slower than *f*, so there is no better schedule for these cases. On the other hand when the allowed time closes *C* at *f*, there is no other way but to use the fastest clock. Somewhere between these two extremes (2x) is the largest energy gain since the stochastic schedule can use the whole spectrum of available frequencies.

3.3 Comparison: Intra-task vs. Stochastic Scheduling

We summarize the most obvious features of the two scheduling approaches in Table 3. Depending on the application, features 1, 2 and 3 can be seen as advantages or drawbacks. If one can access the hardware resources directly without affecting the whole system, intra-task scheduling is better. Yet, an OS level policy hides the hardware peculiarities of the system and allow for better application portability/mobility. Intra-task scheduling seems to require more accurate and detailed timing information than the stochastic approach (feature 3). In some cases these estimates are much harder to obtain than a pessimistic global WCET estimate. Stochastic schedules can take

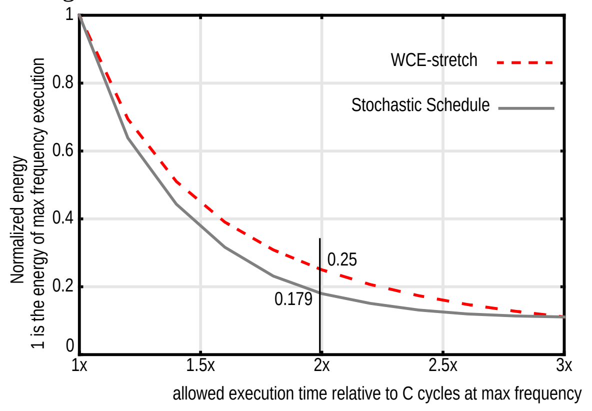

Fig. 6. The average energy consumption of a stochastic voltage schedule compared to the energy consumption of a WCE- stretch schedule.

advantage of the run-time history (feature 5). Intra-task scheduling seems to interfere more with the actual task because of the additional re-scheduling code inserted at compile time (6). Overall, the work overhead for the intra-task scheduling appears bigger than that for the simpler stochastic scheduling strategy.

This comparison would not be complete without an analysis of the actual energy reduction of the two methods. The stochastic schedule does well only when the deadline is rather loose (feature 9). For very tight deadlines, (those close to 1x in Fig. 6), a stochastic schedule is a bad choice, since intra-task scheduling can detect early finishing tasks (close to 0 in Fig. 4) and take advantage of that. Yet, if the task is very "un-predictable" (close to 1 in Fig. 4) the intra-task scheduling method can yield as bad results as the WCE-stretch in Fig. 6. In this cases, for certain deadlines a stochastic schedule is a better choice (feature 8). For example, at a deadline of 2xC in Fig. 6, a stochastic schedule energy is 70% of the WCE-stretch energy. This corresponds to a 0.69 "unaware"-ness factor in Fig. 4. Thus if it takes longer than 69% of a task execution to determine its exact finishing time, a stochastic schedule does better. Otherwise an intra-task schedule is recommended. In conclusion, the actual stochastic behavior of the task is very important and different scheduling strategies can yield better or worse results, depending on the situation.

Table 3:Intra-task vs. Stochastic Scheduling

#	Feature	Intra-task	Stochastic
1	implementable as a task-transparent policy?	no	yes
2	implementable as an OS-transparent policy?	yes/partially	with performance loss
3	needs compiler support?	yes	no/little
4	amount of off-line required estimates	many region-wise WCETs	only the task WCET
5	run-time adjustable? can use history?	possibly	yes
6	run-time interference with the task	moderate to high	low to moderate
7	overall method complexity	large	small
8	performance is sensitive to internal task "predictability"?	yes	no
9	performance is sensitive to deadline variations?	slightly	very

4. Conclusions

In this paper, we addressed scheduling methods targeting energy reduction in hard real-time systems containing dynamic supply voltage processors. Our main purpose was to raise the level of awareness for considering stochastic data during scheduling in such systems. Both task-set and individual task level scheduling decisions were described.

For task sets we started from the simple case of tasks having the same unique deadline and period. Then we generalized the problem by considering different deadlines. For these cases we showed that more efficient schedules, from the energy point of view, can be derived when stochastic data is taken into consideration.

We also addressed task-level voltage scheduling. Two approaches were compared: intra-task scheduling and our own, stochastic scheduling. We compared the two approaches from several point of views. The discussion pointed out the importance of analyzing the designs from the perspective of their stochastic behavior.

5. References

[1] L. Benini and G. DeMicheli, "System-level power optimization: techniques and tools," *ACM Trans. on Design Automation of Electronic Systems*, No. 2, Vol. 5, April 2000, pp. 115-192.

[2] M. Pedram, "Power optimization and management in embedded systems,“ *Proc. of ASP-DAC 2001*, pp. 239-244.

[3] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, "A 300MIPS/W RISC core processor with variable supply-voltage scheme in variable threshold-voltage CMOS," *Proc. of the IEEE Custom Integrated Circuits Conference 1997*, pp. 587-590.

[4] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms,"*Proc. of the '98 ISLPED*, pp 76-81.

[5] A. Chnadrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing," *Proc. of the '96 ISLPED*, pp. 347-352.

[6] M. Weiser, B. Welch, A Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. of the First Symposium on Operating Systems Design and Implementation*, November 1994.

[7] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *Proc. of the 36th Symposium on Foundations of Computer Science*, pp. 374-382, 1995.

[8] I. Hong, M. Potkonjak, and M.B. Srivastava, "On-line scheduling of hard real-time tasks on variable voltage processor," *Digest of Technical Papers of the 1998 ICCAD*, pp. 653-656.

[9] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proc. of the 36th DAC, 1999*, pp. 134-139.

[10] Y.-H. Lee and C.M. Krishna, "Voltage-clock scaling for low energy consumption in real-time embedded systems,“ *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications*, 1999, pp. 272-279.

[11] F. Gruian and K. Kuchcinski, "LEneS: task scheduling for low-energy systems using variable voltage processors," *Proc. of ASP-DAC2001*, pp. 449-455.

[12] F. Gruian, "Hard real-time scheduling using stochastic data and DVS Processors," to be presented at *ISLPED 2001*, August 2001.

[13] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems,“ *Proc. of the 37th DAC*, 2000, pp. 806-809.

[14] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *Special Issue of IEEE Design and Test of Computers*, March 2001, 18(2) pp. 20-30.

[15] D. Mossé, H. Aydin, B. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," *Worksop on Compilers and Operating Systems for Low-Power*, October 2000.

[16] T. Ishihara,H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. of the '98 ISLPED*, pp 197-202.

[17] http://www.transmeta.com