A Probabilistic and Energy-Efficient Scheduling Approach for Online Application in Real-Time Systems

Thorsten Zitterell
Department of Computer Science
Albert-Ludwigs-University, Freiburg, Germany
tzittere@informatik.uni-freiburg.de

Christoph Scholl
Department of Computer Science
Albert-Ludwigs-University, Freiburg, Germany
scholl@informatik.uni-freiburg.de

ABSTRACT

This work considers the problem of minimizing the power consumption for real-time scheduling on processors with discrete operating modes. We provide a model for determining the expected energy demand based on statistical execution profiles which considers both the current and subsequent tasks. If the load after the execution of the current task is expected to be high and slack time is conserved for subsequent tasks, we are able to derive an optimal solution to the energy minimization problem. For the remaining cases we propose a heuristic approach that also achieves a low run time overhead. In contrast to previous work, our scheduling approach is not restricted to single task scenarios, frame-based real-time systems, or pre-computed schedules. Simulations and comparisons with energy-efficient schedulers from literature demonstrate the efficiency of our approach.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—scheduling; D.4.7 [**Operating Systems**]: Organization and Design—real-time systems and embedded systems

General Terms

Algorithms

Keywords

Energy-aware scheduling, hard real-time, dynamic voltage scaling

1. INTRODUCTION

Mobile and battery driven devices like cellular phones have to meet high and competing demands. On the one hand they must be able to run computationally expensive applications which require powerful processors, on the other hand they should offer a long operating time.

Dynamic Voltage and Frequency Scaling (DVFS) is an effective method to control the trade-off between power consumption and processing power. For CMOS designs the energy dissipated per clock cycle is a quadratic function of the operating voltage [2]. However, a voltage reduction requires the reduction of the processor frequency. Energy savings by DVFS can be large if the performance at a lower frequency is still sufficient to execute the applications. DVFS solutions for commercial processors commonly provide only discrete voltage and frequency settings [4, 11].

For hard real-time systems it is mandatory to specify the worst-case execution time (WCET) of tasks or their worst-case execution cycles (WCEC), respectively. This information is required to verify if a schedule is feasible and tasks will not miss their deadlines. However, real-time tasks usually show variations in their execution time and the actual execution time is lower than assumed for the worst case scenario. For instance, the time required for object tracking within a camera picture depends on the motion of the object. Online DVFS algorithms generally are superior to pre-computed, fixed schedules: any static frequency schedule can not utilize slack time resulting from variances in task execution times to reduce the frequency and power consumption.

Some approaches, such as [5, 6], directly use slack time to reduce the execution frequency of a task. Other methods scale the processor speed to a value which corresponds to the (temporarily) reduced processor utilization [10]. However, a limitation of such techniques is that they do not alter the processor speed until a task terminates and only compute schedules based on the worst-case behavior. Other studies showed that it is beneficial to additionally consider statistical information about the execution profile of a task [3, 8, 9, 12, 13, 14, 16]. As the execution probability decreases with the number of executed cycles of a task, they propose to start task execution with a lower frequency which is increased until the job completes (*accelerating frequency schedule (AFS)*). Such approaches aim to minimize the expected energy demand without violating deadlines and defer the usage of higher frequencies associated with higher energy consumption. The authors of [15] showed that it is NP-hard to find an optimal AFS for a single task.

The Lagrange multiplier method in [9] computes optimal accelerating frequency schedules but it is restricted to single tasks. Other approaches, such as [13, 16], provide interesting solutions for so-called frame-based systems where all tasks share the same deadline, have fixed priorities and are non-preemptive. However, they do not provide solutions for more complex real-time systems where preemptive tasks with dynamic priorities and arbitrary deadlines are used. Pre-computed optimal schedules are used in [12] for EDF scheduling with non-uniform deadlines. The authors perform an offline analysis of the (periodic) scheduling problem based on execution profiles. However, their approach is not able to make use of slack resulting from variances in execution time.

In our work, we consider the problem of minimizing the power consumption for scheduling under real-time constraints and for processors which are restricted to discrete operating modes. We present a model which does not only consider the energy demand of

This work has partly been supported by the German Research Foundation (DFG) within the Research Training Group 1103.

DAC'10, June 13-18, 2010, Anaheim, California, USA

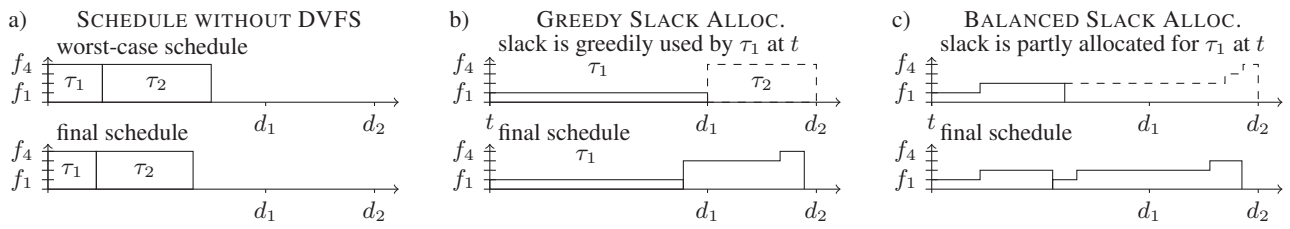

Figure 1: Example for the consecutive execution of two tasks

single tasks executed in isolation but also includes the expected load incurred by subsequent tasks. For the cases where slack time is conserved for subsequent tasks, we are able to derive an optimal solution to the energy minimization problem analytically. For the remaining cases we propose a heuristic approach that also achieves low run time overhead. We also discuss details on how our approach can be implemented efficiently based on histogram representations for execution profiles. Our exact solution can be computed online with a run time which is logarithmic in the number m of discrete frequencies and logarithmic in the number of bins in the histogram based representation for the execution profiles. We further show that a rather small number of bins (in the order of 10 to 20) is enough to obtain nearly optimal results.

The rest of this paper is structured as follows. Preliminaries including the energy model and a motivating example are given in Sect. 2. The energy minimization problem is formulated in Sect. 3. In Sect. 4, we describe our solution to the problem. Simulation results are presented in Sect. 5. Finally, Sect. 6 concludes the paper.

2. PRELIMINARIES

2.1 System and Task Model

The processor provides m different operating frequencies $F = \{f_1, \ldots, f_m\}$ with $f_i < f_j$ for $1 \leq i < j \leq m$. The power at a frequency f is given by a function $p : \mathbb{R}^+ \to \mathbb{R}^+$. We assume that the system always changes to the lowest available frequency when idling and the power function $p(f)$ only accounts for the offset to the power consumption used in idle mode. We assume that $\frac{p(f)}{f}$ (i.e. the energy consumption per cycle) is strictly monotonically increasing with $f \in F$. Moreover, we assume that p is strictly convex. (Note that the power function for CMOS circuits, e.g., fulfills these assumptions.) For our approach we do not need stronger restrictions to the power function (e.g. $p(f) = f^\alpha$ as in [12]).

We assume a set of n periodic tasks. A task instance is specified by its worst-case demand of execution cycles (WCEC) denoted by $\check{C}$ and a period T. The variable $\check{R}$ denotes the number of remaining execution cycles. When a task is to be started, we have $\check{R} = \check{C}$. However, when a task is resumed (after an interrupt or preemption) it is $\check{R} < \check{C}$. Note, the $\check{}$ symbol indicates that a variable refers to a number of clock cycles instead of an amount of time. For example, $\check{Z}$ execution cycles at frequency f will take a time of $Z = \check{Z} \cdot f^{-1}$.

Further, we assume that there exists a function $h : [0, \check{C}] \to [0, 1]$ with $h(y) = p$ for all $y \in [x, x+1)$ if the probability that a task completes its execution exactly at cycle x is p. We define $q(x) = 1 - \int_0^x h(z)\, dz$ which (for natural numbers x) can be interpreted as the probability that a task terminates not earlier than at cycle x or is executed at cycle x, respectively. In Sect. 4.3, we will consider a discretized representation of this function.

In the following, we assume that tasks are scheduled according to the *Earliest Deadline First* (EDF) strategy [7]. The worst-case utilization factor of a task set $\Gamma = \{\tau_i(\check{C}_i, T_i), i = 1, \ldots, n\}$ is given by $\mathcal{U} = f_m^{-1} \sum_{\tau_i \in \Gamma} \check{C}_i / T_i \leq 1$.

2.2 Motivating Example

Fig. 1 shows three different scheduling variants for executing two jobs τ_1 and τ_2. The processor provides four discrete frequencies $f_1 = \frac{1}{4}$, $f_2 = \frac{1}{2}$, $f_3 = \frac{3}{4}$ and $f_4 = 1$ (given on the y-axes). For this example, we use the simple power function $p(f) = f^3$. At maximum frequency, the worst-case execution times of the two jobs are $C_1 = 0.9$ and $C_2 = 1.8$. The deadlines for τ_1 and τ_2 are $d_1 = 3.6$ and $d_2 = 5.4$, respectively. Moreover, we assume that all tasks terminate earlier and the actual time demand at f_4 is $A_1 = 0.8$ and $A_2 = 1.6$.

Fig. 1a) shows both the worst-case and the actual schedule for executing the two jobs without DVFS. For this non-DVFS method (with actual execution times $A_1 = 0.8$ and $A_2 = 1.6$) the consumed energy would be $E_1 = 0.8 \cdot 1^3 + 1.6 \cdot 1^3 = 2.4$.

In Fig. 1b), a DVFS scheduler determines and uses the available time of $S_1 = d_1 - t = 3.6$ time units to greedily reduce the frequency. The lowest possible frequency to execute τ_1 is $\frac{C_1}{S_1} = \frac{1}{4} = f_1$. The instance of τ_1 will also not miss its deadline d_1 using this frequency. However, τ_1 only has an actual time demand of 0.8 (at frequency f_4), and hence will take 3.2 time units until completion. Thus, the remaining unused time (slack) would be 0.4 time units and the second job could reduce the frequency to $\frac{C_2}{d_2 - d_1 + 0.4} \approx 0.81$. As this frequency is not available, one could either use the next higher frequency which would result in an energy consumption of $E_2 = 3.2 \cdot 0.25^3 + 1.6 \cdot 1^3 = 1.65$ or split the execution of τ_2 into intervals with different frequencies: we plan to execute the instance for 1.6 time units with f_3 and 0.6 time units with f_4. As the actual computation time is lower, τ_2 will execute 1.6 time units at f_3 and only 0.4 time units at f_4. This results in an energy consumption of $E_3 = 3.2 \cdot 0.25^3 + 1.6 \cdot 0.75^3 + 0.4 \cdot 1^3 = 1.125$.

For the last considered variant in Fig. 1c) we follow the approach of balancing the distribution of the available slack. When τ_1 is started it does not greedily allocate the available time of 3.6 time units. In contrast to previous methods, it leaves some slack to the subsequent task instance even if it shows its worst-case behavior. This can be achieved for example by planning an execution of 0.8 time units at frequency f_1 and 1.4 time units at frequency f_2. The upper diagram depicts the predicted worst-case schedule at time t. As job τ_1 terminates earlier again it will only spend 1.2 time units in the second interval. In this example, the slack of 1.6 time units is passed to τ_2 which executes in intervals of 0.4, 2.2 and 0.8 time units with discrete frequencies f_1, f_2 and f_3, respectively. In the final schedule, τ_2 terminates earlier again and spends ≈ 0.53 time units in the last interval. The energy consumption for this schedule is $E_4 = (0.8 \cdot 0.25^3 + 1.2 \cdot 0.5^3) + (0.4 \cdot 0.25^3 + 2.2 \cdot 0.5^3 + 0.53 \cdot 0.75^3) \approx 0.67$.

The example shows that greedily assigning slack to the current task as in Fig. 1b) is not always an optimal solution. A variant which distributes slack between the tasks as in Fig. 1c) performs better and shows an energy reduction of 40 % in our example. In contrast, the greedy strategy may have been better if the computation time demand of τ_2 tends to zero. This motivates our probabilistic approach to determine frequency schedules based on the expected energy of task execution which automatically chooses a greedy or conservative strategy. As actual task execution times can only be determined during run time, we took into account the necessity of a low complexity with respect to an online application.

2.3 General Scheduling Concept

In order to minimize the energy required for scheduling a task set, our proposed scheduling approach consists of three steps which are performed before the current task instance τ_X starts its execution: (a) Determine the available time S_X for τ_X. If τ_X will not consume more time than S_X for its execution, then deadlines will neither be violated for τ_X nor for subsequent tasks even in the worst-case scenario. (b) Select other tasks which may follow τ_X and determine their time demand S_Y as well as the number of execution cycles for the worst case $\check{Y}$ and the average case $\check{Y}_{AC}$. (c) Compute a valid schedule for executing the current task by the approach presented in this paper.

Step (a): To determine S_X, we use a slack approximation scheme based on [5]. Here, the system keeps track of the unused time U_i which is the (remaining) time reserved for the execution of each task. On task releases, the unused time U_i is initialized to $w_i = \mathcal{U}^{-1} \cdot \check{C}_i \cdot f_m^{-1}$ where $\mathcal{U} \leq 1$ is the given worst-case utilization factor of the system. The available time S_X for the current task is composed of its unused time and that of completed instances. During run time, the active task consumes unused times of instances in decreasing order of their priority, i.e., unused times of completed higher-priority tasks are consumed first, followed by unused time of the active task itself etc. Note that S_X is also the amount of time which can be given to τ_X so that no deadlines are violated for the current and for subsequent tasks even for the worst-case scenario.

Step (b): For reasons of efficiency it is not be possible to consider all task instances which follow the current task – an exact analysis would have to consider all instances in the hyperperiod. Therefore, at the current time t, we only consider a subset Γ_Y of tasks which will follow or interrupt τ_X. We consider task instances which have been preempted by τ_X and will be executed as soon as τ_X completes as well as task instances which will preempt τ_X in $[t, t + S_X]$. The allocated time S_Y for such instances is the sum of their unused times U_i. Note that S_X and S_Y can be determined in $O(n)$ according to [5]. We specify the cycle demand of Γ_Y for the average case and the worst case ($\check{Y}_{AC}$ and $\check{Y}$) as the sum of the average and worst-case cycle demand of the task instances in Γ_Y. Here, the expected cycle demand of an instance τ_i with $\check{C}_i$ worst-case cycles and $\check{R}_i$ worst-case remaining cycles is computed by $1/q_i(\check{C}_i - \check{R}_i) \cdot \left[\int_{\check{C}_i - \check{R}_i}^{\check{C}_i} q_i(z)\, dz\right]$ using the task specific execution profile q_i. Note that in Sect. 4.3, we will give implementation details and explain how we compute such integrals in $O(1)$.

We compute a new frequency schedule at the start and resumption of each task based on the real information of *when* the previous task terminated. However, it does not make sense to immediately compute a discrete frequency schedule for both the current task *and* the subsequent tasks in Γ_Y. Instead, we combine the tasks in Γ_Y into one 'virtual' subsequent task τ_Y which *models the system load in the near future*. For the time being, we heuristically assume that all subsequent tasks represented by τ_Y can be scheduled with one continuous frequency. In that way, the expected energy demand for the subsequent tasks is approximated and the concrete assignment for AFSs is deferred until a task instance is executed.

In the following, we will focus on step (c) and describe our model for the expected energy as well as the minimization problem and our solution.

2.4 Expected Energy and Time Demand

The current task τ_X will be executed with an accelerating frequency schedule $(\check{X}_1, \ldots, \check{X}_m)$ so that $\check{X}_i$ cycles are executed at frequency f_i for $i \in \{1, \ldots, m\}$. The expected energy demand for τ_X is modeled by

$$E_X(\check{X}_{1,..,m}) = \frac{1}{q(\check{C}-\check{R})} \sum_{i=1}^{m} \left[\frac{p(f_i)}{f_i} \int_{\check{C}-(\check{X}_{i+1}+\cdots+\check{X}_m)}^{\check{C}-(\check{X}_i+\cdots+\check{X}_m)} q(z)\, dz \right].$$

Similarly, the expected time demand is given by

$$X_{AC}(\check{X}_{1,..,m}) = \frac{1}{q(\check{C}-\check{R})} \sum_{i=1}^{m} \left[\frac{1}{f_i} \int_{\check{C}-(\check{X}_{i+1}+\cdots+\check{X}_m)}^{\check{C}-(\check{X}_i+\cdots+\check{X}_m)} q(z)\, dz \right].$$

The worst-case execution time $X(\check{X}_1, \ldots, \check{X}_m) = \sum_{i=1}^{m} \check{X}_i / f_i$ of an AFS must satisfy $X(\check{X}_1, \ldots, \check{X}_m) \leq S_X$.

For the virtual task τ_Y the worst-case time needed for the execution of $\check{Y}$ cycles at frequency f_Y is $Y(f_Y) = \check{Y} \cdot f_Y^{-1}$. The expected time demand for the execution of τ_Y is given by $Y_{AC}(f_Y) = \check{Y}_{AC} \cdot f_Y^{-1}$ which results in an energy demand of $E_Y(f_Y) = \frac{\check{Y}_{AC}}{f_Y} \cdot p(f_Y)$.

3. ENERGY MINIMIZATION PROBLEM

In order to find a schedule which minimizes the expected energy demand, we have to solve the following optimization problem:

$$\begin{aligned}
&\text{min.: } E(\check{X}_1, \ldots, \check{X}_m, f_Y) = E_X(\check{X}_1, \ldots, \check{X}_m) + E_Y(f_Y) \\
&\text{s.t.: } X(\check{X}_1, \ldots, \check{X}_m) \leq S_X \quad (a), \quad \check{X}_1 + \cdots + \check{X}_m = \check{R} \quad (b), \\
&\qquad \check{X}_i \geq 0 \text{ for } i = 1, \ldots, m \quad (c), \\
&\qquad Y(f_Y) \leq S_X + S_Y - X_{AC}(\check{X}_1, \ldots, \check{X}_m) \quad (d).
\end{aligned}$$

Constraints (a-c) guarantee a valid AFS for the worst-case execution of τ_X. With Constraint (d) we assume that τ_X has been executed with an average time demand $X_{AC}(\cdot)$ and now the worst-case execution time $Y(f_Y)$ of τ_Y must not exceed the remaining time. This also causes a distribution of this slack between the current task and the subsequent virtual task (cp. Fig. 1c)): Assume that the system load in the near future (modeled by τ_Y) is high and thus f_Y has to be high in order to fulfill condition (d) with $(\check{X}_1, ..., \check{X}_m)$. In this case, the expected energy consumption may be reduced by accelerating the execution time of τ_X which in turn allows a reduction of f_Y and thus $E_Y(f_Y)$.

Before, we analyze the optimization problem we transform it to a standard form so that g_0 represents the objective function and $g_i \leq 0$ with $i = 1, \ldots, m+2$ represent inequality constraints by performing the following steps:
(a) We use the equality constraint $\sum_{i=1}^{m} \check{X}_i = \check{R}$ and substitute every occurrence of $\check{X}_1$ using $\check{X}_1 = \check{R} - \sum_{i=2}^{m} \check{X}_i$.
(b) We replace the equality constraint $\sum_{i=1}^{m} \check{X}_i = \check{R}$ and $\check{X}_1 \geq 0$ by $\sum_{i=2}^{m} \check{X}_i \leq \check{R}$. Thus, we obtain (with $S := S_X + S_Y$):

$$g_0(\check{X}_2, \ldots, \check{X}_m, f_Y) = E_X(\check{X}_2, \ldots, \check{X}_m) + E_Y(f_Y) \quad (1)$$

$$g_1(\check{X}_2, \ldots, \check{X}_m, f_Y) = X_{AC}(\check{X}_2, \ldots, \check{X}_m) + Y(f_Y) - S \quad (2)$$

$$g_2(\check{X}_2, \ldots, \check{X}_m, f_Y) = X(\check{X}_2, \ldots, \check{X}_m) - S_X \quad (3)$$

$$g_3(\check{X}_2, \ldots, \check{X}_m, f_Y) = -\check{R} + \sum_{i=2}^{m} \check{X}_i \quad (4)$$

$$g_i(\check{X}_2, \ldots, \check{X}_m, f_Y) = -\check{X}_{i-2}, \qquad \text{for } i = 4, \ldots, m+2 \quad (5)$$

By using the short notations $q^{(i)} := \frac{q\left(\check{C} - \sum_{j=i}^{m} \check{X}_j\right)}{q(\check{C}-\check{R})}$, $g_0^{(j)} := \sum_{i=2}^{j} q^{(i)} \left[\frac{p(f_i)}{f_i} - \frac{p(f_{i-1})}{f_{i-1}}\right]$, and $g_1^{(j)} := \sum_{i=2}^{j} q^{(i)} \left[f_i^{-1} - f_{i-1}^{-1}\right]$ we obtain the following gradients:

$$\begin{aligned}
\nabla g_0 &= \left(g_0^{(2)}, \ldots, g_0^{(m)}, \check{Y}_{AC}/f_Y^2 \cdot (\dot{p}(f_Y) \cdot f_Y - p(f_Y))\right)^T \\
\nabla g_1 &= \left(g_1^{(2)}, \ldots, g_1^{(m)}, -\check{Y}/f_Y^2\right)^T \\
\nabla g_2 &= \left(f_2^{-1} - f_1^{-1}, f_3^{-1} - f_1^{-1}, f_4^{-1} - f_1^{-1}, \ldots, 0\right)^T \\
\nabla g_3 &= (1, 1, 1, \ldots, 1, 0)^T \\
\nabla g_{i+3} &= (0, \ldots, \underset{(i^{\text{th}} \text{ element})}{-1}, \ldots, 0)^T \quad \text{for } i = 1, \ldots, m-1
\end{aligned}$$

By a closer analysis of the gradients we are able to prove that the following Karush-Kuhn-Tucker (KKT) condition holds [1]:

THEOREM 1. *If $x := (\check{X}_2, \ldots, \check{X}_m, f_Y)$ is a (locally) minimal solution to the given optimization problem, then there are $\mu_i \geq 0$ with $\nabla g_0(x) + \sum_{i=1}^{m+2} \mu_i \nabla g_i(x) = 0$ and $\mu_i g_i(x) = 0$ for $1 \leq i \leq m+2$.*

4. SOLUTION

To solve the the energy minimization problem given above, we first consider a 'relaxed problem' by omitting constraint g_2. Interestingly, we are able to solve the relaxed problem analytically and we can prove that the unique optimal solution needs at most two frequencies. If this solution fulfills $g_2 \leq 0$ (i.e. $X(\check{X}_2, \ldots, \check{X}_m) \leq S_X$), then we also have a solution to the original problem. If the solution to the relaxed problem does not fulfill $g_2 \leq 0$, we apply a heuristic method for solving the original problem with constraint $X(\check{X}_2, \ldots, \check{X}_m) = S_X$ (i.e. $g_2 = 0$, see Sect. 4.2).

4.1 Relaxed Energy Minimization Problem

For a solution $x = (\check{X}_2, \ldots, \check{X}_m, f_Y)$ with $X(\check{X}_2, \ldots, \check{X}_m) < S_X$ the available time is not greedily assigned to the current task but slack is conserved for subsequent tasks. In this interesting case it is $g_2(x) < 0$ and we obtain the same KKT condition according to Thm. 1 as for the relaxed problem (due to $\mu_2 g_2(x) = 0 \Rightarrow \mu_2 = 0$). By a careful (and non-trivial) analysis we can conclude the following lemma from Thm. 1 and from the strict convexity of the power function $p(f)$ (details of the proof are omitted due to lack of space):

LEMMA 2. *Each solution x to the relaxed optimization problem and each solution x' to the original problem with $g_2(x') < 0$ (i.e. $X(x') < S_X$) uses no more than two discrete frequencies. If two frequencies are used, then they are immediate neighbor frequencies.*

If we fix two discrete neighboring frequencies f_i and f_{i+1}, we have the advantage that the energy function only depends on one independent variable z: $\check{R} - z$ cycles are executed with frequency f_i, z cycles with f_{i+1} and the continuous frequency f_Y can be derived from Eqn. (2), since $g_1(x)$ has to be 0 for an optimal solution x. By using the first and second derivative of the energy function $E(z)$ and by considering the boundaries of the interval $[0, \check{R}]$ for z, we consequentially deduce three different conditions on z to find potential minimal solutions. Of course the solution changes with the available amount of time $S = S_X + S_Y$. For each of these conditions we consider the energy function (depending on S) under the constraint that the condition is fulfilled and we finally are able to derive a complete characterization for the (globally) minimal solutions. This result can be generalized to the case that the pair of discrete neighboring frequencies is not fixed.

Due to lack of space we omit details of the proof and summarize our results as follows:

THEOREM 3. *Let $F = \{f_1, \ldots, f_m\}$ be the set of available discrete frequencies with $f_i < f_j$ for $i < j$. Let x be a solution to the relaxed optimization problem or a solution to the original problem with $X(x) < S_X$. Let $\check{X}^\star_{AC} := \frac{1}{q(\check{C}-\check{R})} \int_{\check{C}-\check{R}}^{\check{C}} q(y)\, dy$, let the frequency $f^\star_{i,i+1}$ be defined as the (unique) solution to the equation $\dot{p}(f^\star_{i,i+1}) \cdot f^\star_{i,i+1} - p(f^\star_{i,i+1}) = \frac{\check{Y}}{\check{Y}_{AC}} \frac{(p(f_{i+1})/f_{i+1}) - (p(f_i)/f_i)}{(1/f_i) - (1/f_{i+1})}$, and let $K_{i,i+1} := \left[\frac{\check{X}^\star_{AC}}{f_{i+1}} + \frac{\check{Y}}{f^\star_{i,i+1}},\ \frac{\check{X}^\star_{AC}}{f_i} + \frac{\check{Y}}{f^\star_{i,i+1}} \right]$. Depending on the available amount of time S, x uses only frequencies from $F(S)$ given as follows:*

$$F(S) = \begin{cases} \{f_1\} & \text{if } S > \max(K_{1,2}) \\ \{f_1, f_2\} & \text{if } S \in K_{1,2} \\ \{f_2\} & \text{if } \min(K_{1,2}) > S > \max(K_{2,3}) \\ \{f_2, f_3\} & \text{if } S \in K_{2,3} \\ \vdots & \vdots \\ \{f_{m-1}, f_m\} & \text{if } S \in K_{m-1,m} \\ \{f_m\} & \text{if } \min(K_{m-1,m}) > S \end{cases}$$

Let $Q^{-1}(y)$ be the inverse of the strictly monotonic function $Q(y) = \int_0^y q(z)\, dz$, i.e., of the anti-derivative of the function $q(z)$.

If $F(S) = \{f_i, f_{i+1}\}$ and

$$z = \check{C} - Q^{-1}\left(\frac{\left(S - \check{Y}/f^\star_{i,i+1}\right) \cdot q(\check{C}-\check{R}) + \frac{Q(\check{C}-\check{R})}{f_i} - \frac{Q(\check{C})}{f_{i+1}}}{\frac{1}{f_i} - \frac{1}{f_{i+1}}} \right),$$

then $x = (0, \ldots, 0, \underbrace{\check{R} - z}_{\text{freq. } f_i}, \underbrace{z}_{\text{freq. } f_{i+1}}, 0, \ldots, 0, f^\star_{i,i+1})$.

If $F(S) = \{f_i\}$, then $x = (0, \ldots, 0, \underbrace{\check{R}}_{\text{freq. } f_i}, 0, \ldots, 0, f_Y)$ with frequency $f_Y = \check{Y} \cdot \left(S - \frac{\check{X}^\star_{AC}}{f_i} \right)^{-1}$.

4.2 General Energy Minimization Problem

The solution $x = (\check{X}_2, \ldots, \check{X}_m, f_Y)$ which is obtained under relaxed constraints is not valid if the worst-case time demand is higher than the available time ($X(\check{X}_2, \ldots, \check{X}_m) > S_X$). For this case we propose a heuristic fallback method so that hard deadlines can still be met. In order to satisfy $X(\check{X}_2, \ldots, \check{X}_m) \leq S_X$, we have to consider solutions which consume less time, i.e., schedules with the property $X(\check{X}_2, \ldots, \check{X}_m) = S_X$.

For a continuous frequency processor, it would be sufficient to schedule the task with the frequency $f = \check{R}/S_X$, and for discrete frequency processors, one could select two neighbor frequencies of f to execute the remaining cycles. However, we will try to do better and determine the expected energy demand of all $m-1$ two-frequency schedules x which satisfy $X(\check{X}_2, \ldots, \check{X}_m) = S_X$ and for which one frequency is an immediate neighbor of f. Note, the expected energy of such schedules can be determined efficiently with low costs as we will describe in Sect. 4.3 below. The distribution (z_a, z_b) of cycles wrt. each pair of two fixed frequencies f_a and f_b can be immediately computed by using $z_a/f_a + z_b/f_b = S_X$ and $z_a + z_b = \check{R}$. The schedule with the lowest expected energy is selected for scheduling the current task.

4.3 Implementation

An adequate data structure to represent the probability function $h(x)$ is a histogram-based representation as in Fig. 2a) which combines execution cycles in groups or bins, respectively. This histogram can be generated for a specific task by profiling methods and can be updated during run time if necessary. The trade-off between accuracy and storage space can be controlled by the number of bins b. An important advantage of such a data structure is that the function $q(x) = 1 - \int_0^x h(z)\, dz$ can be directly derived. As $h(x)$ is stepwise constant, the function $q(x)$ is stepwise linear decreasing as in Fig. 2b). Similarly, its anti-derivative $Q(x) = \int_0^x q(z)\, dz$ is a quadratic function in the specific bins as given in Fig. 2c).

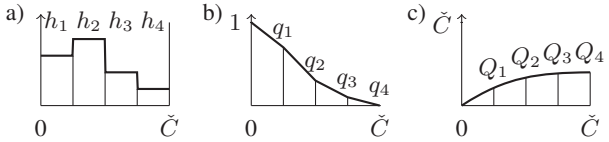

Figure 2: Probability Function

As a result it is convenient to pre-compute some function values at the bin borders q_i and Q_i and interpolate function values for $q(x)$, $Q(x)$ and $Q^{-1}(y)$ as described in the following. Let the execution profile be split into b equal-sized bins and let h_i be the probability that a task completes in the i^{th} bin. Then, we define $q_i := 1 - \sum_{k=1}^{i} h_k$ and $Q_i := \sum_{k=1}^{i} \check{C}/b \cdot ((q_k + q_{k-1})/2) = \check{C}/b \left[\left(\sum_{k=0}^{i} q_k \right) - q_0/2 - q_i/2 \right]$ for $i \in \{0, \ldots, b\}$. To compute $q(x)$ we first determine the corresponding bin $(j+1)$ so that $x \in [j\frac{\check{C}}{b}, (j+1)\frac{\check{C}}{b}]$ and define $x_l := j\frac{\check{C}}{b}$ for the left border of this interval. Then, we perform a linear interpolation $q_j(x) = q_j + (x - x_l) \cdot \frac{q_{j+1} - q_j}{\check{C}/b}$. Similarly, we can determine $Q_j(x) = Q_j + \int_{x_l}^{x} q_j(z)\, dz = Q_j + (q_{j+1} - q_j)\frac{b}{2\check{C}}(x - x_l)^2 + q_j(x - x_l)$. Vice versa, to compute the inverse $Q^{-1}(y)$, we first determine the cor-

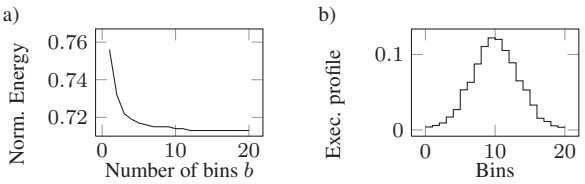

Figure 3: Number of bins and execution profile

responding interval j so that $y \in [Q_j, Q_{j+1}]$ and use $x = x_j(y) = x_l + \left(q_j - \sqrt{q_j^2 - 2(y - Q_j)\frac{q_j - q_{j+1}}{\check{C}/b}}\right) \cdot \frac{\check{C}/b}{q_j - q_{j+1}}$ as an explicit expression to find an x such that $Q_j(x) = y$.

4.4 Complexity

Slack estimation as well as the computation of the worst-case and expected cycle demand of subsequent tasks can be performed in $O(n)$ for n periodic tasks. To find an exact solution to the relaxed problem, we have to determine intervals $K_{i,i+1}$ (see Thm. 3). Each interval $K_{i,i+1}$ can be computed in $O(1)$, if an explicit formula is provided to solve for $f^\star_{i,j}$ in $\dot{p}(f^\star_{i,j}) \cdot f^\star_{i,j} - p(f^\star_{i,j}) = c$. (For a practical system, one would approximate the power function with a polynomial. Otherwise, as the term $\dot{p}(f^\star_{i,j}) \cdot f^\star_{i,j} - p(f^\star_{i,j})$ is strictly monotonic for strictly convex functions $p(f)$, it would also be convenient to provide sample points for this term and perform a binary search.) As the intervals $K_{i,i+1}$ do not overlap and are linearly ordered with i, the appropriate set $F(S)$ of frequencies according to Thm. 3 can be computed using binary search in $O(\log m)$. The interval boundaries need only be computed on-the-fly during the binary search. This leads to an overall complexity of $O(\log m)$ to compute $F(S)$. The complexity for determining the number of cycles z in Thm. 3 is $O(\log b)$ as the corresponding intervals for $Q^{-1}(y)$ and $Q(x)$ have to be found. If a fallback to the heuristic solution is used, then the complexity is $O(m)$ as $m - 1$ schedules are considered. Note that the computation of the integrals in $E_X(\dots)$ and $X_{AC}(\dots)$ for two-frequency schedules takes $O(1)$ using the pre-computed function values $Q_i(x)$ as explained in Sect. 4.3. Therefore, the overall complexity of our approach is $O(n + \log m + \log b + m) = O(n + m + \log b)$.

5. EXPERIMENTS

5.1 Experimental Setup

In our experiments we use a processor model with five discrete frequencies $F = \{150$ MHz, 400 MHz, 600 MHz, 800 MHz, 1000 MHz$\}$ with corresponding power values 80, 170, 400, 900 and 1600 mW as in [14, 12]. The overall system power consumption can be fitted with a curve $p_{\text{idle}} + p(f)$ using $p(f) = 1.55 \cdot 10^{-6} \frac{\text{mW}}{\text{MHz}^3} \cdot f^3$ and $p_{\text{idle}} = 60\,\text{mW}$.

For the simulations, we generate sets of periodic tasks Γ. The relative deadline of a task is equal to its period which is randomly chosen so that $T_i \in [10$ ms, 1000 ms$]$. We distribute the given worst-case utilization factor $\mathcal{U}$ of the task set among the tasks, so that $\mathcal{U} = \sum_{\tau_i \in \Gamma} \mathcal{U}_i$ and $\forall \tau_i \tau_j : \frac{1}{q} \cdot \mathcal{U}_j \leq \mathcal{U}_i \leq q \cdot \mathcal{U}_j$ for a given $q \in \mathbb{R}^+$. We choose $q = 3$ to avoid exceeding differences in task characteristics. Finally, we compute the worst-case number of execution cycles according to $\check{C}_i = \mathcal{U}_i \cdot T_i \cdot f_m$.

In this work, we model variances in the actual number of execution cycles with a Gaussian distribution: First, we provide a parameter $r = \check{B}/\check{C}$ to control the fraction of the best-case $\check{B}$ and worst-case execution cycles $\check{C}$. Second, for each task instance we generate random numbers $\check{A}$ with normal distribution for the execution cycles with a mean $\lambda = (\check{B} + \check{C})/2$ and a deviation of $\mu = (\check{C} - \check{B})/6$. If $\check{A} < \check{B}$ or $\check{A} > \check{C}$, we fix $\check{A}$ to $\check{B}$ or $\check{C}$, respectively.

The scheduling algorithm presented in this work is called 'Probabilistic Frequency Scheduling' (PFS). As we also want to determine the effect of computing the exact solution to the relaxed problem, we additionally consider a variant PFSFB which omits this step and directly proceeds with the heuristic method.

Moreover, we implemented various DVFS schedulers for comparison. We consider two *utilization-based approaches*: A static method (UNI) and a dynamic method *cycle-conserving EDF* (CC) [10]. UNI scales the processor to the lowest discrete frequency f_i for which $\frac{f_i}{f_m}$ is greater or equal to the worst-case utilization factor U of the task set. CC dynamically computes the actual utilization on task releases and completions, and scales to the corresponding frequency during run time.

Slack-based methods either consider the slack of higher-priority task instances (HP) [6] or both lower- and higher-priority task instances (LHP) [5]. In our implementation, we use [5] for both HP and LHP (for HP we omit the step for considering lower-priority tasks). If the execution of the current instance τ_X does not exceed the available time S_X, it is guaranteed that no task will miss its deadlines. For our experiments, the methods HP and LHP are combined with two frequency selection methods. The first method 'Next Higher' (NH) uses the lowest discrete frequency which is higher than or equal to $f = \check{R}/S_X$. The other method 'Worst-case split' (WCS) uses two discrete neighbor frequencies $f_i \leq f \leq f_{i+1}$ to split the execution into two intervals (with $\check{R} - \check{X}$ and $\check{X}$ cycles) so that $\frac{\check{R} - \check{X}}{f_i} + \frac{X}{f_{i+1}} = S_X$. These combined approaches are called HPNH, LHPNH, HPWCS, LHPWCS.

As *probabilistic methods*, we implemented two algorithms. The first approach is called 'Procrastinating' (PC) and considers the slack of lower- and higher-priority tasks. However, in contrast to the simpler methods above it makes use of the statistical profile to find an energy-minimal accelerating frequency schedule for a single task according to the approximation method described in [9]. The load caused by subsequent tasks is not considered and the computation is performed whenever a tasks starts or resumes. Another implemented method which uses execution profiles is 'Statistical Integrated' (SI). For a specific task set, statically optimal accelerating frequency schedules are pre-computed in an offline phase according to [12] and used to schedule the tasks during run time. Note that probabilistic methods as in [16] and [13] are not considered as they are restricted to a frame-based model.

5.2 Results

In a first step, we determined the number of bins b which are necessary to provide an adequate representation of the execution profile. The experiment in Fig. 3a) shows the energy-efficiency for a given number of bins b with $n = 10$ tasks, $\mathcal{U} = 0.8$ and a BCEC/WCEC ratio of $r = 1.0$. The diagram shows that the energy efficiency stabilizes for more than ≈ 12 bins. For our experiments, we used $b = 20$ bins as our benchmarks showed that a higher number of bins no longer gives significant improvements. A possible execution profile is given in 3b).

Simulation results for the energy efficiency are given in Fig. 4. Each diagram shows the normalized energy consumption on the y-axes (as a fraction of the energy consumption without DVFS, i.e., using the maximum frequency f_m). For the three diagrams we varied the number of tasks, the best-case to worst-case ratio r and the worst-case utilization factor $\mathcal{U}$. The results are averaged over 40 simulations, respectively.

First, we determined the energy-efficiency of our approach depending on of the number of tasks in Fig. 4a). For this experiment, we generated task sets with a worst-case utilization factor of $\mathcal{U} = 1$ and set $r = 0$ so that task execution cycles can vary between 0 and $\check{C}_i$. Such a setup results in an average utilization factor of $\mathcal{U}_{\text{AVG}} = 0.5$. However, as the worst-case utilization factor $\mathcal{U}$ is 1, any scheduler which precomputes accelerating frequency sched-

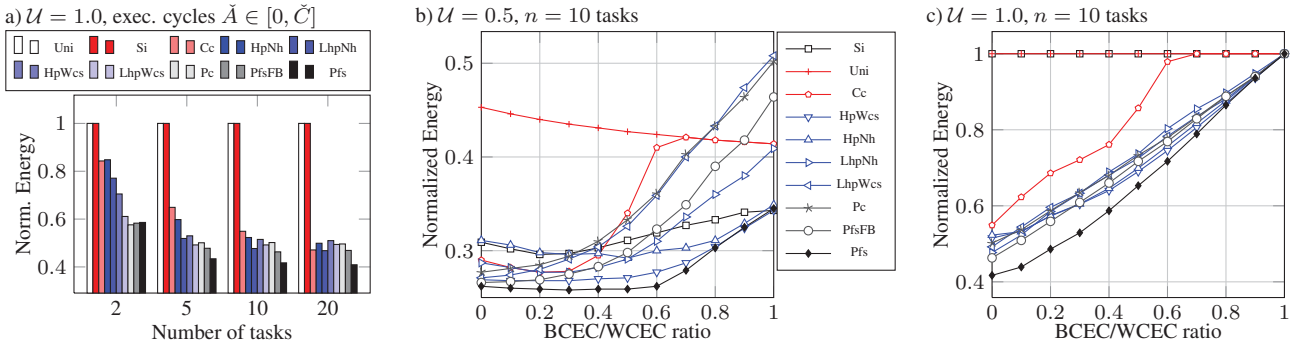

Figure 4: Energy efficiency wrt. number of tasks or variance of execution time

ules (SI) and/or which does not utilize online slack (UNI) *has to* use the highest frequency so that no deadlines are violated. Our schedulers PFS and PFSFB show the best overall performance among all schedulers, the next best competitors are PC and LHPWCS. If the number of tasks is increased, the energy-efficiency of all methods improves, especially for PFS which shows the highest energy-savings. The higher the number of tasks, the more situations occur where a starting or resuming task is allowed to use slack from previous tasks. As our probabilistic method includes the load of subsequent tasks it finds a balanced slack distribution which is neither too aggressive nor too conservative. For a task set with 20 tasks, e.g., PFSFB needs 14.8% more energy than PFS, LHPWCS needs 20.8% more energy than PFS, and PC needs 21.3% more energy. This observation is noteworthy, since compared to PFS much more complex online computations are needed for PC and PC produces schedules with m different frequencies instead of at most two.

In Fig. 4b), we used $n = 10$ tasks, a worst-case utilization factor of $\mathcal{U} = 0.5$ and varied the ratio $r = \check{B}/\check{C}$ to alter the number of best-case execution cycles $\check{B}$. Again, our method shows an overall good performance. Only for $r = 1$, SI shows slightly better performance than PFS. In this rather unrealistic scenario, tasks show no variances at all and slack redistribution during run time does not make sense. The pre-computed but optimal schedules of SI remain optimal. However, as soon as tasks show variances in their computation time, the slack analysis and balanced distribution among the tasks shows a better efficiency. For the scenario of Fig. 4b) the results of HPWCS are pretty close to the results of PFS, since it seems to achieve an appropriate degree of aggressiveness in frequency scaling by using slack of only higher-priority tasks and splitting the execution into two intervals. However, for other scenarios like Fig. 4c) with $\mathcal{U} = 1$ the efficiency of HPWCS can not compete with our approach PFS.

In Fig. 4c), we increased the worst-case utilization factor to $\mathcal{U} = 1$. Here, PFS again shows the best performance among all schedulers, especially for lower ratios $r = \check{B}/\check{C}$. A comparison of PFS with PFSFB which uses only our heuristic fallback method shows that providing the exact solution to the relaxed problem results in large energy savings. In general, we observed that the fallback ratio (the ratio of cases where we consider the heuristic solution in the second step) decreases with a higher number of tasks. For $n = 10$ tasks and $\mathcal{U} = 1$, the average fallback ratio was between 0.1 % and 39.5 %. For $n = 20$ tasks, the average fallback ratio was never higher than 21.8 %. This means that in most cases we are able to provide an optimal solution analytically without having to search for good solutions.

6. CONCLUSIONS

In this work, we proposed a method for estimating and minimizing the expected energy demand for real-time tasks which show variations in their execution times. In our model, we do not only consider the required computational time of single tasks in isolation but also the load of subsequent tasks. Moreover, we performed a theoretical analysis of the presented model and showed that it is possible to find optimal solutions for a subset of problem instances. For the remaining instances, we proposed a heuristic approach with low overhead. For each context switch our approach uses at most two switches of the processor frequency. Thus, the run time efficiency of our frequency scaling scheme is much better than that for straightforward generalizations of single-task approaches such as [9]. Although we restrict the number of frequency changes, our energy minimization outperforms related approaches from literature due to its ability to consider the future load that subsequent tasks place on the system. As a further interesting application one can also think about using our approach for more complex real-time systems which include aperiodic tasks.

7. REFERENCES

[1] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, September 1999.

[2] T. D. Burd and R. W. Brodersen. Energy efficient CMOS microprocessor design. In *Proc. of HICSS 1995*.

[3] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *Proc. of ISLPED 2001*.

[4] Intel. Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor, March 2004. http://www.intel.com/design/intarch/papers/301174.htm.

[5] W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proc. of DATE 2002*.

[6] C.-H. Lee and K. G. Shin. On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm. In *Proc of. RTSS 2004*.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[8] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. *SIGMETRICS Perform. Eval. Rev.*, 29(1):50–61, 2001.

[9] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron. Procrastinating voltage scheduling with discrete frequency sets. In *Proc. of DATE 2006*.

[10] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of SOSP 2001*.

[11] TI. Power Management Guide, 2010. http://power.ti.com (TI MSP430 & TPS780xx).

[12] C. Xian, Y.-H. Lu, and Z. Li. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(8):1467–1478, Aug. 2008.

[13] R. Xu, R. Melhem, and D. Mossé. A unified practical approach to stochastic DVS scheduling. In *Proc. of EMSOFT 2007*.

[14] R. Xu, C. Xi, R. Melhem, and D. Moss. Practical PACE for embedded systems. In *Proc. of EMSOFT 2004*.

[15] W. Yuan and K. Nahrstedt. Energy-efficient CPU scheduling for multimedia applications. *ACM Tran. Comp. Syst.*, 24(3):292–331, 2006.

[16] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. R. Stan. Optimal procrastinating voltage scheduling for hard real-time systems. In *Proc. of DAC 2005*.