Allocation Cost Minimization for Periodic Hard Real-Time Tasks in Energy-Constrained DVS Systems[*]

Jian-Jia Chen
Department of Computer Science and Information Engineering
National Taiwan University, Taiwan
r90079@csie.ntu.edu.tw

Tei-Wei Kuo
Department of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia
National Taiwan University, Taiwan
ktw@csie.ntu.edu.tw

ABSTRACT

Energy-efficiency and power-awareness for electronic systems have been important design issues in hardware and software implementations. We consider the scheduling of periodic hard real-time tasks along with the allocation of processors under a given energy constraint. Each processor type could be associated with its allocation cost. The objective of this work is to minimize the entire allocation cost of processors so that the timing and energy constraints are both satisfied. We develop approximation algorithms for processor types with continuous processor speeds or discrete processor speeds. The capability of the proposed algorithms was evaluated by a series of experiments, and it was shown that the proposed algorithms always derived solutions with system costs close to those of optimal solutions in the experiments.

Keywords: Energy-aware systems, Task scheduling, Real-time systems, Task partitioning, Multiprocessor synthesis, Dynamic voltage scaling.

1. INTRODUCTION

Performance boosting has been a highly important goal in system designs in the past decades. Not until recently, energy efficiency has become another critical feature being pursued in a wide variety of products, especially for battery-powered devices. How to explore the balance between the system performance and the power consumption triggers the advance of the Dynamic Voltage Scaling (DVS) technology, which provides a mean to adjust the supply voltage and thus the speed of microprocessors. Well-known example microprocessors are Intel StrongARM SA1100 and Intel XScale [14]. Technologies, such as Intel SpeedStep® and AMD PowerNOW!™, are also introduced for laptops to prolong the battery lifetime.

Energy-efficient designs and scheduling could not only extend the power-on duration of battery-driven devices but also help in cutting down the power bill of server systems significantly. In the past decade, energy-efficient task scheduling with various deadline constraints has received a lot of attention. Many scheduling algorithms has been proposed for uniprocessor task scheduling on DVS processors. Different heuristics were also proposed for energy consumption minimization under different task models in multiprocessor DVS environments. Beside the minimization of energy consumption, the maximization of the system performance under a given energy constraint is another important direction. In particular, researchers in [2, 13] targeted the maximization problem of the system reward under given timing and energy constraints. Alenawy et al. [1] considered the minimization problem of dynamic faults for soft real-time systems under a given energy constraint. Pruhs et al. [11] explored the performance metrics on the minimization of the average response time of a given job set on a processor under a given energy constraint.

Beside energy-efficient real-time task scheduling, energy-efficient designs are also explored in the minimization of the cost of allocated processors, referred to as the system-level synthesis problem of multiprocessor platforms. How to schedule real-time tasks in a heterogeneous processing environment with the minimization of allocation cost is explored in [15] without energy considerations. The synthesis problem in energy-efficient task scheduling of periodic hard real-time tasks was first explored under a given processor cost constraint in [8], where processors might have different costs. Different variations of the synthesis problems of energy-efficient task scheduling were discussed in [7]. When non-DVS processors are considered, Hsu et al. [6] developed approximation algorithms for the minimization of the allocation cost for periodic hard real-time tasks.

This paper targets energy-constrained scheduling of periodic hard real-time tasks. The goal is to assign each task to a *deployed* DVS processor such that the total cost of deployed processors is minimized, the energy and task timing constraints are satisfied, and each task executes at a valid speed. Note that processors under discussions here could be ASIC chips, such as those for the encoding or decoding of MPEG streams. The problem is $\mathcal{NP}$-hard even when there is only one available processor type. We show the non-existence of $(1.5-\epsilon)$-approximation algorithms, when there is only one processor type for any positive ϵ, and the non-existence of constant-ratio approximation algorithms, when the number of processor types is not a constant (unless $\mathcal{NP} = \mathcal{P}$). Polynomial-time algorithms are developed with a $(m + 2)$-approximation ratio when processors are associated with discrete available speeds, where m is the number of the available processor types. When there is only one processor type with continuously available speeds, we propose a 1.5-approximation algorithm with constraint violations and a 2-approximation algorithm. Extensions are then made to multiple processor types with continuously available speeds. Experimental results show that the proposed algorithms in this paper can derive solutions with system costs close to those of optimal solutions.

The rest of this paper is organized as follows: Section 2 defines the multiprocessor synthesis problems explored in this paper. Section 3 presents the proposed approximation algorithms for processor types with discrete processor speeds. Section 4 considers systems for processor types with continuous speeds. The experimental results for the performance evaluation of the proposed algorithm are presented

[*]Support in parts by research grants from ROC National Science Council NSC-94-2213-E-002-007 and NSC-94-2219-E-002-013.

in Section 5. Section 6 is the conclusion.

2. PROBLEM DEFINITION

Processor Models. This paper is interested in a synthesis problem for multiprocessor environments. We consider an environment with m different processor types. Let $\mathbf{M}$ be the set of the available processor types. For each processor type M_i in $\mathbf{M}$, an allocation cost C_i is associated, where C_i could be the corresponding price, area, or any property under considerations.

The power consumption function of a processor is mainly contributed by the dynamic power consumption resulting from the charging and discharging of gates on the DVS CMOS circuits. The dynamic power consumption function $P_i()$ of the dynamic voltage scaling part of processor type M_i in $\mathbf{M}$ is a function of the adopted processor speed s [12, §5.5]:

$$P_i(s) \propto V_{dd}^2 s, \tag{1}$$

where $s = \mu_i \frac{(V_{dd}-V_t)^2}{V_{dd}}$, and V_t, V_{dd}, and μ_i denote the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, and $\mu_i > 0$). $P_i()$ is a convex and increasing function of processor speeds. When V_t is 0, the dynamic power consumption function $P_i(s)$ can be rephrased as a cubic function of the processor speed s. The power consumption function can be phrased as a function proportional to s^η, where η is a hardware-dependent factor and $1 \leq \eta \leq 3$.

This paper considers processor types in which the charging and discharging of CMOS gates contribute the majority of the power consumption with negligible leakage and short-circuit power consumption. Hence, the power consumption function $P_i(s)$ is a strictly convex and increasing function of s for processor type M_i, so is the energy consumption $P_i(s)/s$ to execute a CPU cycle at speed s. The time and energy overheads on speed (voltage) switching are assumed to be negligible. The number of CPU cycles executed in a time interval is linear of the processor speed. That is, the number of CPU cycles completed in time interval $(t_1, t_2]$ is $\int_{t_1}^{t_2} s(t)\mathrm{d}t$, where $s(t)$ is the processor speed at time t. The energy consumed in $(t_1, t_2]$ is $\int_{t_1}^{t_2} P_i(s(t))\mathrm{d}t$ on a processor of processor type M_i.

In this study, we consider two DVS types of processors: (1) processors (of the processor type M_i) with a continuous spectrum of the available speeds between the upper-bounded speed $s_{i,\max}$ and 0, and (2) processors (of the processor type M_i) with K_i distinctive speeds, i.e., $(s_{i,1}, s_{i,2}, \ldots, s_{i,K_i})$, where $s_{i,1} < s_{i,2} < \cdots < s_{i,K_i}$ and $s_{i,\max}$ is the alias of s_{i,K_i}. The former type of processors is denoted by *ideal* processors while the latter type is denoted by *non-ideal* processors.

Task Models. We consider the scheduling problem of a set $\mathbf{T}$ of n periodic real-time tasks without dependency constraints. A periodic task is an infinite sequence of task instances, referred to as *jobs*, where each job of a task comes in a regular period. Each task τ_i in $\mathbf{T}$ is characterized by three parameters: its *computation requirement*, *period*, and *relative deadline*. $c_{i,j}$ denotes the worst-case execution cycles required to complete any job execution of task τ_i on one processor of processor type M_j. If there does not exist any implementations of task τ_i on M_j, we assume that $c_{i,j}$ is infinity. The period p_i of task τ_i is the minimal arrival interval between two consecutive jobs of the task. The relative deadline of task τ_i is the longest span of the time interval between the latest completion time and the release time of a job of τ_i. In this paper, the relative deadline of a task is assumed to be equal to the period of the task.

For both ideal and non-ideal processors, we are requested to assign an available execution speed for each task in $\mathbf{T}$ on the assigned processor. If $\frac{c_{i,j}}{s_{j,\max}} > p_i$ for some M_j, it is clear that executing τ_i on one processor of M_j will unavoidably let τ_i miss its deadline. For such a case, we just set $c_{i,j}$ as infinity. A task completes in time means that all the jobs of the task completes before their corresponding deadlines. The *hyper-period* of $\mathbf{T}$, denoted by L, is defined so that L/p_i is an integer for any task τ_i in $\mathbf{T}$. The number of jobs in the hyper-period of task τ_i is $\frac{L}{p_i}$. For example, L is the least common multiple (LCM) of the periods of tasks in $\mathbf{T}$ when the periods of tasks are all integers.

*Multiprocessor Allocation for Energy-Constrained Real-Time Task Scheduling (*MARTS*) Problem.* As shown in [10], the earliest-deadline-first (EDF) scheduling algorithm is an optimal uniprocessor scheduling algorithm for independent real-time tasks. A task set is schedulable if and only if the total utilization of the task set is no more than 100%, where the *utilization* of a task is defined as its execution time divided by its period. This paper considers the joint scheduling and allocation problem, in which EDF scheduling is applied to each allocated processor.

The problem considered in this paper is defined as follows: Consider a set $\mathbf{T}$ of n independent tasks over a set $\mathbf{M}$ of m different processor types. The available speeds and the power consumption function $P_j()$ of processor type M_j are specified. Each task $\tau_i \in \mathbf{T}$ is characterized by its period p_i, where the relative deadline of τ_i is equal to p_i. When τ_i is executed on one processor of processor type $M_j \in \mathbf{M}$, τ_i is associated with its execution cycle $c_{i,j}$ for each job execution on the processor type. Moreover, we are given an energy constraint $\mathcal{E}$ for the maximum energy consumption in the hyper-period L of $\mathbf{T}$.[1] The objective of the problem is to derive a schedule of $\mathbf{T}$ and a multi-subset of $\mathbf{M}$ for processor allocation such that each task in $\mathbf{T}$ is executed on an allocated processor and completes in time, the speed constraints are satisfied, the total energy consumption in the hyper-period of $\mathbf{T}$ does not exceed $\mathcal{E}$, and the total cost of allocated processors is minimized.

A solution is feasible for the MARTS problem if the energy constraint is satisfied, each task is assigned on an allocated processor, all the tasks complete in time, and each task is executed at an available speed on the allocated processor on which the task is assigned. An optimal solution for an input instance of the MARTS problem has the minimum cost on the allocated processors among all feasible solutions of the input instance.

LEMMA 1. *The* MARTS *problem is $\mathcal{NP}$-hard in a strong sense even when there is only one processor type for either ideal or non-ideal processors.*

PROOF. The MARTS problem can be proved $\mathcal{NP}$-hard in a strong sense by a reduction from the bin packing problem, which is $\mathcal{NP}$-complete in a strong sense [4]. □

We focus the study on approximation algorithms with worst-case guarantees of processor allocation cost. Based on [16], a polynomial-time α-approximation algorithm for the MARTS problem must have polynomial-time complexity of the input size and could derive a feasible solution with the total cost at most α times of an optimal solution, for any input instance allowing feasible solutions, in which α is also referred to as the approximation ratio of the approximation algorithm. We also adopt constraint violation approaches [9] by violating the maximum speed. An (α, β)-approximation for the MARTS problem derives a feasible solution with an α-approximation ratio while the maximum speed constraint on the resulting solution is relaxed by β times. The use of the constraint-violation approach [9] is to first set an artificial upper bound on the processor speed and then derive feasible solutions based on the relaxed constraint of processor

[1]It might be difficult to define the energy constraint in a hyper-period. In such a case, an average value P_{avg} on the power consumption constraint is assumed, while $\mathcal{E}$ is set as $P_{avg}L$.

speeds. The following theorem reveals the inapproximability result of the MARTS problem when $\mathcal{NP} \neq \mathcal{P}$.

THEOREM 1. *For any positive constant ϵ, there does not exist any $(1.5 - \epsilon)$-approximation of the* MARTS *problem when there is only one processor type for either ideal or non-ideal processors, unless $\mathcal{P} = \mathcal{NP}$.*

THEOREM 2. *There does not exist any polynomial-time approximation algorithm for the* MARTS *problem with a constant approximation ratio when m is not a constant, unless $\mathcal{NP} = \mathcal{P}$, .*

PROOF. The detail proofs of the above theorems are in [3]. □

3. ALGORITHMS FOR NON-IDEAL PROCESSORS

This section considers the MARTS problem with non-ideal processors. The MARTS problem is first formulated as an integer linear programming problem, and a series of relaxations is then performed to derive a feasible schedule and a proper allocation of processors in polynomial time. We will show that the proposed algorithms could derive approximated solutions with the worst-case guarantees.

Suppose that the number of allocated processors of processor type M_j is a non-negative integer Δ_j. Clearly, Δ_j is no more than n. For each task τ_i in $\mathbf{T}$, a binary variable $z_{i,j,k,\ell}$ is set as 1 if τ_i is assigned to execute on the k-th allocated processor of processor type M_j at speed $s_{j,\ell}$; otherwise, $z_{i,j,k,\ell} = 0$. The set $\mathbf{T}_{j,k}$ of tasks assigned onto the k-th allocated processor of M_j is schedulable by EDF if $\sum_{\tau_i \in \mathbf{T}_{j,k}} \sum_{\ell=1}^{K_j} u_{i,j,\ell} \cdot z_{i,j,k,\ell} \leq 1$, where $u_{i,j,\ell}$ is the utilization of τ_i on a processor of M_j at speed $s_{j,\ell}$, i.e., $u_{i,j,\ell} = \frac{c_{i,j}}{p_i \cdot s_{j,\ell}}$. The MARTS problem is formulated as an integer linear programming problem as follows:[2]

$$\begin{aligned}
&\text{minimize } \sum_{M_j \in \mathbf{M}} \Delta_j \cdot C_j \\
&\text{subject to} \\
&\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}} \sum_{k=1}^{n} \sum_{\ell=1}^{K_j} E_{i,j,\ell} \cdot z_{i,j,k,\ell} \leq \mathcal{E}, \\
&\sum_{M_j \in \mathbf{M}} \sum_{k=1}^{\Delta_j} \sum_{\ell=1}^{K_j} z_{i,j,k,\ell} = 1 \quad , \forall \tau_i \in \mathbf{T}, \\
&\sum_{M_j \in \mathbf{M}} \sum_{k=\Delta_j+1}^{n} \sum_{\ell=1}^{K_j} z_{i,j,k,\ell} = 0 \quad , \forall \tau_i \in \mathbf{T}, \\
&\sum_{\tau_i \in \mathbf{T}} \sum_{\ell=1}^{K_j} u_{i,j,\ell} \cdot z_{i,j,k,\ell} \leq 1 \quad , \forall M_j \in \mathbf{M}, k = 1, \ldots, n, \\
&z_{i,j,k,\ell} \in \{0,1\}\, , \forall \tau_i \in \mathbf{T},\ \forall M_j \in \mathbf{M}, k = 1, \ldots, n, \\
&\qquad\qquad\qquad \ell = 1, \ldots, K_j, \\
&\Delta_j \in \{0, 1, \ldots, n\}\, , \forall M_j \in \mathbf{M},
\end{aligned} \tag{2}$$

where $E_{i,j,\ell}$ is the energy consumption to execute task τ_i at speed $s_{j,\ell}$ on processor type M_j in the hyper-period. That is, $E_{i,j,\ell} = P_j(s_{j,\ell}) \frac{c_{i,j}}{s_{j,\ell}} \frac{L}{p_i}$. In Equation (2), the first constraint requires that the total energy consumption of all the tasks is no more than the given energy constraint $\mathcal{E}$, the second and third constraints require that each task τ_i must execute on one allocated processor only, and the fourth constraint means that the total utilization of the tasks executing on one allocated processor must be no more than one (because of EDF scheduling).

Instead of looking for an optimal solution of Equation (2), we could derive an approximated solution in polynomial time by performing a series of relaxations for any input instance. The first relaxation will be on the objective function to reduce the number of variables required in the programming. For each task τ_i in $\mathbf{T}$, a binary variable $y_{i,j,\ell}$ is set as 1 if τ_i is assigned to execute on a processor of processor type M_j at speed $s_{j,\ell}$; otherwise, $y_{i,j,\ell} = 0$. $\left\lceil \sum_{i=1}^{n} \sum_{\ell=1}^{K_j} u_{i,j,\ell} \cdot y_{i,j,\ell} \right\rceil$ is an under-estimated number of the required number of processors of processor type M_j. However, for some processor speed $s_{j,\ell}$ of processor type M_j, executing task τ_i at such a speed might lead the utilization $u_{i,j,\ell}$ of task τ_i being greater than 1. Such a case must be eliminated. Suppose that $\kappa_{i,j}$ is the minimum index ℓ^* with $\frac{c_{i,j}}{p_i \cdot s_{j,\ell^*}} \leq 1$. If no such a speed exists for task τ_i on M_j, let $\kappa_{i,j}$ be $K_j + 1$. Equation (2) could be relaxed into the following integer linear programming problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{M_j \in \mathbf{M}} \left\lceil \sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,j}}^{K_j} u_{i,j,\ell} \cdot y_{i,j,\ell} \right\rceil C_j \\
\text{subject to} \quad & \sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,j}}^{K_j} E_{i,j,\ell} \cdot y_{i,j,\ell} \leq \mathcal{E}, \\
& \sum_{M_j \in \mathbf{M}} \sum_{\ell=\kappa_{i,j}}^{K_j} y_{i,j,\ell} = 1 \quad , \forall \tau_i \in \mathbf{T}, \text{ and} \\
& y_{i,j,\ell} \in \{0,1\}\, , \forall \tau_i \in \mathbf{T},\ \forall M_j \in \mathbf{M}, \kappa_{i,j} \leq \ell \leq K_j.
\end{aligned} \tag{3}$$

For any feasible solution of Equation (3), each task is assigned to exactly one processor type at a feasible processor speed. Let task set $\mathbf{T}_j$ be the set of the tasks in $\mathbf{T}$ assigned on the processors of processor type M_j for a solution of Equation (3), i.e., $\mathbf{T}_j = \{\tau_i \in \mathbf{T} \mid y_{i,j,\ell} = 1 \text{ for some } \kappa_{i,j} \leq \ell \leq K_j\}$. We adopt the first-fit strategy to assign tasks in $\mathbf{T}_j$ to processors of M_j (referred to as Algorithm FF). In each iteration, we assign an un-assigned task τ_i in $\mathbf{T}_j$ to an allocated processor at the speed $s_{j,\ell}$ with $y_{i,j,\ell} = 1$ if the resulting total utilization of the tasks assigned on the processor is no more than 100%. If no such an allocated processor exists, we must get a new processor of M_j and assign τ_i to the newly allocated processor. The time complexity of Algorithm FF is $O(|\mathbf{T}_j|^2)$. The energy consumption of the resulting solution is no more than $\mathcal{E}$. Algorithm FF was shown as a 2-approximation algorithm of the bin packing problem [16, §9]. The following lemma shows that the number of the allocated processors of a solution derived by Algorithm FF is at most $\max\{1, 2\sum_{\tau_i \in \mathbf{T}_j} \sum_{\ell=\kappa_{i,j}}^{K_j} u_{i,j,\ell} \cdot y_{i,j,\ell}\}$ for any $\mathbf{T}_j$.

LEMMA 2. *Given a task set $\mathbf{T}_j$, the number of the allocated processors of processor type M_j in Algorithm* FF *is at most* $\max\{1, 2\sum_{\tau_i \in \mathbf{T}_j} \sum_{\ell=\kappa_{i,j}}^{K_j} u_{i,j,\ell} \cdot y_{i,j,\ell}\}$.

PROOF. It could be proved by applying the proof for the first-fit algorithm in [16, §9]. The detail proof is in [3]. □

We might relax the integral constraints of $y_{i,j,\ell}$ so that $y_{i,j,\ell}$ could be any fractional number. However, the lower bound provided by an optimal solution of the naive relaxation might be far away from an optimal solution of the MARTS problem in the worst case, even for the case that $\kappa_{i,j} = 1$ and $K_j = 1$ for all τ_i in $\mathbf{T}$ and M_j in $\mathbf{M}$.

We show that we could relax Equation (3) in a parametric manner so that the gap between an optimal solution of Equation (3) and an optimal solution of the relaxed problem is bounded for any input instance. First, we re-index the available processor types in $\mathbf{M}$ so that $C_1 \leq C_2 \leq \cdots \leq C_m$. The idea behind the parametric relaxation of Equation (3) is that we restrict the solution of the input instance. When the parameter is specified as m', the solution of the input instance is not to use any processor of M_j with $j > m'$ and to use at least one processor of $M_{m'}$. Clearly, the minimum solution of the following integer programming problem among $m' = 1, 2, \ldots, m$

[2]This formulation might not be feasible for some solvers such as GLPK [5]. It can be reformulated into another one that is solvable for any ILP solver. However, the large amount of variables makes the programming intractable in most solvers. For example, even when K_j is 1, GLPK [5] takes more than a week for $m = 4$ and $n > 15$ and more than a day for $m = 4$ and $n > 10$ in our experiments. The formulation in Equation (2) is for the clearance of the relaxation procedures in this section.

is a lower bound of the MARTS problem:

$$\begin{aligned}
\text{minimize}\quad & \textstyle\sum_{j=1}^{m'-1}\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,j}}^{K_j} u_{i,j,\ell}\cdot y_{i,j,\ell}\cdot C_j \\
& + \left\lceil \textstyle\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,m'}}^{K_{m'}} u_{i,m',\ell}\cdot y_{i,m',\ell}\right\rceil \cdot C_{m'} \\
\text{subject to}\quad & \textstyle\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,m'}}^{K_{m'}} y_{i,m',\ell}\cdot u_{i,m',\ell} > 0, \\
& \textstyle\sum_{j=1}^{m'}\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,j}}^{K_j} E_{i,j,\ell}\cdot y_{i,j,\ell} \le \mathcal{E}, \\
& \textstyle\sum_{j=1}^{m'}\sum_{\ell=\kappa_{i,j}}^{K_j} y_{i,j,\ell} = 1 \ \ ,\forall \tau_i\in\mathbf{T},\ \text{and} \\
& y_{i,j,\ell}\in\{0,1\}, \forall\tau_i\in\mathbf{T},\ 1\le j\le m', \kappa_{i,j}\le\ell\le K_j,
\end{aligned} \tag{4}$$

where the first constraint guarantees that processor type $M_{m'}$ is used.

For each specified m', we relax Equation (4) by relaxing the integrals constraint of $y_{i,j,\ell}$ so that $y_{i,j,\ell}$ could be any non-negative fractional number. As a result, for each m', we could relax Equation (4) into the following two sub-equations:

$$\begin{aligned}
\text{minimize}\quad & \textstyle\sum_{j=1}^{m'}\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,j}}^{K_j} u_{i,j,\ell}\cdot y_{i,j,\ell}\cdot C_j \\
\text{subject to}\quad & \textstyle\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,m'}}^{K_{m'}} y_{i,m',\ell}\cdot u_{i,m',\ell} \ge 1, \\
& \textstyle\sum_{j=1}^{m'}\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,j}}^{K_j} E_{i,j,\ell}\cdot y_{i,j,\ell} \le \mathcal{E}, \\
& \textstyle\sum_{j=1}^{m'}\sum_{\ell=\kappa_{i,j}}^{K_j} y_{i,j,\ell} = 1 \ \ ,\forall \tau_i\in\mathbf{T},\ \text{and} \\
& y_{i,j,\ell}\ge 0, \forall\tau_i\in\mathbf{T},\ 1\le j\le m', \kappa_{i,j}\le\ell\le K_j.
\end{aligned} \tag{5a}$$

$$\begin{aligned}
\text{minimize}\quad & C_{m'} + \textstyle\sum_{j=1}^{m'-1}\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,j}}^{K_j} u_{i,j,\ell}\cdot y_{i,j,\ell}\cdot C_j \\
\text{subject to}\quad & \textstyle\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,m'}}^{K_{m'}} y_{i,m',\ell}\cdot u_{i,m',\ell} \le 1, \\
& \textstyle\sum_{j=1}^{m'}\sum_{\tau_i\in\mathbf{T}}\sum_{\ell=\kappa_{i,j}}^{K_j} E_{i,j,\ell}\cdot y_{i,j,\ell} \le \mathcal{E}, \\
& \textstyle\sum_{j=1}^{m'}\sum_{\ell=\kappa_{i,j}}^{K_j} y_{i,j,\ell} = 1 \ \ ,\forall \tau_i\in\mathbf{T},\ \text{and} \\
& y_{i,j,\ell}\ge 0, \forall\tau_i\in\mathbf{T},\ 1\le j\le m', \kappa_{i,j}\le\ell\le K_j.
\end{aligned} \tag{5b}$$

Our proposed algorithm, denoted as Algorithm ROUNDING, first derives a minimum feasible solution among the $2m$ equations of all combinations in Equation (5). Since Equations (5a) and (5b) are both standard linear programming problems, applying a polynomial-time linear programming solver, such as that in [5], could derive an optimal solution of Equation (5a) or Equation (5b) in polynomial time of n and m' for a fixed parameter m' if a feasible solution exists. If there does not exist any feasible solution for a fixed m' for Equation (5a) or Equation (5b), the infeasibility could also be determined in polynomial time. For a specified m', let $y_{i,j,\ell}$ be 0 for any $j > m'$, or $\ell < \kappa_{i,j}$ for notational brevity. If no feasible solution is derived from these $2m$ linear programming problems, there does not exist any feasible solution for such an input instance. Otherwise, let the variable assignment with the minimum value in the objective function of Equation (5) be $y^*_{i,j,\ell}$, while Y^* is the abbreviated vector for the variable assignment. (Ties are broken arbitrarily.)

Let $\mathbf{T}_{I,j}$ be $\{\tau_i \in \mathbf{T} \mid \exists 1 \le \ell \le K_j \text{ with } y^*_{i,j,\ell} = 1\}$ for $j = 1, 2, \dots, m$. A task in $\mathbf{T}_{I,j}$ is referred to as an *integral task*. For the other tasks that are not in $\mathbf{T}_{I,j}$ for any $j = 1, 2, \dots, m$, we denote these tasks as *fractional* tasks. Let $\mathbf{T}_F$ be the set of fractional tasks, i.e., $\mathbf{T}_F = \{\tau_i \in \mathbf{T} \mid \exists j, \ell \text{ with } 0 < y^*_{i,j,\ell} < 1\}$. For each fractional task τ_i in $\mathbf{T}_F$, we insert τ_i into $\mathbf{T}_{I,j^*}$ by executing τ_i on M_{j^*}, where M_{j^*} is the processor type M_j with $y^*_{i,j,\ell} > 0$ and the minimum $E_{i,j,\ell}$, i.e., $j^* = \arg_{j=1,2,\dots,m} \min_{\ell=1,\dots,K_j \text{ and } y^*_{i,j,\ell}>0} E_{i,j,\ell}$. For tasks in $\mathbf{T}_{I,j}$, let $y'_{i,j,\ell}$ be $\left\lceil \sum_{k=1}^{\ell} y^*_{i,j,k}\right\rceil - \left\lceil \sum_{k=1}^{\ell-1} y^*_{i,j,k}\right\rceil$. That is, $y'_{i,j,\ell}$ is 1 when ℓ is the minimum index with $y^*_{i,j,\ell} > 0$, and $y'_{i,j,\ell}$ is 0, otherwise. We then assign task set $\mathbf{T}_{I,j}$ by applying Algorithm FF to allocate processors of M_j and assign tasks onto the allocated processors of task set $\mathbf{T}_{I,j}$ at the speed $s_{j,\ell}$ with $y'_{i,j,\ell} = 1$ for $j = 1, 2, \dots, m$. After all, we schedule these tasks by applying EDF individually on each allocated processor at the assigned speed.

It is not difficult to see that Algorithm ROUNDING guarantees to derive a feasible solution of the MARTS problem if feasible solutions exist. The time complexity is $O(m\Psi + n^2)$, where Ψ is the time complexity for the applied linear programming solver. Algorithm ROUNDING is presented in Algorithm 1. The following theorem shows the approximation ratio of Algorithm ROUNDING.

Algorithm 1 : ROUNDING

```
1: sort processor types so that $C_1 \le C_2 \le \cdots \le C_m$;
2: let $Y^*$ be the vector of $y^*_{i,j,\ell}$s with the minimum objective value among
   feasible solutions of Equations (5a) and (5b) for $m' = 1, 2, \dots, m$;
3: $\mathbf{T}_{I,j} \leftarrow \{\tau_i \in \mathbf{T} \mid \exists 1 \le \ell \le K_j \text{ with } y^*_{i,j,\ell} = 1\}$, for $j =$
   $1, 2, \dots, m$;
4: $\mathbf{T}_F \leftarrow \mathbf{T} \setminus (\cup_{j=1}^{m} \mathbf{T}_{I,j})$;
5: for each $\tau_i$ in $\mathbf{T}_F$, $\mathbf{T}_{I,j^*} \leftarrow \mathbf{T}_{I,j^*} \cup \{\tau_i\}$, where $M_{j^*}$ is the processor
   type $M_j$ with $y^*_{i,j,\ell} > 0$ for some $\ell$ and the minimum $E_{i,j,\ell}$;
6: $y'_{i,j,\ell} \leftarrow \left\lceil \sum_{k=1}^{\ell} y^*_{i,j,k}\right\rceil - \left\lceil \sum_{k=1}^{\ell-1} y^*_{i,j,k}\right\rceil$, for each task $\tau_i$ in $\mathbf{T}_{I,j}$;
7: apply Algorithm FF to allocate processors of $M_j$ for the tasks $\tau_i$s of task
   set $\mathbf{T}_{I,j}$ at speed $s_{j,\ell}$ with $y'_{i,j,\ell} = 1$, for $j = 1, 2, \dots, m$;
```

THEOREM 3. *Algorithm* ROUNDING *is a polynomial-time* $(m+2)$*-approximation algorithm for the* MARTS *problem.*

PROOF. The proof is shown in [3]. ☐

Another algorithm, denoted as Algorithm E-ROUNDING, to enhance the quality of the derived solutions of Algorithm ROUNDING could be done as follows: Let $Y_1^*, Y_2^*, \dots, Y_\nabla^*$ be the vectors of $y_{i,j,\ell}$s for the feasible solutions of the $2m$ different linear programming problems in Equation (5). For each vector Y_k^* with $1 \le k \le \nabla$, we find a feasible allocation solution with the same allocation strategy of Algorithm ROUNDING. Algorithm E-ROUNDING selects the solution with the minimum resulting allocation cost among these ∇ solutions. It is not difficult to see that Algorithm E-ROUNDING performs no worse than Algorithm ROUNDING does, and, hence, Algorithm E-ROUNDING is also a $(m+2)$-approximation algorithm.

4. ALGORITHMS FOR IDEAL PROCESSORS

This section considers the MARTS problem for ideal processors. We first explore the case in which there is only one processor type. In such a case, the objective is to minimize the number of allocated processors. A lower bound on the allocation cost is first derived. Based on the lower-bound solution, approximation algorithms for ideal processors are developed. Algorithms presented in Section 3 are extended to cope with multiple processor types of ideal processors.

4.1 One Processor Type

For notational brevity, let c_i be the execution cycle of task τ_i on the processor type, the power consumption function be normalized as $P(s) = s^\eta$ with $1 \le \eta \le 3$, and the maximum available speed be $s_{\max}$. For the rest of this subsection, we only consider cases in which $c_i/p_i \le s_{\max}$ for every task τ_i in $\mathbf{T}$ since there does not exist any feasible solution for the MARTS problem for the other case.

Given a specified number $m^\dagger$ of processors of the processor type, where $m^\dagger \ge \sum_{\tau_i\in\mathbf{T}} \frac{c_i}{p_i s_{\max}}$, the lower bound of the minimum energy consumption to schedule tasks in $\mathbf{T}$ at speed no more than $s_{\max}$ can be derived as follows, where the total utilization is no more than $m^\dagger$, and the utilization of every task in $\mathbf{T}$ is no more than 1:[3] If $(\sum_{\tau_i\in\mathbf{T}} \frac{c_i}{p_i})/m^\dagger$ is less than $\frac{c_{i^*}}{p_{i^*}}$ of task τ_{i^*} in which τ_{i^*} is the task τ_i in $\mathbf{T}$ with the greatest c_i/p_i, we assign task τ_{i^*} at speed $\frac{c_{i^*}}{p_{i^*}}$ and perform the algorithm recursively by removing τ_{i^*} in $\mathbf{T}$ on the remaining $m^\dagger - 1$ processors until $\mathbf{T}$ consists of no tasks; otherwise, we assign all the tasks in $\mathbf{T}$ at speed $(\sum_{\tau_i\in\mathbf{T}} \frac{c_i}{p_i})/m^\dagger$. Let $\Phi(m^\dagger)$ be the total energy consumption of all the tasks in $\mathbf{T}$ following the

[3]By applying the well-known KKT optimality condition, it is not difficult to see the optimality.

Algorithm 2 : S-LEUF

```
1: find the integer $m^*$, and the estimated utilization of tasks in $\mathbf{T}$;
2: return "no feasible solution" **if** $m^*$ does not exist;
3: sort tasks in $\mathbf{T}$ in a non-increasing order of the estimated utilization;
4: $\hat{m} \leftarrow m^* - 1$;
5: **repeat**
6:     $\hat{m} \leftarrow \hat{m} + 1$;
7:     $U_1 \leftarrow \cdots \leftarrow U_{\hat{m}} \leftarrow 0$, and $\mathbf{T}_1 \leftarrow \cdots \leftarrow \mathbf{T}_{\hat{m}} \leftarrow \emptyset$;
8:     **for** $i \leftarrow 1$ to $|\mathbf{T}|$ **do**
9:         find the smallest $U_\ell$; (break ties by choosing the smallest $\ell$)
10:        $\mathbf{T}_\ell \leftarrow \mathbf{T}_\ell \cup \{\tau_i\}$ and $U_\ell \leftarrow U_\ell + u_i^*$;
11:    **for** $\ell \leftarrow 1$ to $\hat{m}$ **do**
12:        **for each** task $\tau_i \in \mathbf{T}_\ell$ **do**
13:            $t_i' \leftarrow t_i^* \times \frac{1}{U_\ell}$;
14:    let $S_{\hat{m}}$ be the EDF schedule to execute task $\tau_i$ in $\mathbf{T}_\ell$ $(1 \leq \ell \leq \hat{m})$ at
       the speed $c_i/t_i'$ on the $\ell$-th processor;
15: **until** the energy consumption of schedule $S_{\hat{m}} \leq \mathcal{E}$;
16: return the resulting EDF schedule $S_{\hat{m}}$;
```

speed assignment above. m^* is the smallest integer no less than $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i s_{\max}}$ such that $\Phi(m^*)$ is no more than $\mathcal{E}$ for input task set $\mathbf{T}$. That is, it is not possible for any schedule to complete all the tasks in $\mathbf{T}$ in time without violating the timing or energy constraint when we are given $m^* - 1$ homogeneous processors. Hence, if no such an m^* exists, no feasible schedule can be produced without violating the energy constraint $\mathcal{E}$.

For the rest of this subsection, we only focus on the case in which such an m^* exists. m^* is a lower bound on the number of required processors of an optimal schedule. For notational brevity, let t_i^* be the execution time according to the above speed assignment when the specified number of processors is m^*. The *estimated utilization* u_i^* of task τ_i in $\mathbf{T}$ is defined as t_i^*/p_i. Similar to Lemma 2, applying Algorithm FF according to the above estimated utilization results in a 2-approximation solution.

We now present a 1.5-approximation algorithm with constraint violation. Tasks in $\mathbf{T}$ are sorted in a non-increasing order of their estimated utilization. Let $\hat{m}$ be initialized as m^*. We adopt the *Largest-Estimated-Utilization-First* strategy to assign tasks by increasing the number of available processors $\hat{m}$, until the energy consumption of the resulting schedule is no more than $\mathcal{E}$. The proposed algorithm is illustrated in Algorithm 2 and denoted as Algorithm S-LEUF. Let $\mathbf{T}_\ell$ be the set of tasks assigned to the ℓ-th allocated processor. U_ℓ is the total estimated utilization of $\mathbf{T}_\ell$. Since the utilization of each allocated processor is 100%, and, hence, the EDF schedule on each allocated processor completes all the tasks in time. The performance guarantee of the algorithm is analyzed as follows.

LEMMA 3. *For any fixed $\hat{m}$, if there exists a task set $\mathbf{T}_{\ell^*}$ consisting of at least two tasks and $U_{\ell^*} > 1$, then $U_\ell \geq \frac{1}{2}U_{\ell^*}$ for any $\ell = 1, 2, \ldots, \hat{m}$.*

PROOF. The inequality $U_{\ell^*} \leq 2U_\ell$ holds since $u_j^* \leq U_\ell$ and $U_{\ell^*} - u_j^* \leq U_\ell$ (τ_j is the last task inserted into $\mathbf{T}_{\ell^*}$). □

THEOREM 4. *The number of processors required for the schedule returned by Algorithm S-LEUF is at most $1.189m^* + 1$.*

PROOF. The proof is omitted and shown in [3]. □

The time complexity of Algorithm S-LEUF is $O(|\mathbf{T}|^2 \log |\mathbf{T}|)$ by a straightforward implementation. It can be reduced to $O(|\mathbf{T}| \log^2 |\mathbf{T}|)$ by searching the values of m^* and $\hat{m}$ both in a binary search manner. We can have the following corollary.

COROLLARY 1. *Algorithm S-LEUF is a $(1.5, 2)$-approximation algorithm for the MARTS problem with constraint violations.*

PROOF. The speed violation bound comes from Lemma 3. The 1.5-approximation factor comes as follows: If $m^* = 1$, Algorithm S-LEUF derives an optimal schedule. As for the case that $m^* > 1$, $\hat{m} \leq 1.5m^*$ because of Theorem 4. □

If the maximum speed violation is not permitted, Algorithm S-LEUF can be revised into Algorithm RS-LEUF by changing Step 15 with "**until** the energy consumption of schedule $S_{\hat{m}}$ is no more than $\mathcal{E}$ and $c_i/t_i' \leq s_{\max} \forall \tau_i \in \mathbf{T}$". The time complexity remains, and the approximation ratio of Algorithm RS-LEUF is $2 - \frac{1}{n}$.

4.2 Multiple Processor Types

For multiple processor types, we can divide the available speeds into a user-defined spectrum with a number of discrete speeds on each processor type. After that, we apply Algorithm ROUNDING or E-ROUNDING to assign tasks in $\mathbf{T}$. For tasks assigned onto a processor type M_j, the energy constraint of these tasks on the processor type is the sum of the energy consumption of these tasks. Under the energy constraint of each M_j, we apply Algorithms S-LEUF or RS-LEUF to allocate processors of processor type M_j and schedule the assigned tasks.

5. PERFORMANCE SIMULATION

This section provides extensive evaluations of the proposed algorithms. The hyper-period was 1000. The number of jobs of task τ_i within the hyper-period, denoted by ζ_i, was an integer uniformly distributed in the range of $[1, 50]$. The period of task τ_i was set as $\frac{1000}{\zeta_i}$. The maximum speed $s_{j,\max}$ of processor type M_j was a random variable between $[1, 10]$. The power consumption function $P_j(s)$ of processor type M_j was $h_j s^{\eta_j}$, where h_j was random variable uniformly distributed in the range of $[0.1, 10]$, and η_j was in the range of $[2.5, 3]$. The execution cycles $c_{i,j}$ of jobs of task τ_i on processor type M_j was a random variable uniformly distributed in the range of $[1, \frac{1000}{\zeta_i \times s_{j,\max}}]$. For each processor type M_j, the cost C_j was an integral variable uniformly distributed in the range of $[100, 1000]$.

For non-ideal processor type M_j with a specified number K_j of available speeds, the lowest available speed $s_{j,1}$ was $0.2 \times s_{j,\max}$, while s_{j,K_j} was $s_{j,\max}$. $s_{j,\ell}$ was $s_{j,1} + (s_{j,\max} - s_{j,1})\frac{\ell-1}{K_j-1}$. For a specified *energy constraint ratio* f, the energy constraint $\mathcal{E}$ for a task set $\mathbf{T}$ on a set of processor types $\mathbf{M}$ was set as $(E_{\max} - E_{\min})f + E_{\min}$, where $E_{\min} = \sum_{\tau_i \in \mathbf{T}} E_{i,\min}$ and $E_{\max} = \sum_{\tau_i \in \mathbf{T}} E_{i,\max}$. $E_{i,\max}$ was $\max_{M_j : \frac{c_{i,j}}{p_i \cdot s_{j,\max}} \leq 1} P_j(s_{j,\max})\frac{c_{i,j}}{s_{j,\max}}\frac{L}{p_i}$, and $E_{i,\min}$ was $\min_{M_j : \kappa_{i,j} \leq K_j} P_j(s_{j,\kappa_{i,j}})\frac{c_{i,j}}{s_{j,\kappa_{i,j}}}\frac{L}{p_i}$ for non-ideal processor types or $E_{i,\min}$ was $\min_{M_j : \frac{c_{i,j}}{p_i} \leq s_{j,\max}} P_j(\frac{c_{i,j}}{p_i})p_i\frac{L}{p_i}$ for ideal processor types.

For non-ideal processor types, we had two types of experiments by varying the number of processor types and the number of tasks and by adjusting the energy constraint, while K_j for processor type M_j was an integral variable uniformly distributed in the range of $[2, 20]$. For the first type, the numbers of processor types and tasks were from 2 to 10 and 6 to 50, respectively. For the second type, we varied the energy constraint ratio from 0.05 to 1, stepped by 0.05, where the number of processor types was an integral variable in the range of $[2, 10]$, and the number of tasks was an integral variable in the range of $[2, 50]$. For systems with one ideal processor type, the number of tasks was from 5 to 50, and the energy constraint ratio was from 0.05 to 1, stepped by 0.05. Each configuration was evaluated with independent settings.

For non-ideal processor types, the *normalized allocation cost* of an algorithm for an input instance was the ratio of the allocation cost of a solution derived from the algorithm to that of the lower bound derived from the minimum cost among the $2m$ linear programming problems of Equation (5). For one ideal processor type, the *normalized allocation cost* of an algorithm for an input instance was the ratio of the allocation cost of a solution derived from the algorithm to the value of m^* derived in Algorithm S-LEUF.

Figure 1(a) shows the average normalized allocation cost of Algorithms ROUNDING and E-ROUNDING when the energy constraint

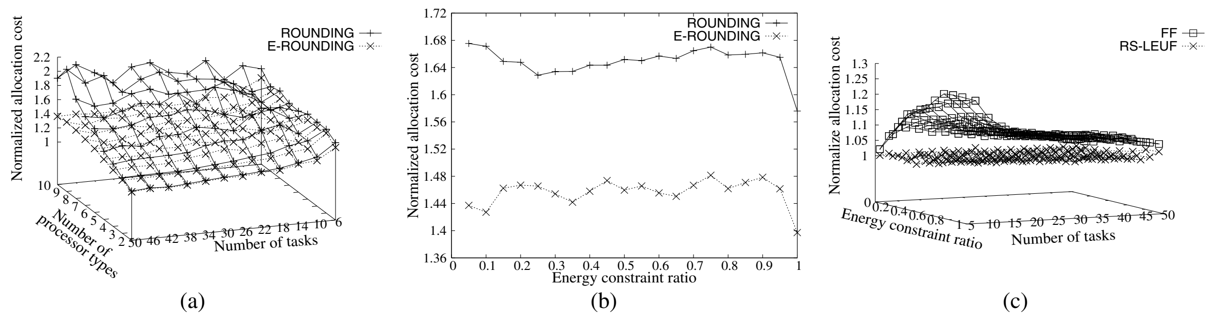

Figure 1: The experimental results: (a) the number of available non-ideal processor types ranged from 2 to 10, and the number of tasks ranged from 6 to 50 with energy constraint ratio equal to 0.2, (b) the energy constraint ratio ranged from 0.05 to 1, the number of available non-ideal processor types was an integer in $[2, 10]$, and the number of tasks was an integer in $[2, 50]$, and (c) one ideal processor type.

ratio was 0.2, the number of processor types varied from 2 to 10, and the number of tasks varied from 6 to 50 stepped by 4. The performance of E-ROUNDING was no worse than that of ROUNDING in the experimental results. Both of the proposed algorithms could derive solutions with costs close to those of optimal solutions. The performance gap between the two algorithms became wider for a larger number of processor types. In Figure 1(a), the average normalized allocation cost became larger when the number of processor types increased. However, the growing tendency was slow. Figure 1(b) shows the average normalized allocation cost of Algorithms ROUNDING and E-ROUNDING when the energy constraint ratio varied from 0.05 to 1 with a step equal to 0.05. Both of the proposed algorithms could derive solutions with costs close to those of optimal ones. Moreover, Algorithm E-ROUNDING outperformed Algorithm ROUNDING in all the cases.

Figure 1(c) shows the average normalized allocation cost of Algorithms FF and RS-LEUF proposed in Section 4 for one ideal processor type. Algorithm RS-LEUF derived solutions close to optimal ones. Algorithm RS-LEUF outperformed Algorithm FF greatly when the energy constraint ratio was large and the number of tasks was small, i.e., $f \geq 0.4$ and $n \leq 20$. We also simulated Algorithm S-LEUF. The results were very close to Algorithm RS-LEUF with slight violation on the speed when the energy constraint ratio was large. Simulation results for multiple types of ideal processors are omitted, due to the similar trends and the space limitation.

6. CONCLUSION

This paper targets energy-constrained scheduling of periodic hard real-time tasks. The goal is to minimize the total cost of processors, and the energy and task timing constraints are satisfied. The problem is $\mathcal{NP}$-hard even when there is only one available processor type. When the number of processor type is not a constant, there does not exist any constant-ratio approximation algorithm (unless $\mathcal{NP} = \mathcal{P}$). Polynomial-time algorithms are developed with a $(m+2)$-approximation ratio when processors have discrete available speeds, where m is the number of processor types. When there is only one processor type with continuously available speeds, we propose a 1.5-approximation algorithm with constraint violations and a 2-approximation algorithm. Extensions are made to multiple processor types with continuously available speeds. The proposed algorithms could also be applied to systems consisting of ideal and non-ideal processor types. The proposed algorithms were evaluated by a series of experiments, in which the system costs of the derived solutions were close to those of optimal solutions.

For future research, it is interesting to explore energy-constrained system synthesis for more general models, such as tasks with precedence constraints or processor types with non-negligible leakage power consumption in nano-meter manufacturing. The hardness results in this research still hold for these general cases. Moreover, for systems with non-negligible leakage power consumption, the derivation of a feasible solution to satisfy the energy constraint can be proved to be $\mathcal{NP}$-hard in a strong sense by a reduction from the 3-Partition problem [4], and, hence, constraint violation might be useful.

References

[1] T. A. Alenawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. In *Proceedings of the 26th IEEE Real-time Systems Symposium (RTSS'05)*, pages 376–385, 2005.

[2] J.-J. Chen and T.-W. Kuo. Voltage-scaling scheduling for periodic real-time tasks in reward maximization. In *the 26th IEEE Real-Time Systems Symposium (RTSS)*, pages 345–355, 2005.

[3] J.-J. Chen and T.-W. Kuo. Allocation cost minimization for periodic hard real-time tasks in energy-constrained DVS systems. Technical Report 0604, Department of Computer Science and Information Engineering, National Taiwan University, 2006.

[4] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Co., 1979.

[5] GNU Linear Programming Kit. http://www.gnu.org/software/glpk/glpk.html.

[6] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo. Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint. In *ACM/IEEE Conference of Design, Automation, and Test in Europe (DATE)*, pages 1061–1066, 2006.

[7] N. K. Jha. Low power system scheduling and synthesis. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 259–263, 2001.

[8] D. Kirovski and M. Potkonjak. System-level synthesis of low-power hard real-time systems. In *Proceedings of the 34th ACM/IEEE Conference on Design Automation Conference*, pages 697–702, 1997.

[9] J.-H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Symposium on Theory of Computing*, pages 771–782. ACM Press, 1992.

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

[11] K. Pruhs, P. Uthaisombut, and G. J. Woeginger. Getting the best response for your erg. In *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 14–25, 2004.

[12] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2002.

[13] C. Rusu, R. Melhem, and D. Mossé. Multiversion scheduling in rechargeable energy-aware real-time systems. In *EuroMicro Conference on Real-Time Systems (ECRTS'03)*, pages 95–104, 2003.

[14] INTEL-XSCALE, 2003. http://developer.intel.com/design/xscale/.

[15] Z. Shao, Q. Zhuge, X. Chun, and E. H.-M. Sha. Efficient assignment and scheduling for heterogeneous dsp systems. *IEEE Transaction on Parallel and Distributed Systems*, 16(6):516–525, June 2005.

[16] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.