Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems

Jian-Jia Chen, Chuan-Yue Yang, Tei-Wei Kuo, and Chi-Sheng Shih
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, ROC.
Email: {r90079, r92032, ktw, cshih}@csie.ntu.edu.tw

Abstract

Dynamic voltage scaling (DVS) circuits have been widely adopted in many computing systems to provide tradeoff between performance and power consumption. The effective use of energy could not only extend operation duration for hand-held devices but also cut down power bills of server systems. Moreover, while many chip makers are releasing multi-core chips and multiprocessor system-on-a-chips (SoCs), multiprocessor platforms for different applications become even more popular. Multiprocessor platforms could improve the system performance and accommodate the growing demand of computing power and the variety of application functionality. This paper summarizes our work on several important issues in energy-efficient scheduling for real-time tasks in multiprocessor DVS systems. Distinct from most previous work based on heuristics, we aim at the provision of approximated solutions with worst-case guarantees. The proposed algorithms are evaluated by a series of experiments to provide insights in system designs.

Keywords: Energy-Efficient Scheduling, Real-Time Systems, DVS, Multiprocessor Systems.

I. Introduction

With the advanced technology of VLSI circuit designs, a modern processor might operate at different supply voltages. Technologies, such as Intel SpeedStep® and AMD PowerNOW!™, provide dynamic voltage scaling (DVS) for laptops to prolong the battery lifetime. Different supply voltages lead to different execution speeds on a dynamic voltage scaling processor. For example, Intel StrongARM SA1100 processor [27] and the Intel XScale [28] are well-known DVS processors for embedded systems. The power consumption of processors in the dynamic voltage scaling is a convex and increasing function of processor speeds, and the function definition is highly dependent on the hardware designs. The lower the speed, the less the power consumption in the dynamic voltage scaling is.

For many applications, (average or worst-case) response time is an important non-functional requirement of the system. For example, embedded real-time systems must complete tasks before their deadlines to maintain the system stability. In the past decade, energy-efficient task scheduling with different deadline constraints has received a lot of attention. Many studies have been done for uniprocessor systems [4, 6, 11, 12, 19–22, 24, 31].

Recently, researchers have started exploring energy-efficient scheduling with the considerations of the non-negligible power consumption of leakage current for nano-meter manufacture process [21]. In such a direction, a processor might be turned off (or into a dormant mode) when needed. For uniprocessor scheduling of aperiodic real-time tasks, Irani et al. [19] proposed a 3-approximation algorithm for the minimization of energy consumption with the considerations of leakage current. For periodic real-time tasks, Jejurikar et al. [21] and Lee et al. [22] proposed energy-efficient strategies on a uniprocessor by applying the procrastination scheduling to decide when to turn the processor into the dormant mode.

Since many chip makers are releasing multi-core chips and multiprocessor system-on-a-chips (SoCs), multiprocessor platforms for different applications become even more popular. Multiprocessor platforms could improve the system performance and accommodate the growing demand of computing power and the variety of application functionality. Implementations of real-time systems with multiple processors are often more energy-efficient than those with a single processor [3], because of the convexity of power consumption functions. Although many results were proposed for uniprocessor energy-efficient scheduling, e.g., [6, 12, 19, 20, 31], with theoretical analysis, little work with theoretical analysis has been done for energy-efficient scheduling in multiprocessor systems.

Various heuristics were proposed for energy consumption minimization under different task models in multiprocessor environments [2, 5, 15, 16, 25, 32, 33]. In particular, several energy-efficient algorithms for task scheduling were proposed based on list heuristics [15, 16, 32] for real-time jobs with precedence constraints. There are also heuristics for periodic tasks in multiprocessor environments [2, 5]. Zhu et al. [33] explored on-line task scheduling with reclamation of slacks resulting from early completion of tasks during the run time. Mishra et al. [25] explored energy-efficient scheduling issues with the considerations of the communication delay of tasks. In addition to the considerations of energy-efficient scheduling, Anderson and Sanjoy [3] explored the trade-off between the total energy consumption of task executions and the number of required processors, where all of the tasks in the proposed solutions run at the same speed.

This paper summarizes our work in several issues in energy-efficient scheduling in multiprocessor DVS systems for periodic real-time tasks [1, 7–10, 17, 18, 30]. Two different topics for energy efficiency are included in this paper: (1) the minimization of energy consumption for real-time systems [7–9, 18, 30], and (2) the minimization of allocation cost of processors under a given energy constraint [10, 17]. For the minimization of energy consumption, we will first present our work for homogeneous multiprocessor DVS systems with negligible leakage power consumption when tasks have the same power consumption charac-

1-4244-0630-7/07/$20.00 ©2007 IEEE.

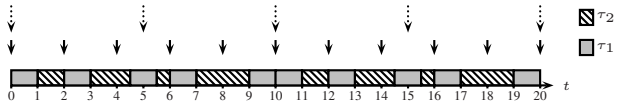
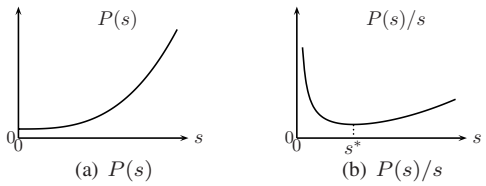


Fig. 1. An example for a set of periodic real-time tasks when $n = 2$ with $p_1 = 2$ and $p_2 = 5$.



(a) $P(s)$ (b) $P(s)/s$

Fig. 2. An illustrative example for $P(s)$ and $P(s)/s$

teristics [7, 30] and different power consumption characteristics [1, 9]. Then, we will show how to exploit the results to cope with systems with non-negligible leakage power consumption [8]. We will then present energy-efficient scheduling for systems with a DVS processor and a non-DVS processor [18]. We will then summarize our work for the minimization of allocation cost of processors under a given energy constraint when non-DVS processors [17] or DVS processors [10] are considered. Experimental results are presented to demonstrate the capability of the proposed algorithms.

The rest of this paper is organized as follows: Section II shows the system models. Section III summarizes our work for multiprocessor energy-efficient scheduling designs [1, 7–10, 17, 18, 30]. Section IV provides the performance evaluation of the proposed algorithms. Section V is the conclusion.

II. System Models

We explore the scheduling of periodic real-time tasks that are independent in execution. There is no precedence constraint among tasks. A periodic task τ_i is an infinite number of task instances (or jobs), in which a task is characterized by its initial arrival time a_i and its period p_i. The relative deadline of τ_i is equal to its period, so the arrival time and the absolute deadline of j-th job of τ_i are $a_i + (j-1) \cdot p_i$ and $a_i + j \cdot p_i$, respectively. The workload is measured in worst-case execution cycles, where the worst-case execution cycles of task τ_i is c_i. Let $\mathbf{T}$ be the set of n periodic real-time tasks. The *hyper-period* of $\mathbf{T}$, denoted by L, is the minimum positive number L so that L/p_i is an integer for every task τ_i in $\mathbf{T}$. 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. Figure 1 illustrates a set of task with two tasks, in which both tasks τ_1 and τ_2 arrive at time 0, p_1 is 2, and p_2 is 5. The execution time of task τ_1 is 1, and the execution time of task τ_2 is 2.5. The hyper-period L is 10, and, hence, the resulting schedule in Figure 1 in time interval $(10, 20]$ is the same as that in time interval $(0, 10]$.

The power consumption function $P(s)$ of the adopted speed s on a DVS processor can be divided into two parts: $P_d(s)$ and P_{ind}, where $P_d(s)$ is dependent and P_{ind} is independent upon the adopted speed. Leakage power consumption mainly contributes to P_{ind}, while the dynamic power consumption resulting from the charging of gates on a CMOS DVS processor and the short-circuit power consumption contribute to $P_d(s)$. The dynamic power consumption could be modeled as a convex function of the speed. For example, in CMOS DVS processors [26], the dynamic power consumption $P_{switch}(s)$ due to gate switching at speed s is

$$P_{switch}(s) = C_{ef} V_{dd}^2 s, \tag{1}$$

where $s = \kappa \frac{(V_{dd} - V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd}, and κ denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, $\kappa > 0$, and $C_{ef} > 0$). The short-circuit power consumption is proportional to the supply voltage. As a result, the speed-dependent power consumption function $P_d(s)$ is a convex and increasing function of the adopted speed. The power consumption function can model many power consumption models in [26, §5.5]. If the leakage power consumption is related to the speeds/voltages, i.e., not a constant, the leakage power is divided into two parts that contribute to $P_d(s)$ and P_{ind} accordingly. In other words, $P_d(s)$ models the voltage-related power consumption while P_{ind} models the voltage-independent power consumption.

The number of CPU cycles executed in a time interval is linear to the processor speed, and the energy τ_i consumed at the processor speed s for t time units is $t \cdot P(s)$. Moreover, the time and energy overheads on speed (voltage) switching are assumed to be negligible. We consider two DVS types of processors: (1) processors with a continuous spectrum of the available speeds between the upper-bounded speed $s_{\max}$ and the lower-bounded speed $s_{\min}$, and (2) processors with distinctive speeds. The former type of processors is called the *ideal* processors while the latter type is called *non-ideal* processors.

We explore two types of processors on handling the leakage current: (1) processors with a dormant mode, and (2) processors without any dormant mode. The former type is denoted as *dormant-enable* processors, while the latter is *dormant-disable* processors. For dormant-enable processors, the processor can be turned to the dormant mode (or be turned off). When the processor is turned to the dormant mode, the power consumption of the processor can be treated as 0 by scaling the speed-independent power consumption. To execute jobs, the processor has to be turned to the active mode. However, switching between the two modes requires time t_{sw} and energy E_{sw} overheads. For dormant-disable processors, the power consumption P_{ind} cannot be reduced, and, hence, to reduce the power consumption, we have to reduce the speed. For dormant-disable processors, we assume that $P_d(s)$ and $P_d(s)/s$ are both convex and increasing functions of s, and we replace $P(s)$ by $P_d(s)$.

To execute a cycle at speed s consumes $P(s)/s$ energy. For dormant-enable processors, there is a *critical speed*, denoted by s^*, among the available speeds, at which the processor executes a cycle with the minimum energy consumption. The critical speed s^* is the available speed s with the minimum first derivative of $P(s)/s$. Figure 2 illustrates the function $P(s)$ and the function $P(s)/s$ when $P(s)$ is $s^3 + \beta$.

For multiprocessor systems, a schedule might be a global schedule or a partition schedule. A global schedule allows different task instances of a task to be executed on different processors, while a partition schedule restricts all the task instances of a task to be executed on a processor. Here, we consider partition schedules, in which each task is executed on a processor. As shown in [23], the earliest-deadline-first (EDF) scheduling algo-

rithm is an optimal uniprocessor scheduling algorithm for independent real-time tasks. Here, on each processor, we apply the earliest-deadline-first scheduling algorithm to execute real-time jobs after task partition is done.

Since all the studied problems are $\mathcal{NP}$-hard, we focus on approximation algorithms with worst-case guarantees. A polynomial-time α-approximation algorithm for the minimization of energy consumption (allocation cost of processors, respectively) must have a polynomial-time complexity of the input size and could derive a feasible solution with the energy consumption (allocation cost of processors, respectively) at most α times of an optimal solution, for any input instance, in which α is also referred to as the *approximation ratio (bound)* of the approximation algorithm.

III. Scheduling Algorithms

Energy-efficient scheduling for periodic real-time tasks in multiprocessor systems is explored from two perspectives: (1) on the minimization of energy consumption with satisfaction of timing constraints in Section A, Section B, and Section C for different system models, and (2) on the minimization of allocation cost of processors under specified energy and timing constraints in Section D. The results of this section are based on the work in [1, 7–10, 17, 18, 30].

A. Energy-Efficient Scheduling for Homogeneous Multiprocessor Systems

For the minimization of energy consumption in homogeneous multiprocessor DVS systems, we consider systems with M ideal processors or cores, where every processor has the same architecture and capability. We assume that all of the M processors or cores can operate at any speed s in $[0, \infty)$ in this subsection. We first focus on the scheduling of a frame-based task set $\mathbf{T}$ [7, 9, 30] for different system models, in which each task τ_i in $\mathbf{T}$ arrives at time 0 and has a common deadline D. Extensions to periodic real-time tasks are then provided [1, 8]. We assume that the speed-independent power consumption is negligible in this section, where systems with non-negligible speed-independent power consumption will be presented in the next session. Every processor is dormant-disable in [1, 7, 9], while every processor is dormant-enable in [30].

Frame-based Real-Time Tasks Since partition schedules are considered in this paper, we have to first determine the task partition of $\mathbf{T}$ onto processors, and, then, derive a speed assignment of tasks without violating the timing constraints. We first show how to derive optimal speed assignments when the task partition is given, following approximation algorithms to determine the task partitions.

If all the processors can adjust their processor speeds independently, and all the tasks have the same power consumption characteristics, i.e., the same power consumption function, the speed assignment to minimize the energy consumption of the tasks assigned on a processor is to execute all the tasks at a common speed from time 0 to D [7]. If, at any time instant, all the processors must execute tasks at the same processor speed or be idle at speed 0, and all the tasks have the same power consumption characteristics, the speed assignment which minimizes the energy consumption under the task partition can be derived as follows [30]: (1) First, we re-index the task partition of $\mathbf{T}$ into M disjoint task sets $(\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M)$ so that the workload w_i of task set $\mathbf{T}_i$ is no more than the workload w_j of task set $\mathbf{T}_j$ with $j \geq i$, i.e., $w_i = \sum_{\tau_k \in \mathbf{T}_i} c_k \leq \sum_{\tau_k \in \mathbf{T}_j} c_k = w_j$. (2) By assuming $w_0 = 0$ for the boundary condition, solve the following problem:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{M} (M-i+1) P_d(\frac{w_i - w_{i-1}}{t_i}) t_i \\ \text{subject to} & \sum_{i=1}^{M} t_i = D, \end{array} \quad (2)$$

where t_i is a variable for determining the execution speeds for the i-th processor. The optimization problem in Equation (2) can be solved by adopting the Lagrange Multiplier Method, and then the i-th processor to execute tasks in $\mathbf{T}_i$ should operate at speed $(w_j - w_{j-1})/t_j$ in time interval $(\sum_{k=1}^{j-1} t_k, \sum_{k=1}^{j} t_k]$ for $j \leq i$ and be turned into the dormant mode at time $\sum_{k=1}^{i} t_k$.

Although deriving a task partition with the minimum energy consumption is $\mathcal{NP}$-hard [7, 30], the *Largest-Task-First* strategy, denoted by Algorithm LTF, is shown to be a good approximation strategy in deriving approximated solutions with worst-case guarantees. Algorithm LTF first sorts tasks in a non-increasing order of the worst-case execution cycles of tasks, i.e., $c_i \geq c_j$ for $i \leq j$, and, then, considers the tasks in the sorted order to assign the to-be-assigned task to the processor with the minimum workload so far. With the Largest-Task-First task partition strategy and the presented speed assignment approaches, the approximation ratio of the scheduling algorithm is 1.13 (2.371, respectively) if the operating speeds of processors can (can not, respectively) be adjusted independently [7, 30].

For tasks with different power consumption characteristics, i.e., different power consumption functions, applying Algorithm LTF and executing tasks on a processor at a common speed might be energy-inefficient since it does not take heterogeneous power consumption functions of the tasks into considerations. However, we can revise Algorithm LTF as follows to have approximation solutions [9]: (1) Derive an estimated speed assignment for $\mathbf{T}$ by treating that the available time for task execution is from time 0 to MD, in which each task executes no longer than D. (2) Sort the tasks in a non-increasing order of the estimated execution times, and, then, considers the tasks in the sorted order to assign the task to the processor with the minimum total estimated execution time so far. (3) Adjust the processor speed on each processor individually so that the energy consumption to execute tasks on a processor is minimized. The above algorithm is denoted by Algorithm LEET, which stands for the Largest-Estimated-Execution-Time-first strategy. Algorithm LEET is proved to have a 1.412-approximation ratio [9].

In [7], we show that deriving a feasible schedule under $s_{\max} \neq \infty$ is a $\mathcal{NP}$-Complete problem regardless energy minimization is pursued or not. Although we assume the processor speeds can go to infinity, applying Algorithm LTF can bound the violation on processor speeds by a constant factor as shown in [7]. For systems with the possibility on task rejection, we provide hardness analysis and heuristic algorithms in [13].

Periodic Real-Time Tasks The results for frame-based real-time tasks are then extended to periodic real-time tasks with slight modifications. The largest-task-first strategy in [7] is revised by considering tasks in the sorted order of the worst-case execution cycles of tasks divided by the period, i.e., $\frac{c_i}{p_i} \geq \frac{c_j}{p_j}$

for $i \leq j$ to the processor with the minimum workload, defined as the summation of $\frac{c_i}{p_i}$ of tasks τ_is on a processor, so far [8]. Similarly, Algorithm LEET is revised into Algorithm LEUF, stands for the Largest-Estimated-Utilization-First strategy. Algorithm LTF (Algorithm LEUF, respectively) is proved to have a 1.13-approximation (1.412-approximation, respectively) ratio for periodic real-time tasks with the same (different, respectively) power consumption characteristics in [8] (in [1], respectively).

B. Leakage-Aware Energy-Efficient Scheduling for Homogeneous Multiprocessor Systems

This section explores energy-efficient scheduling of periodic real-time tasks for homogeneous multiprocessor systems with non-negligible leakage power consumption, in which the processors are dormant-enable. We provide a 1.283-approximation algorithm when the energy E_{sw} of switching overhead is negligible and a 2-approximation algorithm when the energy of switching overhead is non-negligible [8]. For dormant-disable processors, the results in Section A can be directly adopted.

For systems with negligible energy of switching overhead, we will not execute tasks at any speed lower than the critical speed. The largest-task-first strategy is applied to perform task partition. Once the workload on a processor is smaller than the critical speed, the scheduler executes the tasks on the processor at the critical speed. Theoretical analysis shows that the above algorithm, denoted by Algorithm LA+LTF, is a 1.283-approximation algorithm.

For systems with non-negligible energy of switching overhead, the largest-task-scheduling strategy might lead into a schedule that uses many processors to execute tasks at the critical speed. Therefore, we have to reduce the number of processors that execute tasks at the critical speed so that we can reduce the energy consumption while the system is idle. First of all, we collect all the tasks that are executed at the critical speed, denoted as task set $\mathbf{T}^*$. Suppose that $\bar{m}$ is the number of processors that execute tasks at the critical speed. Initially, all of these $\bar{m}$ processors are marked as *unused*. In each iteration, we assign an un-assigned task τ_i in $\mathbf{T}^*$ to a processor marked as *used* if the resulting workload of the tasks, i.e., the summation of execution cycles divided by the periods, assigned on the processor is no more than the critical speed. If no such a processor exists, and all of these $\bar{m}$ processors are marked as *used*, then we return the task assignment derived from Algorithm LA+LTF; otherwise, we mark an unused processor as used and assign task τ_i to the processor. The above algorithm is denoted by Algorithm LA+LTF+FF, which is a 2-approximation algorithm. Moreover, after task assignment is done, we can also use the procrastination algorithm, denoted by Algorithm PROC, proposed in [21] on each processor to reduce the energy consumption.

C. Energy-Efficient Scheduling for Heterogeneous Two-Processor Systems

This section explores energy-efficient scheduling of periodic real-time tasks in a heterogeneous system with two processing elements (PEs). One is a processor with DVS capability, while the other is a processing element without DVS capability, denoted by non-DVS PE. If the energy consumption of the non-DVS PE depends on its workload, it is called a *workload-dependent PE*. Otherwise, it is called a *workload-independent PE*. The energy consumption of the workload-dependent non-DVS PE in the hyper-period is proportional to the utilization of tasks assigned onto it. That is, the energy consumption in such a case is $(P_2 \cdot L)U_2$, where P_2 is the power consumption of the non-DVS PE and U_2 is the total utilization of the tasks assigned onto it. The energy consumption of the workload-independent non-DVS PE is $P_2 \cdot L$. Here, we only present the results for ideal DVS processors [18], while the extensions can be simply made to non-DVS processors. For notational brevity, let u_i be the execution requirement of task τ_i on the non-DVS PE, while the utilization constraint on the non-DVS PE is 100%. Table I and Figure 3 provide an example when the non-DVS PE is a one-dimensional FPGA.

	τ_1	τ_2	τ_3	τ_4	τ_5
$\frac{c_i}{p_i}$	$\frac{3.5}{10}$	$\frac{2}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{1.5}{10}$
u_i	$\frac{1}{10}$	$\frac{1.5}{10}$	$\frac{2}{10}$	$\frac{2}{10}$	$\frac{3}{10}$

TABLE I
AN EXAMPLE FOR TASK PARAMETERS

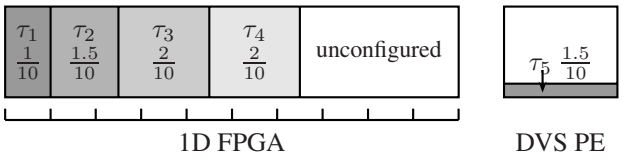


Fig. 3. An illustrative example for 1D FPGA Model

Algorithms for Workload-Independent non-DVS PE If a task has high computational demand on the DVS PE but low utilization on the non-DVS PE, it should be a good candidate to be assigned on the non-DVS PE to reduce the workload of tasks assigned on the DVS PE. For example, task τ_1 in Table I is a good candidate to be executed on the non-DVS PE, since its utilization on the non-DVS PE is low and its computation demand on the DVS PE is high. Therefore, an intuitive greedy algorithm, denoted by Algorithm GREEDY, is to sort tasks in $\mathbf{T}$ in a non-decreasing order of $\frac{u_i}{c_i/p_i}$, and then assign tasks in the sorted order to the non-DVS PE when the total utilization of the tasks assigned on the non-DVS PE does not violate the utilization constraint 100%. However, the resulting solution might be unbounded to an optimal solution.

Another point of view is to reformulate the optimization problem as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i} \cdot x_i \\ \text{subject to} & \sum_{\tau_i \in \mathbf{T}} u_i \cdot x_i \geq (\sum_{\tau_i \in \mathbf{T}} u_i) - 1, \quad \text{and} \\ & x_i \in \{0,1\} \quad , \forall \tau_i \in \mathbf{T}, \end{array} \tag{3}$$

where x_i is 0 if task τ_i is assigned on the non-DVS PE, and 1, otherwise. For brevity, let U^* be $(\sum_{\tau_i \in \mathbf{T}} u_i) - 1$. The optimization problem can be transformed into the minimum knapsack problem, which admits a 2-approximation algorithm [14], denoted by Algorithm E-GREEDY in this paper.

The ideas of Algorithm E-GREEDY are: (1) Sort tasks in a non-decreasing order of $\frac{c_i/p_i}{u_i}$. (2) Select the first k tasks on the DVS PE so that $\sum_{i=1}^{k} u_i \geq U^*$ and $\sum_{i=1}^{k-1} u_i < U^*$ as the initial solution, and evict τ_k from $\mathbf{T}$. (3) Seek the smallest index k' with

$$\sum_{\tau_i \in \mathbf{T} \text{ and } i \leq k'} u_i \geq U^*.$$

If $\sum_{\tau_i \in \mathbf{T} \text{ and } i \le k'} \frac{c_i}{p_i}$ is smaller than the best solution so far, we replace the best solution so far by the solution. (4) Evict task $\tau_{k'}$ from $\mathbf{T}$, and repeat Step (3) until the remaining tasks in $\mathbf{T}$ do not admit a feasible solution to Equation (3). After all, we assign all the tasks that are selected in the best solution in the above algorithm to the DVS PE and all the other tasks to the non-DVS PE. Then, the DVS PE executes the tasks assigned on it with the minimum energy consumption without violating the timing constraints. Algorithm E-GREEDY is shown to be a 8-approximation for the minimization of energy consumption.

Moreover, we also develop an approximation algorithm, denoted by Algorithm DP, that can trade the approximation ratio with the running time. The developed algorithm has a $(1+\delta)$-approximation ratio for any user-specified positive parameter δ with polynomial-time complexity by treating δ as the input. The basic idea is to scale up the execution cycle of tasks on the DVS PE so that we only have to construct a dynamic programming table in polynomial time.

Algorithms for Workload-Dependent non-DVS PE For workload-dependent non-DVS PEs, we present an algorithm that provides worst-case guarantees on the energy saving compared to that by assigning all the tasks on the DVS PE. Since the non-DVS PE is workload-dependent, we have to evaluate the reduction of energy consumption on the DVS PE and the increase of the energy consumption on the non-DVS PE, while considering the assignment of a task to the non-DVS PE. Sometimes, moving a task to the non-DVS PE is feasible, but the energy reduction on the DVS PE is less than the energy increase on the non-DVS PE. Algorithm S-GREEDY is an extension of Algorithm E-GREEDY with additional considerations.

Initially, tasks are sorted in a *non-increasing* order of $\frac{c_i/p_i}{u_i}$, which is different from Algorithm E-GREEDY. We put all the tasks on the DVS PE as the initial solution. According to the sorted order, we consider the assignment of task τ_i in the i-th iteration: According to the solution so far, if moving more portion of task τ_i to the non-DVS PE can reduce more energy consumption, then assign task τ_i to the non-DVS PE; otherwise, assign τ_i on the DVS PE. After n iterations, we can have a task assignment for task set $\mathbf{T}$. If we restrict the task assignment with at most one task on the non-DVS PE, we can find a task assignment with the minimum energy consumption under the restriction in $O(n)$. Algorithm S-GREEDY then chooses the better one between the two task assignments derived above. Algorithm S-GREEDY is shown to be a 0.5-approximation algorithm for the maximization of energy savings.

D. Allocation Cost Minimization under Energy Constraints

Beside energy-efficient real-time task scheduling for a given platform, another critical issue is on energy-aware synthesis to allocate processors and map tasks onto the allocated processors so that the energy and timing constraints can be satisfied. In this subsection, we provide a parametric relaxation methodology to provide approximation solutions for non-ideal processors [10, 17] and a greedy algorithm for ideal processors [10].

The problem considered in this section is as follows: Consider a set $\mathbf{T}$ of 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 $\hat{M}_j$ are specified. Each task $\tau_i \in \mathbf{T}$ arrives at time 0. When task τ_i is executed on one processor of processor type $\hat{M}_j \in \mathbf{M}$, task τ_i is associated with its execution cycle $c_{i,j}$ for each job execution on the processor type. The objective is to allocate processors of processor types in $\mathbf{M}$ with the minimum allocation cost along with a schedule of $\mathbf{T}$ on these allocated processors without violating timing constraints or the energy constraint $\mathcal{E}$ in the hyper-period.

Approximation Algorithms for Non-Ideal Processors The problem under considerations is shown to be a $\mathcal{NP}$-hard problem in a strong sense in [17]. Moreover, we also prove that there does not exist any polynomial-time approximation algorithm with a constant approximation ratio by providing an L-reduction. We formulate the problem into an integer linear programming (ILP) formulation. To solve the problem efficiently, the ILP formulation is relaxed into a linear programming (LP) formulation by allowing a task to be assigned on more than one processor type. However, we show that the naive ILP relaxation might be an unbounded relaxation. In [17], we provide a parametric relaxation technique on the ILP relaxation when there is only one speed on each processor, and, in [10], we extend the technique to DVS systems.

Suppose that $s_{j,\ell}$ is the ℓ-th slowest available speed on processor type $\hat{M}_j$, and F_j is the number of speeds for $\hat{M}_j$. Let C_j be the allocation cost of processor type $\hat{M}_j$, $E_{i,j,\ell}$ ($u_{i,j,\ell}$, respectively) be the energy consumption in the hyper-period (utilization, respectively) by executing task τ_i on processor type $\hat{M}_j$ at speed $s_{j,\ell}$, and $s_{j,\kappa_{i,j}}$ be the minimum available speeds to execute task τ_i on processor type $\hat{M}_j$ without violating the timing constraint, i.e., $u_{i,j,\ell} > 1$ for any $\ell < \kappa_{i,j}$. The variable $y_{i,j,\ell}$ denotes the portion of task τ_i to be executed on processor type $\hat{M}_j$ at speed $s_{j,\ell}$.

First, we re-index the available processor types in $\mathbf{M}$ so that $C_1 \le C_2 \le \cdots \le C_m$. The idea behind the parametric relaxation is that we restrict the solution of the input instance. When the parameter is specified as m', we are restricted to allocate at least one processor of processor type $\hat{M}_{m'}$ and none of processor type $\hat{M}_j$ with $j > m'$. For a fixed number m', we could relax the ILP into the following two sub-equations:

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{j=1}^{m'} \sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,j}}^{F_j} u_{i,j,\ell} \cdot y_{i,j,\ell} \cdot C_j \\
\text{subject to} \quad & \textstyle\sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,m'}}^{F_{m'}} y_{i,m',\ell} \cdot u_{i,m',\ell} \ge 1, \\
& \textstyle\sum_{j=1}^{m'} \sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,j}}^{F_j} E_{i,j,\ell} \cdot y_{i,j,\ell} \le \mathcal{E}, \\
& \textstyle\sum_{j=1}^{m'} \sum_{\ell=\kappa_{i,j}}^{F_j} y_{i,j,\ell} = 1 \quad, \forall \tau_i \in \mathbf{T}, \text{ and} \\
& y_{i,j,\ell} \ge 0, \forall \tau_i \in \mathbf{T}, 1 \le j \le m', \kappa_{i,j} \le \ell \le F_j.
\end{aligned} \tag{4a}$$

$$\begin{aligned}
\text{minimize} \quad & C_{m'} + \textstyle\sum_{j=1}^{m'-1} \sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,j}}^{F_j} u_{i,j,\ell} \cdot y_{i,j,\ell} \cdot C_j \\
\text{subject to} \quad & \textstyle\sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,m'}}^{F_{m'}} y_{i,m',\ell} \cdot u_{i,m',\ell} \le 1, \\
& \textstyle\sum_{j=1}^{m'} \sum_{\tau_i \in \mathbf{T}} \sum_{\ell=\kappa_{i,j}}^{F_j} E_{i,j,\ell} \cdot y_{i,j,\ell} \le \mathcal{E}, \\
& \textstyle\sum_{j=1}^{m'} \sum_{\ell=\kappa_{i,j}}^{F_j} y_{i,j,\ell} = 1 \quad, \forall \tau_i \in \mathbf{T}, \text{ and} \\
& y_{i,j,\ell} \ge 0, \forall \tau_i \in \mathbf{T}, 1 \le j \le m', \kappa_{i,j} \le \ell \le F_j.
\end{aligned} \tag{4b}$$

Algorithm ROUNDING first finds a feasible solution with the minimum value of the objective function among the $2m$ equations of all combinations in Equation (4). Let the variable assignment with the minimum value in the objective function of Equation (4) be $y^*_{i,j,\ell}$. For a task τ_i with $y^*_{i,j,\ell} = 1$ for some j and ℓ, we execute task τ_i on processor type $\hat{M}_j$ at speed $s_{j,\ell}$. For a task τ_i with $0 < y^*_{i,j,\ell} < 1$ for some j and ℓ, we execute

task τ_i on the processor type $\hat{M}_{j'}$, in which some $y^*_{i,j',\ell} > 0$ and the energy consumption $E_{i,j',\kappa_{i,j'}}$ is the minimum. Therefore, each task is assigned on a processor type. Then, for each processor type, we use the first-fit algorithm of the bin-packing problem [29, §9] to allocate processors without adjusting execution speeds so that the utilization on each processor is at most 100%. Algorithm E-ROUNDING enhances Algorithm ROUNDING by finding the schedule with the minimum allocation cost among the assignments based on the feasible solutions of the $2m$ equations. Theoretical analysis shows that both Algorithms ROUNDING and E-ROUNDING are with a $(m+2)$-approximation ratio, and the analysis is almost tight.

Approximation Algorithms for Ideal Processors For systems with ideal processors, we divide the available speeds into a user-defined spectrum with a number of discrete speeds on each processor type. Then, we apply Algorithm ROUNDING or E-ROUNDING to assign tasks. For tasks assigned onto a processor type, the energy constraint of these tasks on the processor type is the total energy consumption of these tasks. Under the energy constraint $\mathcal{E}_j$ of processor type $\hat{M}_j$, we use Algorithm RS-LEUF as follows to allocate processors of $\hat{M}_j$ and schedule the assigned tasks $\hat{\mathbf{T}}_j$.

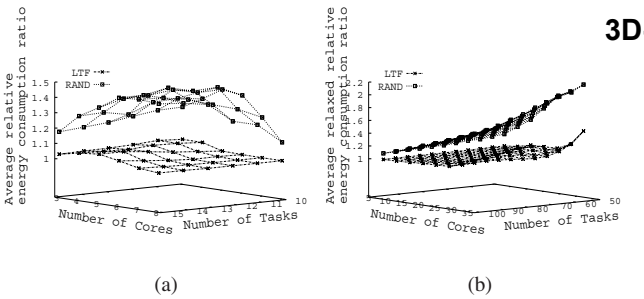
Let m^* be the minimum number of processors to execute tasks on the processor type without missing the timing or energy constraints by allowing a task to be executed simultaneously on more than one processor. The value of m^* can be obtained by applying the Kuhn-Tucker optimization condition in $O(|\hat{\mathbf{T}}_j|^2 \log |\hat{\mathbf{T}}_j|)$ time. Let t^*_i be the execution time for allowing simultaneous execution of tasks with m^* processors. Then, the *estimated utilization* u^*_i of task τ_i in $\hat{\mathbf{T}}_j$ is t^*_i/p_i. We could simply apply the first-fit algorithm of the bin-packing problem [29, §9] according to the above estimated utilization. However, the performance of the first-fit algorithm might not be good, since it does not intend to change the execution speeds of tasks. Instead, Algorithm RS-LEUF tries to change the execution speeds of some tasks so that we might reduce the number of processors required to meet the energy constraint. First, tasks in $\hat{\mathbf{T}}_j$ 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 without speed violation by increasing the number of available processors $\hat{m}$, until the energy consumption of the resulting schedule is no more than $\mathcal{E}_j$.

IV. Experimental Results

A series of simulations is conducted to evaluate the capability of the proposed algorithms. We consider systems with synthetic real-time tasks. The power consumption function is $\beta_1 + \beta_2 s^3$. For example, we can normalize the power consumption function of Intel XScale as $P(s) = 0.08 + 1.52s^3$ Watt by normalizing the highest available speed as 1.

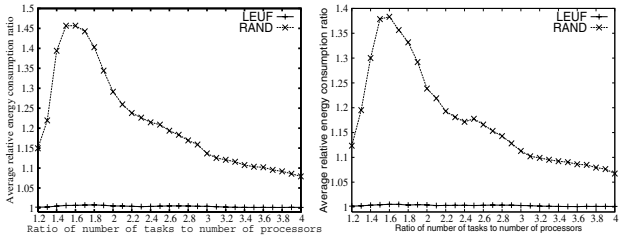
A. Homogeneous Multiprocessor Systems

Figure 4(a) shows the average relative energy consumption ratios for the simulated algorithms when $P(s) = s^3$, where the relative energy consumption ratio is defined as the energy consumption of the schedule derived from the obtained task assignment via the simulated algorithm divided by the energy consumption of the schedule derived from the optimal task assignment by exhaustive searches. Algorithm RAND is simulated for reference, where Algorithm RAND greedily assigns a task to a processor with minimum workload without sorting tasks. Figure 4(b) shows the average relaxed relative energy consumption ratios, which are obtained by dividing the energy consumption in the derived schedule by a lower bound. As shown in Figure 4, Algorithm LTF can derive solutions that are close to optimal.



(a) (b)

Fig. 4. The simulation results of Algorithm LTF and Algorithm RAND.



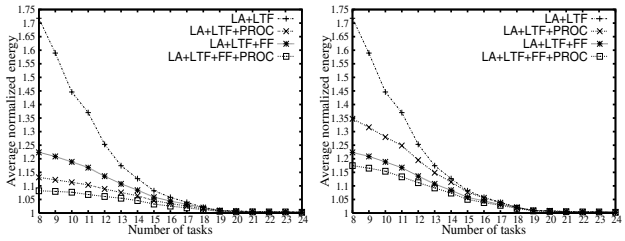
(a) Common exponent (b) Different exponent

Fig. 5. The simulation results of Algorithm LEUF and Algorithm RAND.

Figure 5(a) shows the average relative energy consumption ratios for the simulated algorithms when the power consumption function of task τ_i is $\rho_i s^3$. For a given ratio η of the number of tasks to the number of processors, the number of processors M is an integral random variable between 10 and 30, and the number of tasks is set as the floor of the multiplication of η and M, i.e., $\lfloor \eta \cdot M \rfloor$. The performance of Algorithm LEUF is very close to that of the optimal solutions. When the ratio of the number of tasks to the number of processors is small, both of Algorithm LEUF and Algorithm RAND might assign a task along with improper tasks on a processor. Such an assignment might result in a significant increase on the energy consumption of these tasks when the energy consumption for the other tasks are almost as the same as that in the optimal schedule. When the ratio of the number of tasks to the number of processors is small, in most cases, most processors are assigned with only one task, and the assignment is almost as the same as that of an optimal schedule. Therefore, the average energy consumption ratio is relatively small when the ratio of the number of tasks to the number of processors is less than 1.6. Figure 5(b) shows the average relative energy consumption ratios for the simulated algorithms when the power consumption function of task τ_i is $\rho_i s^{\alpha_i}$, where α_i is a random variable in $[2.5, 3]$.

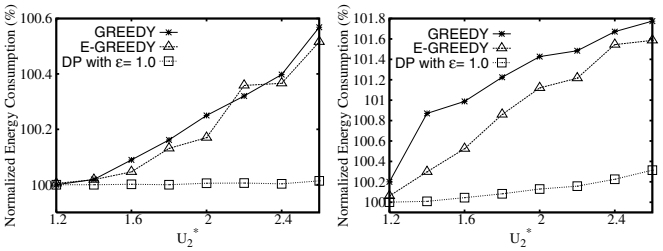
B. Leakage-Aware Homogeneous Multiprocessor Systems

Figure 6 shows the average energy consumption of the algorithms in Section III.B when there are 8 processors in the system, normalized to the energy consumption of a lower-bounded solution. Algorithm LA+LTF+FF+PROC denotes the algorithm by



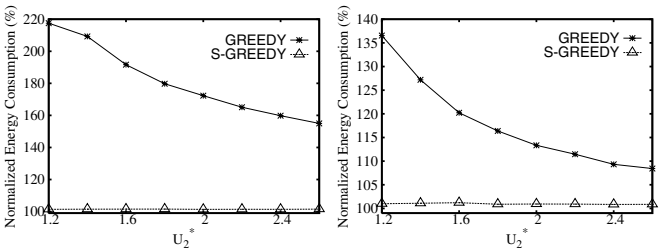
(a) $E_{sw} = 4$mJ (b) $E_{sw} = 12$mJ

Fig. 6. Simulation results when 8 processors.



(a) inverse (b) proportional

Fig. 7. An ideal DVS PE and a workload-independent non-DVS PE with $n = 10$ and $\epsilon = 1.0$.



(a) inverse (b) proportional

Fig. 8. An ideal DVS PE and a workload-dependent non-DVS PE with $n = 10$.

applying Algorithms LA+LTF, FF, and PROC accordingly. Algorithm LA+LTF+PROC is defined in a similar way. In both Figure 6(a) and Figure 6(b), Algorithm LA+LTF+FF+PROC always outperforms the other evaluated algorithms. Algorithm LA+LTF+PROC outperforms LA+LTF+FF in Figure 6(a), and vice versa in Figure 6(b). Since the break-even time when $E_{sw} = 4$mJ is less than that when $E_{sw} = 12$mJ, Algorithm PROC could save more energy when $E_{sw} = 4$mJ by turning a processor into the dormant mode.

C. Heterogeneous Two-Processor Systems

We perform evaluations for two different settings on task models. In the *proportional* model, the utilization of task τ_i on the non-DVS PE is a random variable proportional to the computation demand on the DVS PE. In the *inverse* model, the utilization of task τ_i on the non-DVS PE is less if its computation demand on the DVS PE is greater. We consider the non-DVS PE as an FPGA, where the Xilinx FPGA for the XC4VLX100 part with package FF1513 is adopted with 588mW power consumption. The variable U_2^* is the total utilization to execute all the tasks on the non-DVS PE.

Figure 7 shows the energy consumption for workload-independent non-DVS PEs, normalized to the energy consumption of the optimal solution by an exhaustive search. Algorithm E-GREEDY outperforms Algorithm GREEDY, and Algorithm DP outperforms all the other evaluated algorithms for workload-independent non-DVS PEs. As the utilization on the non-DVS PE becomes greater, Algorithm E-GREEDY and Algorithm GREEDY perform worse, i.e., with greater normalized energy consumption. This is because the execution requirement on the non-DVS PE for a task becomes larger, and the unused utilization with improper task assignments increases the system energy consumption significantly.

Figure 8 shows the results for workload-dependent non-DVS PEs. Algorithm S-GREEDY greatly outperforms Algorithm GREEDY. The reason why Algorithm GREEDY is worse in energy efficiency when U_2^* is small comes from that Algorithm GREEDY assigns tasks with too much utilization on the non-DVS PE.

D. Allocation Cost Minimization under Energy Constraints

Figure 9(a) shows the average normalized allocation cost (normalized to a lower bound) of Algorithms ROUNDING and E-ROUNDING when the energy constraint ratio is 0.2, the number of processor types varies from 2 to 10, and the number of tasks varies from 6 to 50. The energy constraint $\mathcal{E}$ for a task set $\mathbf{T}$ on a set of processor types $\mathbf{M}$ under an energy consumption ratio γ is set as $(E_{\max} - E_{\min})\gamma + E_{\min}$, where $E_{\min}$ ($E_{\max}$, respectively) are the minimum (maximum, respectively) energy consumption to complete the tasks. The performance of E-ROUNDING is 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 becomes wider for a larger number of processor types. Figure 9(b) shows the average normalized allocation cost of Algorithms ROUNDING and E-ROUNDING when the energy constraint ratio varies from 0.05 to 1. Both of the proposed algorithms could derive solutions with costs close to those of optimal ones. Moreover, Algorithm E-ROUNDING outperforms Algorithm ROUNDING in all the cases.

Figure 9(c) shows the average normalized allocation cost of Algorithms First-Fit and RS-LEUF for one ideal processor type. Algorithm RS-LEUF derives solutions close to optimal ones. Algorithm RS-LEUF outperforms Algorithm First-Fit greatly when the energy constraint ratio is large and the number of tasks is small, i.e., $\gamma \geq 0.4$ and $n \leq 20$.

V. CONCLUSION

Dynamic voltage scaling (DVS) circuits have been widely adopted in many computing systems to provide tradeoff between performance and power consumption. This paper presents energy-efficient algorithms for real-time tasks in multiprocessor DVS systems to provide approximation solutions. Energy-efficient scheduling is considered by two perspectives: (1) on the minimization of energy consumption for real-time systems [7–9, 18, 30], and (2) on the minimization of allocation cost of processors under a given energy constraint [10, 17]. For energy consumption minimization, we first present our work for homogeneous multiprocessor DVS systems with negligible leakage power consumption when tasks have the same power consumption characteristics [7, 30] and different power consumption characteristics [1, 9]. Then, we show how to exploit the results to cope with systems with non-negligible leakage power consumption [8]. We then present energy-efficient scheduling for sys-

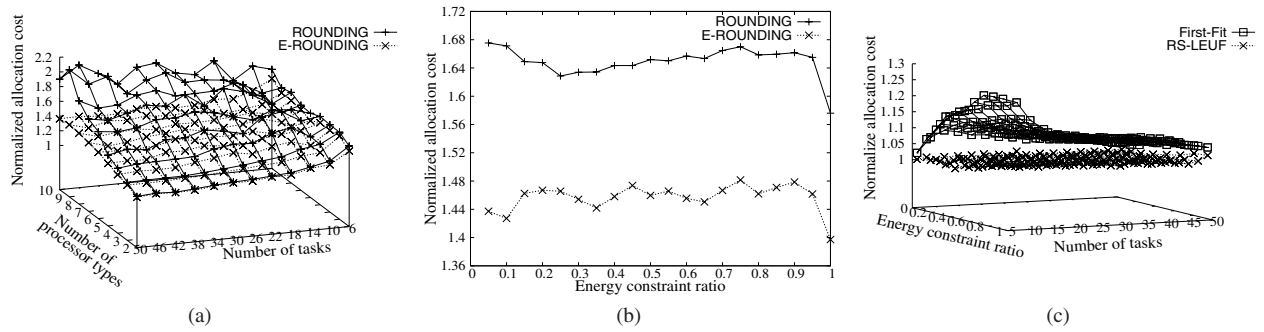


Fig. 9. The experimental results for the minimization of allocation cost.

tems with a DVS processor and a non-DVS processor [18]. We then summarize our work for the minimization of allocation cost of processors under a given energy constraint when non-DVS processors [17] or DVS processors [10] are considered. Experimental results are presented to demonstrate the capability of the proposed algorithms.

For future research, we would like to extend our research results to systems with real-time tasks with precedence constraints or resource competitions. We will also explore energy-efficient scheduling for multiprocessor systems with considerations on peripheral devices.

REFERENCES

[1] *Energy-Efficient Scheduling of Periodic Real-Time Tasks over Homogeneous Multiprocessors*, 2005.

[2] T. A. Alenawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. In *Proceedings of the 11th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'05)*, pages 213–223, 2005.

[3] J. H. Anderson and S. K. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 428–435, 2004.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the IEEE EuroMicro Conference on Real-Time Systems*, pages 225–232, 2001.

[5] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 113 – 121, 2003.

[6] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 520–529, 2004.

[7] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *EuroMicro Conference on Real-Time Systems (ECRTS'04)*, pages 101–108, 2004.

[8] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *IEEE Real-time and Embedded Technology and Applications Symposium*, pages 408–417, 2006.

[9] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks. In *International Conference on Parallel Processing (ICPP)*, pages 13–20, 2005.

[10] J.-J. Chen and T.-W. Kuo. Allocation cost minimization for periodic hard real-time tasks in energy-constrained DVS systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2006.

[11] J.-J. Chen, T.-W. Kuo, and H.-I. Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *Workshop on Algorithms and Data Structures (WADS)*, pages 338–349, 2005.

[12] J.-J. Chen, T.-W. Kuo, and C.-S. Shih. $1+\epsilon$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor. In *the 2nd ACM Conference on Embedded Software (EMSOFT)*, pages 247–250, 2005.

[13] J.-J. Chen, T.-W. Kuo, C.-L. Yang, and K.-J. King. Energy-efficient real-time task scheduling with task rejection. In *Proceedings of the 8th Conference of Design, Automation, and Test in Europe (DATE)*, 2007.

[14] G. Gens and E. Levner. *Computational complexity of approximation algorithms for combinatorial problems*. Springer, 1979.

[15] F. Gruian. System-level design methods for low-energy architectures containing variable voltage processors. In *Power-Aware Computing Systems*, pages 1–12, 2000.

[16] F. Gruian and K. Kuchcinski. Lenes: Task scheduling for low energy systems using variable supply voltage processors. In *Proceedings of Asia South Pacific Design Automation Conference*, pages 449–455, 2001.

[17] 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)*, 2006.

[18] C.-M. Hung, J.-J. Chen, and T.-W. Kuo. Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element. In *the 27th IEEE Real-Time Systems Symposium (RTSS)*, 2006.

[19] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 37–46, 2003.

[20] T. Ishihara and H. Yasuura. Voltage scheduling problems for dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 197–202, 1998.

[21] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, pages 275–280, 2004.

[22] Y.-H. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–112, 2003.

[23] 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.

[24] P. Mejía-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2):284–306, 2004.

[25] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem. Energy aware scheduling for distributed real-time systems. In *International Parallel and Distributed Processing Symposium*, page 21, 2003.

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

[27] INTEL. Strong ARM SA-1100 Microprocessor Developer's Manual, 2003. INTEL.

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

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

[30] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *Proceedings of the 8th Conference of Design, Automation, and Test in Europe (DATE)*, pages 468–473, 2005.

[31] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382. IEEE, 1995.

[32] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Annual ACM IEEE Design Automation Conference*, pages 183–188, 2002.

[33] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *Proceedings of IEEE 22th Real-Time System Symposium*, pages 84–94, 2001.