

Energy-Efficient Task Partition for Periodic Real-Time Tasks on Platforms with Dual Processing Elements*

Jian-Jia Chen and Lothar Thiele

Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
Email: jchen@tik.ee.ethz.ch, thiele@tik.ee.ethz.ch

Abstract

Modern computing systems often adopt multiple processing elements to enhance the computing capability or reduce the power consumption, especially for embedded systems. Such configurations impose challenges on energy efficiency in hardware and software implementations. This paper targets energy-efficient task partitioning for real-time tasks on a platform with two heterogeneous processing elements (processors), in which each one has its own characteristics on power consumption and job execution. This paper proposes a general framework for different hardware configurations in energy/power consumption. The framework provides a fully polynomial-time approximation scheme (FPTAS) to derive a solution with energy consumption very close to the optimal energy consumption in tolerable time/space complexity. Experimental results reveal that the proposed framework is effective in energy efficiency.

Keywords: Energy-efficient task partition, Real-time systems, DualCore systems.

1 Introduction

In the past decade, energy-efficient and low-power designs have become important issues in a wide range of computer systems. The pursuit of energy efficiency could be not only useful for mobile devices for the improvement on operating duration but also helpful for server systems for the reduction of power bills. *Dynamic voltage scaling* (DVS) is one of the most popular hardware methodologies in energy-efficient designs and has been widely adopted on many platforms. For example, Intel StrongARM SA1100 processor and Intel XScale are well-known processors for embedded systems. Moreover, technologies, such as Intel SpeedStep® and AMD PowerNOW!™, provide DVS for laptops to prolong the battery lifetime.

Due to the convexity of power consumption functions, we can trade performance for energy consumption. For real-time systems, the timing requirements must be satisfied, while the minimization of energy consumption is pur-

sued. In the past decade, energy-efficient scheduling of real-time tasks/jobs on uniprocessor DVS platforms has been widely explored, such as [2, 3, 5, 14, 16, 19, 25]. Specifically, periodic real-time tasks are considered in [2, 3, 19], while aperiodic real-time tasks are explored in [5, 14, 16, 25].

However, due to the dramatic increase on power density, multiprocessor platforms or platforms with co-processing elements have become more and more popular in architecture designs. For example, chip makers, such as Intel and AMD, are releasing multi-core chips to improve the system performance instead of increasing operating frequencies in desktop/laptop systems. On system-on-a-chip (SoC) platforms, a field-programmable gate array (FPGA) might be adopted to provide flexibility to execute tasks/jobs in hardware for acceleration. Some helper devices might also be used to reduce the workload on the processor for the enhancement of special functions, such as discrete cosine transform (DCT) or fast Fourier transform (FFT) functions. Such configurations in embedded systems trigger the researches in hardware/software co-designs to improve the system performance with energy-efficient considerations [13].

Energy-efficient task scheduling/partition in homogeneous multiprocessor systems has been studied extensively in the literature, e.g., [1, 4, 6, 9, 10, 20, 24, 27]. However, only few results have been developed for energy-efficient task partition in heterogeneous multiprocessor systems. Specifically, Yu and Prasanna [26] proposed an algorithm based on *Integer Linear Programming* (ILP) to minimize the energy consumption without guarantees on the schedulability of a derived solution for systems with discrete processor speeds. Huang, Tsai, and Chou [12] developed a greedy algorithm based on affinity to assign frame-based real-time tasks with re-assignment in pseudo polynomial-time to minimize the energy consumption when any processing speed can be assigned for a processor. Luo and Jha [18] proposed heuristics based on the list-scheduling strategy for real-time tasks with precedence constraints in heterogeneous distributed systems, while genetic list-scheduling algorithms were developed in [22]. Hsu et al. [11] and Chen and Kuo [7] developed approximation algorithms for the minimization of allocation cost of processors under energy constraints. However, the above research results for energy-efficient task partition in heterogeneous multiprocessor systems do not pro-

*This work is supported in parts by grants from ROC National Science Council NSC-096-2917-I-564-121 and the European Community's Seventh Framework Programme FP7/2007-2013 project Predator.

vide guarantees on quality or feasibility of the derived solutions. By contrast, Hung, Chen, and Kuo [13] considered platforms with two processing elements, in which one has the DVS capability and another does not, and developed approximation algorithms for different hardware models in terms of energy consumption minimization or energy saving maximization.

This paper explores energy-efficient task partition of periodic real-time tasks in a system with two heterogeneous processing elements (or processors). The power consumption functions of the processing elements are independent, so are the characteristics to execute tasks. Distinct from the configuration of the system in [13] with a DVS processing element and a non-DVS processing element, we focus on a very general system configuration, in which both processing elements might have the DVS capability. This paper develops a general framework to provide a fully polynomial-time approximation scheme (FPTAS) to derive a solution with energy consumption close to the optimal energy consumption in tolerable time/space complexity. It also provides approximated solutions with worst-case guarantees for the system models in [13], in which only approximation algorithms for the maximization of energy saving (compared to the solution by executing all the tasks on the DVS processing element) is provided. Experiments are performed to evaluate the performance of the proposed framework in terms of energy consumption minimization with deadline satisfaction. The results reveal the effectiveness in the minimization of energy consumption of the framework.

The rest of this paper is organized as follows: Section 2 defines the system model and the energy-efficient task partition problem in systems with two heterogeneous processing elements. Section 3 presents energy-efficient algorithms for task partition. The evaluation results are presented in Section 4. Section 5 concludes this paper.

2 System Models

2.1 Models of Processing Elements

The power consumption function $P(s)$ of speed s on a dynamic voltage scaling PE has two parts $P_d(s)$ and P_{ind} , where $P_d(s)$ (P_{ind} , respectively) is dependent (independent, respectively) on speed s . Leakage power consumption mainly contributes to P_{ind} , while the dynamic power consumption resulting from the charging/discharging of gates on a CMOS DVS PE and the short-circuit power consumption contribute to $P_d(s)$.

The speed-dependent power consumption function $P_d(s)$ could be modeled as a convex and increasing function of speed s . For example, the dynamic power consumption $P_{switch}(s)$, which dominates the power consumption in function $P_d(s)$ for PEs in micro-meter manufacturing, in CMOS DVS PEs due to gate switching at speed s is

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

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 sup-

ply voltage, and a hardware-design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, $\kappa > 0$, and $C_{ef} > 0$) [21]. Note that, if the leakage power consumption is related to the speeds/voltages, i.e., not a constant, the leakage power consumption is divided into two parts that contribute to $P_d(s)$ and P_{ind} accordingly. In other words, $P_d(s)$ models the voltage-dependent power consumption while P_{ind} models the voltage-independent power consumption [13].

Systems under considerations are with power consumption $P_d(s)$ as a polynomial function of s with exponents between 0 and 3, i.e., $P_d(s) = \sum_j b_j s^{\chi_j}$ for any $b_j > 0$ and $3 \geq \chi_j > 0$. As shown in the literature, the maximum exponent of $P_d(s)$ must be at least 1, and $P_d(s)$ is a convex and increasing function of s . The amount of cycles executed in an interval $(t_1, t_2]$ is $\int_{t_1}^{t_2} s(t) dt$, and the energy consumption is $\int_{t_1}^{t_2} P(s(t)) dt$, where $s(t)$ is the speed at time t . The energy consumption $(P_d(s) + P_{ind})/s$ to execute a cycle at speed s is a convex function in the available speed range. However, function $(P_d(s) + P_{ind})/s$ might not be an increasing function of speed s , and there is a *critical speed* s^* in which $(P_d(s) + P_{ind})/s$ is non-increasing for $s \leq s^*$ and non-decreasing for $s \geq s^*$ [6].

The power consumption function of speed s on the first (second, respectively) PE is denoted by $P_1(s)$ ($P_2(s)$, respectively). Let $s_{max,1}$ ($s_{max,2}$, respectively) be the maximum available speed on the first (second, respectively) PE. The critical speed for $P_1(s)/s$ is s_1^* , while that for $P_2(s)/s$ is s_2^* . A PE here is assumed to be able to use any speed not higher than its maximum available speed. Extensions can be easily achieved for PEs that have discrete speeds only or have a minimum available speed.

2.2 Task Model

This paper explores the scheduling of periodic real-time tasks that are independent in execution, in which there is no precedence constraint among tasks. A periodic task τ_i is an infinite number of jobs. A task is characterized by its initial arrival time a_i and its period p_i . The relative deadline of task τ_i is equal to its period. The worst-case execution cycles of task τ_i on the first (second, respectively) PE is $c_{i,1}$ ($c_{i,2}$, respectively). Let \mathbf{T} be the input task 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. Note that the hyper-period of the task set is used only to be the length of the time interval to evaluate the energy consumption. If it does not exist, we can use any other value that is large enough with the same analytical results.

2.3 Scheduling Policy and Energy Consumption

The earliest-deadline-first (EDF) policy is an optimal uniprocessor scheduling policy for independent real-time tasks. A set of tasks is schedulable by EDF if and only if the total utilization of the set of tasks is no more than

100%, where the *utilization* of a task is defined as its execution time divided by its period.

To execute a subset $\hat{\mathbf{T}}$ of the input task set \mathbf{T} on a DVS PE without violating the timing constraints of tasks in $\hat{\mathbf{T}}$, a schedule which minimizes the energy consumption executes all the tasks in $\hat{\mathbf{T}}$ at a common speed so that the total utilization is no more than 100% [3, 6]. We take the execution on the first PE for example. Here are two cases to calculate the optimal energy consumption to execute task set $\hat{\mathbf{T}}$ with $\hat{s} = \sum_{\tau_i \in \hat{\mathbf{T}}} \frac{c_{i,1}}{p_i}$ for a hyper-period L of \mathbf{T} on the first DVS PE: (1) If the DVS PE cannot be turned into the dormant mode due to the significant switching overhead [3], the optimal energy consumption is $L \cdot P_1(\hat{s})$. (2) If it is possible to turn the PE into the dormant mode with negligible switching overhead [6], the optimal energy consumption is $L \cdot P_1(\hat{s})$ when \hat{s} is more than the critical speed s_1^* and is $L \cdot \frac{\hat{s}}{s_1^*} P_1(s_1^*)$ when \hat{s} is no more than s_1^* . Hence, $P_1(s)$ can be revised into $P_1^*(s)$ so that the optimal energy consumption is $L \cdot P_1^*(s)$, where

$$P_1^*(s) = \begin{cases} P_1(s) & \text{if there is no dormant mode or } s \geq s_1^*; \\ \frac{s}{s_1^*} P_1(s_1^*) & \text{otherwise.} \end{cases}$$

Moreover, function $P_2^*(s)$ is defined similarly as above. Note that for systems with non-negligible switching overhead to the dormant mode, we can use the algorithms in this paper for task partition, and then use the procrastination algorithms in [6, 8, 15] for task procrastination.

The system model in this paper covers the configurations in [13]. When the power consumption $P_2(s)$ of the second PE is a constant, the problem here is the same as the workload-independent energy consumption model on non-DVS PEs in [13]. When power consumption $P_2(s)$ is linear to s , the problem here is the same as the workload-dependent energy consumption model on non-DVS PEs in [13], in which the energy consumption on the workload-dependent PE is proportional to the workload of task set $\hat{\mathbf{T}}$.

2.4 Problem Definition

This paper explores the Minimization of Energy consumption of periodic real-time tasks on platforms with Two Heterogeneous PEs (abbreviated as the METHE problem). The objective is to partition the input task set \mathbf{T} of n tasks into two disjoint subsets \mathbf{T}_1 and \mathbf{T}_2 such that the execution of task set \mathbf{T}_1 on the first PE and task set \mathbf{T}_2 on the second PE under the EDF scheduling policy will satisfy the timing constraints of tasks in \mathbf{T} , and the energy consumption in the hyper-period L of \mathbf{T} is minimized. Since the METHE problem is a super set of the \mathcal{NP} -hard problems in [13], it is, obviously, a \mathcal{NP} -hard problem.

This paper focuses on approximation algorithms with worst-case guarantees. A polynomial-time α -approximation algorithm for the studied problem must have a polynomial-time complexity of the input size and could derive a feasible schedule with the energy consumption at most α times of an optimal solution, for any input instance. This paper derives a fully polynomial-time approximation scheme (FPTAS), in which it is a $(1 + \epsilon)$ -approximation algorithm with polynomial-time complexity by treating $\frac{1}{\epsilon}$ as

an input parameter for any ϵ in specified ranges. Unless $\mathcal{NP} = \mathcal{P}$, fully polynomial-time approximation schemes are the best in terms of approximation algorithms with worst-case guarantees.

3 Our Algorithms

This section presents algorithms for the METHE problem. If the maximum available speeds $s_{\max,1}$ and $s_{\max,2}$ are both finite, deriving a feasible solution for the METHE problem is \mathcal{NP} -complete. This section will first show how to cope with systems in which there is no speed limitation on the first PE by proposing a polynomial-time greedy algorithm in Section 3.1 and an FPTAS in Section 3.2. The derivation of solutions in polynomial time by resource augmentation for $s_{\max,1} \neq \infty$ is presented by the end of this section.

3.1 A Greedy Algorithm

The greedy algorithm here are extended from the greedy algorithms in [13]. Consider two tasks τ_i and τ_j . If $\frac{c_{i,1}}{c_{i,2}}$ is greater than $\frac{c_{j,1}}{c_{j,2}}$, task τ_i is more suitable to execute on the second PE than τ_j is. The greedy algorithm here first sorts the input tasks so that $\frac{c_{i,1}}{c_{i,2}}$ is non-increasing for $i = 1, 2, \dots, n$.

We now show how to derive a task partition $(\mathbf{T}_1, \mathbf{T}_2)$ of \mathbf{T} . The initial solution is to assign all the tasks on the first PE by setting task set \mathbf{T}_1 as \mathbf{T} and task set \mathbf{T}_2 as an empty set. Let $E(\mathbf{T}_1, \mathbf{T}_2)$ be the optimal energy consumption in the hyper-period L of task set \mathbf{T} to execute task set \mathbf{T}_1 (task set \mathbf{T}_2 , respectively) on the first (second, respectively) PE without violating timing constraints. That is, $E(\mathbf{T}_1, \mathbf{T}_2)$ is $L \times (P_1^*(\sum_{\tau_j \in \mathbf{T}_1} \frac{c_{j,1}}{p_j}) + P_2^*(\sum_{\tau_j \in \mathbf{T}_2} \frac{c_{j,2}}{p_j}))$. Then, by considering tasks, says τ_i , from τ_1 to τ_n , we try to move task τ_i to execute on the second PE so that the energy consumption can be reduced. A naive greedy strategy is to move task τ_i from task set \mathbf{T}_1 to task set \mathbf{T}_2 if executing task set $\mathbf{T}_2 \cup \{\tau_i\}$ on the second PE is feasible and $E(\mathbf{T}_1, \mathbf{T}_2) > E(\mathbf{T}_1 \setminus \{\tau_i\}, \mathbf{T}_2 \cup \{\tau_i\})$.

However, since the energy consumption by moving x ($0 \leq x \leq 1$) portion of task τ_i from \mathbf{T}_1 to \mathbf{T}_2 might be decreasing when x is small but increasing when x is large, assigning τ_i to the second PE in such a case might sacrifice the possibility to assign another task τ_j to the second PE with more energy reduction. We should not move task τ_i to \mathbf{T}_2 if the first derivative (denoted by $E'(\mathbf{T}_1, \mathbf{T}_2, \tau_i, x)$) of function $P_1^*(x \frac{c_{i,1}}{p_i} + \sum_{\tau_j \in \mathbf{T}_1 \setminus \{\tau_i\}} \frac{c_{j,1}}{p_j}) + P_2^*((1-x) \frac{c_{i,2}}{p_i} + \sum_{\tau_j \in \mathbf{T}_2 \cup \{\tau_i\}} \frac{c_{j,2}}{p_j})$ on x is less than 0 when $x = 0$ and is greater than 0 when $x = 1$.

Algorithm 1 illustrates the pseudo-code of the greedy algorithm, denoted by Algorithm GREEDY. Task partition $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger)$ is derived in Step 4 and Step 5 in Algorithm 1 by adopting the naive greedy strategy to move τ_i from task set \mathbf{T}_1^\dagger to task set \mathbf{T}_2^\dagger if $E(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger) > E(\mathbf{T}_1^\dagger \setminus \{\tau_i\}, \mathbf{T}_2^\dagger \cup \{\tau_i\})$, while task partition $(\mathbf{T}_1^\sharp, \mathbf{T}_2^\sharp)$ is derived from Step 6 to Step 8 in Algorithm 1 by further considering the derivative of

Algorithm 1 : GREEDY

Input: $(\mathbf{T}, P_1^*, P_2^*, s_{\max,2})$;

- 1: order all the tasks in \mathbf{T} non-increasingly in $\frac{c_{i,1}}{c_{i,2}}$;
 - 2: $\mathbf{T}_1^\dagger \leftarrow \mathbf{T}, \mathbf{T}_1^\sharp \leftarrow \mathbf{T}, \mathbf{T}_2^\dagger \leftarrow \emptyset, \mathbf{T}_2^\sharp \leftarrow \emptyset$;
 - 3: **for** $i \leftarrow 1$ to n **do**
 - 4: **if** $E(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger) > E(\mathbf{T}_1^\dagger \setminus \{\tau_i\}, \mathbf{T}_2^\dagger \cup \{\tau_i\})$ and $\sum_{\tau_j \in \mathbf{T}_2^\dagger \cup \{\tau_i\}} \frac{c_{j,2}}{p_j} < s_{\max,2}$ **then**
 - 5: $\mathbf{T}_1^\dagger \leftarrow \mathbf{T}_1^\dagger \setminus \{\tau_i\}, \mathbf{T}_2^\dagger \leftarrow \mathbf{T}_2^\dagger \cup \{\tau_i\}$;
 - 6: **if** $E(\mathbf{T}_1^\sharp, \mathbf{T}_2^\sharp) > E(\mathbf{T}_1^\sharp \setminus \{\tau_i\}, \mathbf{T}_2^\sharp \cup \{\tau_i\})$ and $\sum_{\tau_j \in \mathbf{T}_2^\sharp \cup \{\tau_i\}} \frac{c_{j,2}}{p_j} < s_{\max,2}$ **then**
 - 7: **if** $E'(\mathbf{T}_1^\sharp, \mathbf{T}_2^\sharp, \tau_i, 0) \geq 0$ or $E'(\mathbf{T}_1^\sharp, \mathbf{T}_2^\sharp, \tau_i, 1) \leq 0$ **then**
 - 8: $\mathbf{T}_1^\sharp \leftarrow \mathbf{T}_1^\sharp \setminus \{\tau_i\}, \mathbf{T}_2^\sharp \leftarrow \mathbf{T}_2^\sharp \cup \{\tau_i\}$;
 - 9: let $(\mathbf{T}'_1, \mathbf{T}'_2)$ be the task partition with the minimum energy consumption under $|\mathbf{T}'_1| = 1$ or $|\mathbf{T}'_2| = 1$;
 - 10: return the best task partition among $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger), (\mathbf{T}_1^\sharp, \mathbf{T}_2^\sharp)$, and $(\mathbf{T}'_1, \mathbf{T}'_2)$ in terms of energy consumption;
-

energy consumption. Step 9 derives a task partition with the minimum energy consumption under the constraint that only one task is executed on the first or the second PE. Based on the above strategies, we can choose the best partition among these derived solutions. Since each step in the **for** loop in Algorithm 1 can be done in constant time by using proper data structures and Step 9 can be done in $O(n)$, the time complexity of Algorithm GREEDY is $O(n \log n)$, dominated by the sorting of tasks.

The S-GREEDY algorithm in [13] chooses the better solution between task partition $(\mathbf{T}_1^\sharp, \mathbf{T}_2^\sharp)$ and task partition $(\mathbf{T}'_1, \mathbf{T}'_2)$ when one PE is non-DVS. As a result, the performance of Algorithm GREEDY, in terms of energy consumption, is no worse than that of Algorithm S-GREEDY proposed in [13].

3.2 A General Framework

We now present a framework for the METHE problem, in which the approximation ratio is $(1 + \epsilon)$ for some user-specified parameter ϵ and the time complexity is polynomial in $\frac{1}{\epsilon}$ and the input size. We will first show a dynamic programming approach to derive optimal solutions for the METHE problem in exponential time, and, then, present the framework to trim states to achieve an FPTAS.

3.2.1 Optimal solutions by dynamic programming

For a task partition $(\mathbf{T}_1, \mathbf{T}_2)$ of task set \mathbf{T} , the *workload tuple* of the partition is defined as $(\sum_{\tau_i \in \mathbf{T}_1} \frac{c_{i,1}}{p_i}, \sum_{\tau_i \in \mathbf{T}_2} \frac{c_{i,2}}{p_i})$. For a workload tuple θ , let u_θ (w_θ , respectively) be its first (second, respectively) element. Suppose that tasks in \mathbf{T} are ordered arbitrarily, says $\tau_1, \tau_2, \dots, \tau_n$.

Let \mathcal{F}_i be the collection of all workload tuples of the first i tasks. Clearly, collection \mathcal{F}_1 is $\left\{ (0, \frac{c_{1,2}}{p_1}), (\frac{c_{1,1}}{p_1}, 0) \right\}$.

Moreover, collection \mathcal{F}_i for $i = 2, 3, \dots, n$ can be derived by applying the following recursive function:

$$\mathcal{F}_i = \left\{ (u_\theta, w_\theta + \frac{c_{i,2}}{p_i}), (u_\theta + \frac{c_{i,1}}{p_i}, w_\theta) \mid \theta \in \mathcal{F}_{i-1} \right\}. \quad (2)$$

The optimal solution of the METHE problem is to find the tuple θ^* in collection \mathcal{F}_n so that $P_1^*(u_{\theta^*}) + P_2^*(w_{\theta^*})$ is the minimum under the constraint that $u_{\theta^*} \leq s_{\max,1}$ and $w_{\theta^*} \leq s_{\max,2}$. If there is no tuple in collection \mathcal{F}_n with $u_{\theta^*} \leq s_{\max,1}$ and $w_{\theta^*} \leq s_{\max,2}$, there is no feasible solution for the input instance. According to the tuple, we can backtrack the collections of workload tuples to derive a task partition $(\mathbf{T}_1, \mathbf{T}_2)$ with $\sum_{\tau_i \in \mathbf{T}_1} \frac{c_{i,1}}{p_i}$ equal to u_{θ^*} and $\sum_{\tau_i \in \mathbf{T}_2} \frac{c_{i,2}}{p_i}$ equal to w_{θ^*} .

Since there are $O(2^i)$ tuples in collection \mathcal{F}_i , the time complexity to derive an optimal solution as above is $O(2^n)$, which is exponential in the input size. The time complexity can be improved by eliminating some tuples in collections \mathcal{F}_i s as follows:

- If there are two workload tuples θ^\dagger and θ^\sharp in collection \mathcal{F}_i with $u_{\theta^\dagger} \leq u_{\theta^\sharp}$ and $w_{\theta^\dagger} \leq w_{\theta^\sharp}$, then removing workload tuple θ^\sharp from \mathcal{F}_i does not affect the optimality of the derived solution.
- If there is a workload tuple θ^\sharp with $u_{\theta^\sharp} > s_{\max,1}$ or $w_{\theta^\sharp} > s_{\max,2}$, then removing workload tuple θ^\sharp from \mathcal{F}_i does not affect the optimality of the derived solution.

It is not difficult to see that the derived solution is optimal for the METHE problem. For brevity, here-after, let \mathcal{F}_i be the collection of workload tuples after applying the above procedures.

3.2.2 State-trimming for achieving FPTAS

The developed framework is to trim the state of the dynamic programming in Section 3.2.1 by grouping some workload tuples whose solutions are quite close. Suppose that δ is a positive real with $P_1^*((1 + \delta)s) \leq (1 + \epsilon)P_1^*(s)$ for any positive user-specified ϵ in a range. For example, if $P_1(s)$ is s^3 and ϵ is 0.1, then δ can be 0.032 since $1.032^3 < 1.1$. Or, δ can be $\frac{\epsilon}{7}$ for any $\epsilon \leq 7$, since $((1 + \delta)s)^\chi \leq (1 + 7\delta)s^\chi$ for any $\chi \leq 3$.

Let γ be $\frac{\ln(1+\delta)}{n-1}$, which will be used for grouping workload tuples. After calling the recursive procedure in Equation (2), we have to select representative tuples to form the new collection \mathcal{F}_i^\flat with state trimming. Algorithm 2, abbreviated as Algorithm TRIM, is the pseudo-code of the proposed framework.

Initially, \mathcal{F}_1^\flat is the same as \mathcal{F}_1 . For each iteration, we apply the following recursive function instead:

$$\mathcal{F}_i^\flat \leftarrow \left\{ (u_\theta, w_\theta + \frac{c_{i,2}}{p_i}), (u_\theta + \frac{c_{i,1}}{p_i}, w_\theta) \mid \theta \in \mathcal{F}_{i-1}^\flat \right\}. \quad (3)$$

Then, the workload tuples in \mathcal{F}_i^\flat are revised to eliminate some workload tuples as shown in Step 7 to Step 14 in Algorithm 2. Only representative workload tuples are left in

Algorithm 2 : TRIM**Input:** $(\mathbf{T}, P_1^*, P_2^*, s_{\max,1}, s_{\max,2}, \epsilon)$;

- 1: let δ be a positive real with $P_1^*((1+\delta)s) \leq (1+\epsilon)P_1^*(s)$ for any s ;
- 2: $\gamma \leftarrow \frac{\ln(1+\delta)}{n-1}$, where n is the number of tasks in \mathbf{T} ;
- 3: $\mathcal{F}_1^b \leftarrow \left\{ \left(0, \frac{c_{1,2}}{p_1}\right), \left(\frac{c_{1,1}}{p_1}, 0\right) \right\}$;
- 4: **for** $i \leftarrow 2$ to n **do**
- 5: $\mathcal{F}_i^b \leftarrow \left\{ \left(u_\theta, w_\theta + \frac{c_{i,2}}{p_i}\right), \left(u_\theta + \frac{c_{i,1}}{p_i}, w_\theta\right) \mid \theta \in \mathcal{F}_{i-1}^b \right\}$;
- 6: remove every workload tuple θ from \mathcal{F}_i^b if $u_\theta > s_{\max,1}$ or $w_\theta > s_{\max,2}$;
- 7: sort tuples in \mathcal{F}_i^b as $\theta_1, \theta_2, \dots, \theta_{|\mathcal{F}_i^b|}$ so that $u_{\theta_j} \leq u_{\theta_k}$ for any $j \leq k$;
- 8: $\mathcal{F}_i' \leftarrow \emptyset, \theta^b \leftarrow \theta_1$;
- 9: **for** $j \leftarrow 2$ to $|\mathcal{F}_i^b|$ **do**
- 10: **if** $u_{\theta^b} \leq u_{\theta_j} \leq (1+\gamma)u_{\theta^b}$ **then**
- 11: $w_{\theta^b} \leftarrow \min\{w_{\theta^b}, w_{\theta_j}\}$;
- 12: **else**
- 13: $\mathcal{F}_i' \leftarrow \mathcal{F}_i' \cup \{\theta^b\}, \theta^b \leftarrow \theta_j$;
- 14: $\mathcal{F}_i^b \leftarrow \mathcal{F}_i' \cup \{\theta^b\}$;
- 15: find the tuple $\theta^{*,b}$ in collection \mathcal{F}_i^b with the minimum $P_1^*(u_{\theta^{*,b}}) + P_2^*(w_{\theta^{*,b}})$;
- 16: backtrack the dynamic programming table and return the corresponding task partition of representative workload tuple $\theta^{*,b}$;

the resulting collection. We first sort workload tuples in \mathcal{F}_i^b as $\theta_1, \theta_2, \dots, \theta_{|\mathcal{F}_i^b|}$ so that $u_{\theta_j} \leq u_{\theta_k}$ for any $j \leq k$. Then, let temporary collection \mathcal{F}_i' be an empty set for the revision of collection \mathcal{F}_i^b . Workload tuple θ^b is the representative workload tuple of several workload tuples, in which θ^b is initialized as θ_1 . Then, we check workload tuples θ_j s from θ_2 to $\theta_{|\mathcal{F}_i^b|}$, i.e., Steps 9-13 in Algorithm 2, as follows: If the first element of workload tuple θ^b is no more than $(1+\gamma)u_{\theta_j}$, i.e., $u_{\theta_j} \leq (1+\gamma)u_{\theta^b}$, workload tuple θ^b can represent θ_j by revising w_{θ^b} as the minimum between w_{θ_j} and w_{θ^b} . Otherwise, we can insert the representative workload tuple θ^b into collection \mathcal{F}_i' , and workload tuple θ^b is updated to θ_j for representing for some other tuples later. Collection \mathcal{F}_i^b is then updated to the resulting collection of those representative workload tuples, i.e., Step 14 in Algorithm 2. After constructing collection \mathcal{F}_i^b , we can find the workload tuple $\theta^{*,b}$ in which $P_1^*(u_{\theta^{*,b}}) + P_2^*(w_{\theta^{*,b}})$ is the minimum. Backtracking can have a task partition of representative workload tuple $\theta^{*,b}$.

We now show the optimality and the complexity of Algorithm TRIM. Based on the steps from Step 7 to Step 14, we can have the following lemmas for the workload tuples in collection \mathcal{F}_i^b .

Lemma 1 For each workload tuple $\theta^\#$ in collection \mathcal{F}_i , there exists a workload tuple θ^b in collection \mathcal{F}_i^b with $u_{\theta^b} \leq u_{\theta^\#}$ and $w_{\theta^b} \leq w_{\theta^\#}$.

Proof. This lemma can be proved by induction. It is clear that the statement holds when $i = 1$ because \mathcal{F}_1^b is the same as \mathcal{F}_1 . Suppose that the statement holds when $i = \ell$, i.e., for

each workload tuple $\theta^\#$ in \mathcal{F}_ℓ , there exists a workload tuple θ^b in collection \mathcal{F}_ℓ^b with $u_{\theta^b} \leq u_{\theta^\#}$ and $w_{\theta^b} \leq w_{\theta^\#}$.

By the definition of $\mathcal{F}_{\ell+1}$, each workload tuple $\theta^\#$ in \mathcal{F}_ℓ spans at most two workload tuples $(u_{\theta^\#}, w_{\theta^\#} + \frac{c_{\ell+1,2}}{p_{\ell+1}})$ and $(u_{\theta^\#} + \frac{c_{\ell+1,1}}{p_{\ell+1}}, w_{\theta^\#})$. Moreover, each workload tuple θ^b in \mathcal{F}_ℓ^b spans at most two workload tuples $(u_{\theta^b}, w_{\theta^b} + \frac{c_{\ell+1,2}}{p_{\ell+1}})$ and $(u_{\theta^b} + \frac{c_{\ell+1,1}}{p_{\ell+1}}, w_{\theta^b})$ in Step 5 in Algorithm 2, while the values in the workload tuples might be reduced in Steps 9-14 in Algorithm 2. As a result, for each workload tuple $\theta^\#$ in collection $\mathcal{F}_{\ell+1}$, there exists a workload tuple θ^b in collection $\mathcal{F}_{\ell+1}^b$ with $u_{\theta^b} \leq u_{\theta^\#}$ and $w_{\theta^b} \leq w_{\theta^\#}$. \square

Lemma 2 For each workload tuple θ^b in collection \mathcal{F}_i^b , backtracking the dynamic programming table can derive a task partition $(\mathbf{T}_1, \mathbf{T}_2)$ of task set $\{\tau_j \mid j = 1, 2, \dots, i\}$ of the first i tasks such that $\sum_{\tau_j \in \mathbf{T}_1} \frac{c_{j,1}}{p_j} \leq (1+\gamma)^{i-1}u_{\theta^b}$ and $\sum_{\tau_j \in \mathbf{T}_2} \frac{c_{j,2}}{p_j} \leq w_{\theta^b}$.

Proof. This lemma can be proved by induction. It is clear that the statement holds when $i = 1$. Assume that the statement holds when $i = \ell$. For a workload tuple θ' in $\mathcal{F}_{\ell+1}^b$, there must be a tuple θ^b in \mathcal{F}_ℓ^b with $u_{\theta'} \leq u_{\theta^b} \leq (1+\gamma)u_{\theta'}$ or $u_{\theta'} \leq u_{\theta^b} + \frac{c_{\ell+1,1}}{p_{\ell+1}} \leq (1+\gamma)u_{\theta'}$. For the case that $u_{\theta'} \leq u_{\theta^b} \leq (1+\gamma)u_{\theta'}$, backtracking the dynamic programming table can derive a task partition $(\mathbf{T}_1, \mathbf{T}_2)$ of task set $\{\tau_j \mid j = 1, 2, \dots, \ell+1\}$ of the first $\ell+1$ tasks such that $\sum_{\tau_j \in \mathbf{T}_1} \frac{c_{j,1}}{p_j} \leq (1+\gamma)^\ell u_{\theta'}$. It is similar for the other case. Hence, by the induction hypothesis, the lemma is proved. \square

The following lemma shows that Algorithm TRIM is a $(1+\epsilon)$ -approximation algorithm.

Lemma 3 The energy consumption of the solution derived from Algorithm TRIM is at most $(1+\epsilon)$ times of the optimal energy consumption of the METHE problem.

Proof. Let $(\mathbf{T}_1^b, \mathbf{T}_2^b)$ be the task partition derived from Algorithm TRIM. Based on Lemma 2, we know that $\sum_{\tau_i \in \mathbf{T}_1^b} \frac{c_{i,1}}{p_i} \leq (1+\gamma)^{n-1}u_{\theta^{*,b}}$ and $\sum_{\tau_i \in \mathbf{T}_2^b} \frac{c_{i,2}}{p_i} \leq w_{\theta^{*,b}}$, where $\theta^{*,b}$ is the workload tuple θ in collection \mathcal{F}_n^b with the minimum $P_1^*(u_\theta) + P_2^*(w_\theta)$. Because

$$(1+\gamma)^{n-1} = \left(1 + \frac{\ln(1+\delta)}{n-1}\right)^{n-1} \leq e^{\ln(1+\delta)} \leq 1+\delta,$$

we know that $\sum_{\tau_i \in \mathbf{T}_1^b} \frac{c_{i,1}}{p_i} \leq (1+\delta)u_{\theta^{*,b}}$.

Let workload tuple θ^* be the workload of the optimal task partition for the METHE problem. By Lemma 1, there exists a workload tuple θ^b in collection \mathcal{F}_n^b with $u_{\theta^b} \leq u_{\theta^*}$ and $w_{\theta^b} \leq w_{\theta^*}$. By the definition of workload tuple $\theta^{*,b}$, we know that $P_1^*(u_{\theta^{*,b}}) + P_2^*(w_{\theta^{*,b}}) \leq P_1^*(u_{\theta^b}) + P_2^*(w_{\theta^b}) \leq P_1^*(u_{\theta^*}) + P_2^*(w_{\theta^*})$. Hence, we have $P_1^*(\sum_{\tau_i \in \mathbf{T}_1^b} \frac{c_{i,1}}{p_i}) + P_2^*(\sum_{\tau_i \in \mathbf{T}_2^b} \frac{c_{i,2}}{p_i}) \leq P_1^*((1+\delta)u_{\theta^{*,b}}) + P_2^*(w_{\theta^{*,b}}) \leq (1+\epsilon)(P_1^*(u_{\theta^*}) + P_2^*(w_{\theta^*}))$. \square

The following lemma shows the size of collection \mathcal{F}_i is polynomial in the input size and $\frac{1}{\delta}$.

Lemma 4 There are at most $O(\frac{n \log \lambda}{\delta})$ tuples in each collection \mathcal{F}_i^b for $i = 1, 2, \dots, n$, where λ is
$$\frac{\min\{s_{\max,1}, \sum_{\tau_i \in \mathbf{T}} \frac{c_{i,1}}{p_i}\}}{\min_{i=1,2,\dots,n} \frac{c_{i,1}}{p_i}}.$$

Proof. Suppose that m is the number of workload tuples in \mathcal{F}_i^b after Step 14 in Algorithm 2 is done. By the elimination procedure between Step 7 and Step 14 in Algorithm 2, we know that $(1 + \gamma)^{m-1} u_{\theta_1^b} < (1 + \gamma)^{m-2} u_{\theta_2^b} < \dots < u_{\theta_m^b}$ since any two workload tuples θ_j^b and θ_k^b with $\theta_j^b < \theta_k^b$ must satisfy $(1 + \gamma) u_{\theta_j^b} < u_{\theta_k^b}$. Because the first elements of workload tuples must be 0 or in the range of $\min_{i=1,2,\dots,n} \frac{c_{i,1}}{p_i}$ and $\min\{s_{\max,1}, \sum_{\tau_i \in T} \frac{c_{i,1}}{p_i}\}$, we know that either $(1 + \gamma)^{m-1} < \lambda$ or $(1 + \gamma)^{m-2} < \lambda$. As a result,

$$\begin{aligned} m &< 2 + \frac{\ln \lambda}{\ln(1+\gamma)} \leq 2 + \frac{\ln \lambda}{1+\gamma} \\ &= 2 + \ln \lambda \left(1 + \frac{n-1}{\ln 1+\delta}\right) = O\left(\frac{n \log \lambda}{\delta}\right), \end{aligned}$$

where the second inequality comes from $\ln x \geq \frac{x-1}{x}$ when $x > 1$. \square

With the above lemmas, we can now show that Algorithm TRIM is an FPTAS of the METHE problem.

Theorem 1 *Algorithm TRIM is a fully polynomial-time approximation scheme of the METHE problem.*

Proof. Algorithm TRIM is a $(1 + \epsilon)$ -approximation algorithm due to Lemma 3. Since sorting in Step 7 in Algorithm 2 dominates the time complexity of the iteration in the loop, constructing collection \mathcal{F}_i^b can be done in $O(m \log m)$, where $m = \frac{n \log \lambda}{\delta}$. Clearly, the time complexity of Algorithm TRIM is $O(nm \log m)$, which is polynomial in $\frac{1}{\delta}$ and the input size since $\log \lambda$ is polynomial in the number of bits required to encode the input. By the definition of power consumption functions, we know that $\frac{1}{\delta}$ is $O(\frac{1}{\epsilon})$, since δ in Algorithm TRIM can be set as $\frac{\epsilon}{7}$ for any $\epsilon \leq 7$. \square

Remarks Till now, we only cope with input instances in which $c_{i,1} \neq \infty$ for every task τ_i . For those input instances with some τ_i s with $c_{i,1} = \infty$, we just have to assign them on the second PE by revising Step 2 to construct \mathcal{F}_1^b in Algorithm 2 and Step 2 for the initialization of task partition in Algorithm 1.

3.3 Resource Augmentation

Deriving a feasible task partition for the METHE problem is \mathcal{NP} -complete if $s_{\max,1} \neq \infty$ and $s_{\max,2} \neq \infty$. Task rejection [9] or resource augmentation with constraint violation [6, 17] to violate the constraint on the maximum speed for a little might be needed. Algorithm GREEDY does not provide any guarantee in speed violation, but Algorithm TRIM does. For an input instance, Algorithm TRIM can derive a solution, in which those tasks executed on the second PE can meet their timing constraints, but those on the first PE might not. However, according to the first paragraph in the proof of Lemma 3, executing those tasks on the first PE at speed $(1 + \delta) s_{\max,1}$ can complete them in time; otherwise, there is no feasible solution for the input instance. Hence, Algorithm TRIM is a $(1 + \epsilon)$ -approximation algorithm with $(1 + \delta)$ -resource augmentation on the maximum speed.

Processor Model	Min $\kappa(\text{mW}/(\text{MHz})^3)$	Max $\kappa(\text{mW}/(\text{MHz})^3)$
ARM92x	1.5026×10^{-5}	3.1855×10^{-5}
ARM10x	3.0469×10^{-6}	3.4466×10^{-6}
ARM11x	4.0718×10^{-7}	1.1478×10^{-6}
TMS320Cx	3.2277×10^{-9}	5.2083×10^{-7}
TMS320Dx	1.1250×10^{-8}	3.5095×10^{-8}
Intel XScale	1.52×10^{-6}	1.52×10^{-6}

Table 1. Power consumption parameters of DVS processors.

4 Experiments

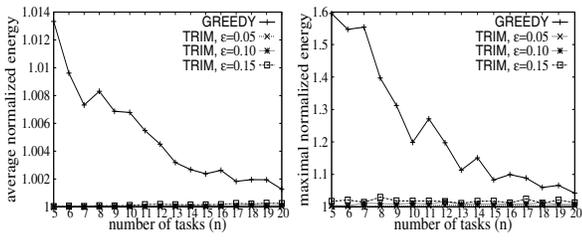
This section provides performance evaluation of the proposed framework, while Algorithm GREEDY is adopted for comparison. Note that the performance of Algorithm GREEDY, in terms of energy consumption, is no worse than that of Algorithm S-GREEDY proposed in [13] when one of the PEs is non-DVS PE.

Experimental Setup The user parameter ϵ in Algorithm TRIM is set as 0.05, 0.10, and 0.15 for evaluation. Two different hardware configurations are considered in the experiments. The first configuration uses two heterogeneous DVS processors for task execution, while the second one uses a DVS processor and an FPGA. For DVS processors, the general-purpose embedded processors, such as ARM92x, ARM10x, ARM11x, TMS320Cx, TMS320Dx, and Intel XScale, are adopted. The power consumption function of the evaluated processors are $P(s) = \kappa s^3$. The parameters κ s of the evaluated processors are summarized in Table 1 [12, 13]. For non-DVS PE, the Xilinx FPGA Virtex-4 with 1-D model is selected, in which the power consumption of the Xilinx FPGA for the XC4VLX100 part with package FF1513 is 588mW [23].

For each task τ_i , period p_i is a random number in $(0, 1]$. For a specified total workload U_1^* and U_2^* of a set of n tasks, each task τ_i has two weights $\mu_{i,1}$ and $\mu_{i,2}$ to determine the amount of CPU cycles of tasks on the PEs. The value of $\mu_{i,1}$ is a random variable in $(0, 1]$. Similar to the experimental setup in [13], we explore different types of distribution of $\mu_{i,2}$. In the *independent distribution model*, $\mu_{i,2}$ is a random variable in $(0, 1]$; in the *proportional distribution model*, $\mu_{i,2}$ is a random variable in $(0, 1 + \mu_{i,1}]$. Then, the execution cycles $c_{i,1}$ on the first DVS PE of task τ_i is set as $\frac{\mu_{i,1}}{\sum_{j=1}^n \mu_{j,1}} U_1^* p_i$, and $c_{i,2}$ is $\frac{\mu_{i,2}}{\sum_{j=1}^n \mu_{j,2}} U_2^* p_i$. For configurations with an FPGA, $\frac{c_{i,2}}{p_i}$ is interpreted as the occupied size of the FPGA.

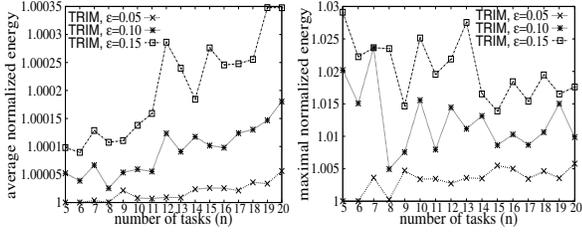
The *normalized energy* for an algorithm of an input instance is the energy consumption of the derived solution divided by the optimal solution, by performing an exhaustive search, of the input instance. We perform 1024 independent tests for each configuration and report the average and maximal normalized energy.

Experimental Results Figure 1 shows the experimental results under the independent distribution model when both



(a) average normalized energy (b) maximal normalized energy

Figure 1. Average and maximal normalized energy under the independent distribution model when both PEs are DVS processors.



(a) average normalized energy (b) maximal normalized energy

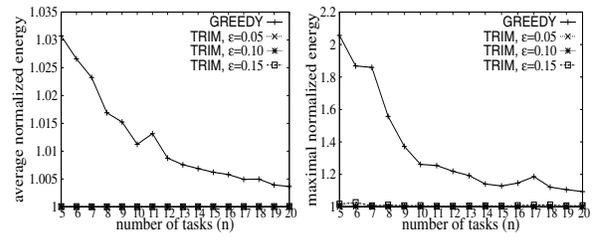
Figure 2. Average and maximal normalized energy under the proportional distribution model when both PEs are DVS processors.

PEs are DVS processors. When the number of tasks increases, the performance, in terms of energy consumption, of Algorithm GREEDY becomes better in both the average case in Figure 1(a) and the worst case in Figure 1(b) in the experiments. This is because the more tasks in the system with similar workload distribution, the less error an improper assignment would make. The performance of Algorithm TRIM is very close to optimal solutions in average cases.

To tell the difference of performance of Algorithm TRIM under different settings of ϵ , Figure 2 shows the experimental results of Algorithm TRIM under the proportional distribution model when both PEs are DVS processors, while the results for Algorithm GREEDY are similar to those in Figure 1 and are omitted for clarity. The worst-case normalized energy in the experiments is about 1.03 when ϵ is set as 0.15, which is a 1.15-approximation algorithm. The smaller the value of ϵ is, the better the solution derived by Algorithm TRIM in both average and worst cases.

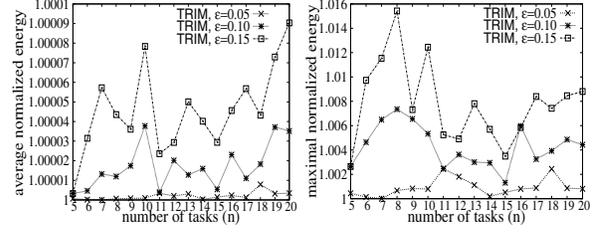
Figure 3 (Figure 4, respectively) shows the average normalized energy and the maximal normalized energy under the independent (proportional, respectively) distribution model when one PE is with the DVS capability and the other is an FPGA with speed $s_{\max,2}$ set as $\frac{U_2^*}{2}$. Compared to those results for systems with two DVS PEs, Algorithm GREEDY performs worse, but Algorithm TRIM performs better.

Figure 5(a) illustrates the average running time of the algorithms running on a system with a Pentium4 3GHz CPU for both distribution models when both PEs are with the DVS capability, while Figure 5(b) is for systems with one



(a) average normalized energy (b) maximal normalized energy

Figure 3. Average and maximal normalized energy under the independent distribution model when one PE is with the DVS capability and the other is an FPGA.



(a) average normalized energy (b) maximal normalized energy

Figure 4. Average and maximal normalized energy under the proportional distribution model when one PE is with the DVS capability and the other is an FPGA.

DVS PE and one FPGA. Deriving an optimal solution when n is large takes from minutes to hours, and their results are omitted for clarity. Algorithm GREEDY is very efficient to return a solution in 0.03 milliseconds. As for Algorithm TRIM, the running time is related to the value of ϵ . Clearly, when ϵ is smaller, the running time is longer with more precise solution. Hence, our framework, i.e., Algorithm TRIM, provides system engineers a means to trade the quality of the derived schedule for the running time.

5 Conclusion

This paper explores energy-efficient task partition of periodic real-time tasks in a system with two heterogeneous processing elements. The power consumption functions of the processing elements are independent, so are the characteristics to execute tasks. A general framework is proposed to provide a fully polynomial-time approximation scheme (FPTAS) to derive a solution with energy consumption close to the optimal energy consumption in tolerable time/space complexity. This paper answers the open problem in [13] for deriving energy-efficient task partitions with worst-case guarantees in energy consumption minimization on two heterogeneous processing elements, in which only approximation algorithms for the maximization of energy saving (compared to the solution by executing all the tasks on the DVS PE) is known [13]. Experiments are performed to evaluate the performance of the proposed framework in terms of energy consumption minimization with deadline satisfaction. The results reveal the effectiveness in the min-

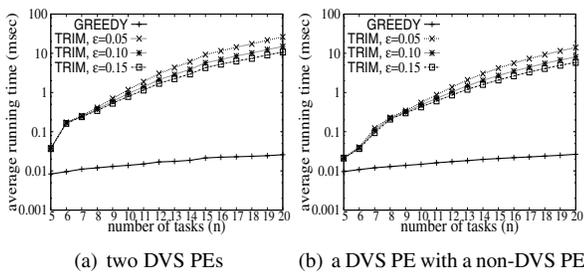


Figure 5. The average running time of the algorithms.

imization of energy consumption of the proposed framework.

A straightforward extension of the proposed FPTAS to more than two heterogeneous multiprocessor systems is possible, but the approximation ratio is not guaranteed any more. For future research, we would like to explore approximation algorithms for systems with more than two heterogeneous processing elements.

References

- [1] 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.
- [2] 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.
- [3] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 95–105, 2001.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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*, pages 255–260, 2006.
- [8] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, pages 289–294, 2007.
- [9] J.-J. Chen, T.-W. Kuo, C.-L. Yang, and K.-J. King. Energy-efficient real-time task scheduling with task rejection. In *DATE*, pages 1629–1634, 2007.
- [10] F. Gruian and K. Kuchcinski. Lenex: Task scheduling for low energy systems using variable supply voltage processors. In *Proceedings of Asia South Pacific Design Automation Conference*, pages 449–455, 2001.
- [11] 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.
- [12] T.-Y. Huang, Y.-C. Tsai, and E. T.-H. Chu. A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem. In *21th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2007.
- [13] 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)*, pages 303–312, 2006.
- [14] 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.
- [15] 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.
- [16] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the 40th Design Automation Conference*, pages 125–130, 2003.
- [17] 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.
- [18] J. Luo and N. Jha. Static and dynamic variable voltage scheduling algorithms for realtime heterogeneous distributed embedded systems. In *the 15th International Conference on VLSI Design (VLSID'02)*, pages 719–726, 2002.
- [19] 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.
- [20] 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.
- [21] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2002.
- [22] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*. IEEE, 2002.
- [23] XILINX. <http://www.xilinx.com/>.
- [24] 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.
- [25] 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.
- [26] Y. Yu and V. K. Prasanna. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In *Proceedings of the Ninth International Conference on Parallel and Distributed Systems(ICPADS'02)*. IEEE, 2002.
- [27] 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.