Minimizing Energy Consumption for Frame-Based Tasks on Heterogeneous Multiprocessor Platforms

Dawei Li and Jie Wu, *Fellow, IEEE*

Abstract—Heterogeneous multiprocessors have been widely used in modern computational systems to increase the computing capability. As the performance increases, the energy consumption in these systems also increases significantly. Dynamic Voltage and Frequency Scaling (DVFS) is considered an efficient scheme to achieve the goal of saving energy, because it allows processors to dynamically adjust their supply voltages and/or execution frequencies to work on different power/energy levels. In this paper, we consider scheduling non-preemptive frame-based tasks on DVFS-enabled heterogeneous multiprocessor platforms with the goal of achieving minimal overall energy consumption. We consider three types of heterogeneous platforms, namely, dependent platforms without runtime adjusting, dependent platforms with runtime adjusting, and independent platforms. For these three platforms, we first formulate the problems as binary integer programming problems, and then, relax them as convex optimization problems, which can be solved by the well-known interior point method. We propose a Relaxation-based Iterative Rounding Algorithm (RIRA), which tries to achieve the task set partition, that is closest to the optimal solution of the relaxed problems, in every step of a task-to-processor assignment. Experiments and comparisons show that our RIRA produces a better performance than existing methods and a simple but naive method, and achieves near-optimal scheduling under most cases. We also provide comprehensive complexity, accuracy and scalability analysis for the RIRA approach by investigating the interior-point method and by running specially designed experiments. Experimental results also show that the proposed RIRA approach is an efficient and practically applicable scheme with reasonable complexity.

Index Terms—Dynamic voltage and frequency scaling (DVFS), heterogeneous multiprocessor platforms, iteration-based task partitioning, energy-aware scheduling

1 Introduction

High energy consumption in modern computational systems has been a critical problem. Increased energy consumption influences the society from various aspects. As has been reported, desktop computers in the United States account for over 10 percent of commercial electricity consumption; a large data center can consume as much electricity as a city. High energy consumption in modern computational systems also increases the global carbon dioxide emissions. Besides, it also increases the requirements for packaging and cooling technologies, and demonstrates the need for more sophisticated fault-tolerant mechanisms [1].

However, the need of high computational performance never stops. To meet the increasing performance requirements, modern computational systems adopt multiprocessor platforms. As the computational performance increases, energy consumption in these systems also increases significantly. Dynamic Voltage and Frequency Scaling (DVFS)[2], which allows processors to dynamically adjust the supply voltage or the clock frequency to operate on different power/energy levels, is considered an effective way to achieve the goal of saving energy.

Energy-aware scheduling on uniprocessors has received tremendous research endeavors [3], [4], [5], [6], [7], [8]. In this paper, we address scheduling on multiprocessors. A multiprocessor platform is considered *homogeneous* if all of the processors on the platform are identical; if all of the processors are not identical, it is considered *heterogeneous*. Task scheduling approaches on multiprocessor platforms can be classified into two categories, namely, *partition-based scheduling* and *global scheduling*. In partition-based scheduling, each task is assigned statically to one processor. Partition-based scheduling allows schedulability to be verified by mature uniprocessor analysis techniques. In global scheduling, there is a single job queue from which jobs are dispatched to any available processor according to a global priority scheme.

1.1 Related Work

Energy-aware scheduling on both homogeneous and heterogeneous multiprocessor platforms also attracted significant research interests.

For energy-aware scheduling on homogeneous multiprocessors, extensive research has been conducted, which falls into both categories of partition-based scheduling and global scheduling. Representative works on partition-based scheduling include [9], [10], [11], [12], [13], [14]. Yang et al.

• *The authors are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: {dawei.li, jiewu}@temple.edu.*

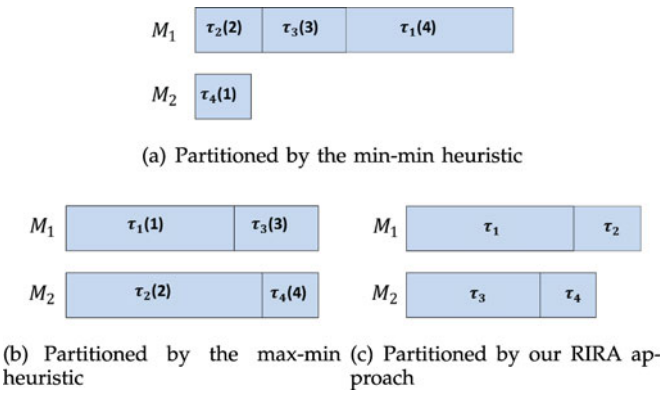
Manuscript received 20 Dec. 2013; revised 10 Mar. 2014; accepted 18 Mar. 2014. Date of publication 23 Mar. 2014; date of current version 6 Feb. 2015. Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2313338

[11] addresses scheduling on dependent platforms. The Largest Task First (LTF) strategy is applied to conduct task partitioning. After this, the optimal frequency scheduling for different time intervals is also derived. Chen et al. [12] consider scheduling on independent platforms with the consideration of task migration. Chen and Kuo [13] study scheduling with the consideration of application-specific power consumption on independent platforms. Kong et al. [14] handle scheduling on partitioned multi-core platforms, where cores within the same partition are dependent and cores from different partitions are independent. Research on global scheduling also exists [15], [16], [17], though comparatively less than partition-based scheduling.

Energy-aware scheduling on heterogeneous systems also receives extensive research endeavors [18], [19], [20], [21], [22], [23], [24], [25], [26]. Since global scheduling generally only queues jobs first and then assigns jobs to available processors, without considering the heterogeneity of processors, energy-aware scheduling on heterogeneous processors are mainly partition-based [20], [21], [22], [23], [24], [25], [26]. In [22], the authors address the problem of mapping a set of frame-based tasks to heterogeneous multiprocessors. Several heuristics are described and analyzed in detail. One typical heuristic is the min-min heuristic. Yang et al. [23] study platforms with a fixed number of heterogeneous processors. Chen and Thiele [24] investigate platforms with a fixed number of heterogeneous processor types, while one processor type may still have multiple processors. Hung et al. [25] consider energy-aware scheduling on a heterogeneous platform with one non-DVFS Processing Unit (PU) and one DVFS processor. Awan and Petters [20] address the partitioning problem where DVFS is not enabled. Lee and Zomaya [26] consider scheduling precedence constrained tasks/applications.

In this paper, we consider partition-based energy-aware scheduling for frame-based tasks on heterogeneous DVFS-enabled multiprocessor platforms. The main difference between these above-mentioned works and ours is, that our proposed method has a strong theoretical foundation to produce energy-efficient scheduling, as we will show later. For the same problem, widely used methods derive an "energy-efficient" partition that tries to achieve balanced workloads among all processors. It is believed that a balanced partition also demonstrates a good performance in terms of overall energy consumption. For example, in [22], the min-min heuristic is considered an energy-aware method for mapping tasks on heterogeneous platforms; in [27], it is pointed out that, in some situations, the max-min heuristic can achieve better load balancing than the min-min heuristic. However, we show that workload-balanced partitioning methods are not optimal in terms of overall energy consumption. We describe the two heuristics here, since we will compare our proposed RIRA approach with them throughout the paper.

Min-min: in [22], the min-min heuristic is applied to map frame-based tasks to heterogeneous multiprocessors with the goal of saving energy. It begins with the set of all unassigned tasks, which is initialized as the original task set. The heuristics consists of two phases. In the first phase, the set of tasks' minimum expected completion times is calculated (for all unassigned tasks). In the second phase, the task with the overall minimum expected completion time among all unassigned tasks is chosen and assigned to the corresponding processor. Then, this task is removed from the unassigned task set, and the procedure is repeated until all tasks are assigned.



(a) Partitioned by the min-min heuristic

(b) Partitioned by the max-min heuristic

(c) Partitioned by our RIRA approach

Fig. 1. Task set partition using different methods.

Max-min [27], [28], [29]: this heuristic is very similar to the min-min heuristic. It also begins with the set of all unassigned tasks. The only difference is that, in the second phase, the task with the overall maximum expected completion time among all of the unassigned tasks is chosen and assigned to the corresponding processor.

Our work in this paper is based on our previous work in [30]. Different from our previous work, we provide comprehensive analysis on the complexity, accuracy and scalability of the proposed RIRA; corresponding theoretical analysis and experimental results are presented to justify the strength of our approach. Also, we apply our proposed algorithm to a representative practical processor's power configuration; comparisons with other method also demonstrate the advantage of our proposed RIRA approach.

1.2 Motivational Example

We provide an example to show that workload-balanced partitioning methods do not always work well on heterogeneous platforms in terms of reducing overall energy consumption. Due to heterogeneities of tasks and processors, different processors may have different execution efficiencies for different tasks. Denote, when executing at a same fixed frequency f_s, the execution time of task τ_i on the jth processor M_j by $t_{i,j}$. Consider a simple example consisting of four frame-based tasks and two processors, where $t_{1,1} = 30$, $t_{1,2} = 50$, $t_{2,1} = 12$, $t_{2,2} = 35$, $t_{3,1} = 15$, $t_{3,2} = 24$, $t_{4,1} = 12$, and $t_{4,2} = 10$. Assume that the tasks' shared deadline is 100. The min-min heuristic and max-min heuristic produce partitions as shown in Figs. 1a and 1b, respectively. A better partition, as shown in Fig. 1c, can be achieved by our proposed RIRA. We do not delve into the details of RIRA now, and just give the result to show that RIRA outperforms the workload-balanced partitioning methods.

After partitioning, under a given assumption about the platform, processor frequencies are adjusted correspondingly, to achieve the goal of saving energy. Assume that the power consumption of each processor running at frequency f is $p = f^3$; thus, the energy consumption of the processor during a time interval t, is $e = f^3 t$.

TABLE 1
The Overall Energy Consumption of Different Partitions on Different Platforms

Platform type	Overall energy consumption		
	min-min	max-min	Our Proposed RIRA
Type I	$21.7683f_s^3$	$18.225f_s^3$	$13.4064f_s^3$
Type II	$21.17f_s^3$	$18.225f_s^3$	$13.139f_s^3$
Type III	$18.6193f_s^3$	$18.225f_s^3$	$11.3392f_s^3$

We first calculate the energy consumptions of the final schedulings based on the first partition (Fig. 1a) on different platforms. Readers might want to refer to Section 2.2 for clear definitions of the three platform types. On a dependent platform without runtime adjusting, to achieve minimal energy consumption, both of the two processors should operate at $0.57f_s$; the overall energy consumption on the two processors is $21.7683f_s^3$. On a dependent platform with runtime adjusting, applying the method in [11], we can set the shared optimal frequency before and after finishing task τ_4 for the two processors, respectively; at these optimal frequency settings, the minimal overall energy consumption can be calculated as $21.17f_s^3$. On an independent platform, to achieve minimal energy consumption, both processors should reduce their processing frequency such that their workloads finish exactly at the deadline 100; thus, processor M_1executes at $0.57f_s$, and processor M_2 executes at $0.1f_s$; the overall energy consumption can be calculated as $18.6193f_s^3$.

By similar calculations, the energy consumptions of the final schedulings based on the two other partitions (Figs. 1b and 1c) on the three types of platforms can also be achieved, and are listed in Table 1, in which "Type I" represents dependent platforms without runtime adjusting, "Type II" represents dependent platforms with runtime adjusting, and "Type III" represents independent platforms. As can be seen, our approach achieves the best performance in terms of overall energy consumption on these three types of platforms. Note that, for this special example, our approach produces the same partition for both dependent and independent platforms; generally speaking, our RIRA will produce different optimal partitions (in terms of reducing overall energy consumption) for dependent platforms and independent platforms, respectively.

1.3 Main Contributions

In this paper, we propose a Relaxation-based Iterative Rounding Algorithm (RIRA) for energy-aware task partitioning on heterogeneous multiprocessor platforms. Our main contributions are as follows:

- First, we assume that different processors have different execution efficiencies for different tasks. On the one hand, due to the heterogeneity of the platform, different processors may have different hardware implementations and different instruction set architectures, etc. On the other hand, different tasks/applications may have variously different characteristics. Thus, this general assumption is practical on real platforms.
- Second, most previous works derive partitions according to existing work that tries to achieve a workload-balanced partition. However, we show that a "workload-balanced" partition is not optimal in terms of reducing overall energy consumption. Since the execution efficiencies vary from processor to processor, it may be better to assign a heavier workload to a more efficient processor and a lighter workload to a less efficient processor. Thus, in our consideration, we always place the energy consumption at the highest priority.
- Third, we propose a novel Relaxation-based Iterative Rounding Algorithm for partitioning a task set on heterogeneous multiprocessor platforms. The main idea of our approach is to relax the original binary integer programming problem; then, assign the most "influential" task to a processor according to the relaxed optimal solution in the sense that the assignment is closest to the optimal solution. After having assigned some task(s), we update the relaxed optimization problem, and assign the next most "influential" task, based on the solution for the updated optimization problem. Experiments and comparisons verify that our RIRA produces a better performance than existing methods, and achieves near-optimal scheduling under most cases.
- Finally, we conduct comprehensive analysis on the complexity, accuracy and scalability of the proposed algorithm RIRA. Corresponding theoretical analysis and numerical simulations are presented to justify the strength and applicability of our overall approach. We believe this "iterative rounding" technique also has its merits when we come to other similar integer, especially binary integer, programming problems.

1.4 Paper Organization

The organization of this paper is as follows. Section 2 gives the system model and problem definition; the main idea of our approach is briefly introduced. In Sections 3 and 4, our proposed approach is applied to schedule frame-based tasks on dependent platforms; solutions for dependent platforms without and with runtime adjusting are provided in Sections 3 and 4, respectively. Section 5 applies our approach to independent platforms. Simulations and experiments are provided in Section 6. A brief conclusion is made in Section 7.

2 System Model and Problem Definition

2.1 Task Model

In this paper, we consider scheduling a set of independent frame-based tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ that are released at the same time 0 and share a common deadline D. This task model is a typical one which reflects various practical applications. Here, tasks τ_i's are assumed to have no precedence constraints, and tasks are non-preemptive. Each task τ_i's execution requirement is quantified by its Worst Case Execution Cycles (WCECs), denoted by C_i. The Worst Case Execution Time (WCET) of task τ_i, when it is executed at frequency f on a unit-efficiency processor, can be calculated as C_i/f. Correspondingly, the WCET of task τ_i, when

it is executed at frequency f on a processor with execution efficiency λ, ($\lambda \in (0,1]$), can be calculated as $C_i/(\lambda f)$.

2.2 Platform Model

We consider platforms with m heterogeneous processors. All processors are DVFS-enabled processors that can adjust their supply voltages and execution frequencies. We define $\lambda_{i,j} \in (0,1]$ as the execution efficiency of processor M_j when it is executing task τ_i. These kinds of platform are widely adopted in various practical systems, ranging from multiprocessor mobile phones, multiprocessor workstations, or even distributed systems. The WCET of task τ_i, when it is executed at frequency f on processor M_j, can be calculated as $C_i/(\lambda_{i,j} f)$. We assume ideal processors whose frequency ranges are continuous on $(0, +\infty)$. Processors can operate in two modes: one is *run* mode, where the power consumption only consists of dynamic power $p(f) = f^3$; the other one is *idle* mode, where the processor consumes zero power. Additionally, we assume that when a processor has no task to execute, it transitions into idle mode immediately. The transition time and energy overhead is considered very small compared to that required to complete a task, and is assumed tobe incorporated into the execution time and energy of the task. The power consumption model that we consider in this paper is widely adopted by existing works [31], [32].

Under these assumptions, we further consider three types of platforms. If all of the processors must operate at a common frequency, and the shared execution frequency cannot be adjusted during runtime after setting the initial frequency, the platform is called a *dependent platform without runtime adjusting*. If all the processors must operate at a common frequency, and the shared frequency can be adjusted during runtime after setting the initial frequency, the platform is called a *dependent platform with runtime adjusting*. If processors can operate at different frequencies at any time and can adjust their execution frequencies independently, the platform is considered an *independent platform*. The dependent platform and independent platform consist with the concepts of full-chip DVFS and per-core DVFS, respectively [21], [33].

2.3 Problem Definition

Given a set of frame-based tasks, $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, our goal is to schedule all of the tasks on m heterogeneous processors, $M_1, M_2, \ldots, M_m$, such that the overall energy consumption on all the processors is minimized. Scheduling consists of two steps. The first and the main step is to produce a partition with the goal of achieving minimal energy consumption.

Since power consumption is proportional to the cube of execution frequency, while execution time is just inversely proportional to execution frequency, after achieving a partition, the execution frequency of each processor is slowed down as much as possible under the constraints of our three different assumptions about platforms. Namely, for dependent platforms without runtime adjusting, the common frequency should be chosen such that the processor with the greatest workload completes all of the tasks assigned to it, exactly at the deadline D; for dependent platforms with runtime adjusting, we can further determine the optimal frequencies in different time intervals; for independent platforms, all processors are slowed down independently such that each processor completes all of the tasks assigned to it, exactly at deadline D. Notations that are consistently used in this paper are listed in Table 2. Some of the notations will be made clear later.

TABLE 2
Notations Used in This Paper

Notation	Description
n, m	the number of tasks and processors.
$\mathcal{T}$	a frame-based task set $\{\tau_1, \tau_2, \cdots, \tau_n\}$. τ_i is the ith task.
C_i	Worst Case Execution Cycles (WCEC) of task τ_i.
M_j	the jth processor ($j = 1, 2, \cdots, m$).
$p(f)$	a processor's power when it is executing task(s) at f.
D	the common deadline of task set $\mathcal{T}$.
$\lambda_{i,j}$	M_j's execution efficiency when it is executing τ_i.
AEC_i	average execution cycles of task τ_i.
$x_{n\times m}$	an binary $n \times m$ matrix indicating tasks assignment.
$x_{i,j}$	a binary variable indicating whether τ_i is assigned to M_j. $x_{i,j} = 1$, if τ_i is assigned to M_j; else, $x_{i,j} = 0$; if relaxed, $x_{i,j}$ can be any fraction between 0 and 1.
U_j	normalized effective execution cycles assign to M_j.
P_1	original relaxed optimization problem for dependent platforms without runtime adjusting.
P_i	the ith iterated relaxed optimization problem for dependent platforms without runtime adjusting.
P_1'	original relaxed optimization problem for independent platforms.
P_i'	the ith iterated relaxed optimization problem for independent platforms.
ϵ, γ, μ, c	parameters used during solving a problem by the interior point method.

2.4 Our Approach

By the motivational example, we have noticed that a "workload-balanced" partition is not optimal in terms of overall energy consumption, especially on heterogeneous multiprocessor platforms. Thus, in our consideration, we always place the energy consumption at the highest priority and propose a relaxation-based rounding algorithm for this problem. Our intuition is that an assignment that is closest to the optimal solution for the relaxed problem will achieve a better partition in terms of overall energy consumption.

We first describe a Relaxation-based Naive Rounding Algorithm (RNRA), which solves the relaxed optimization problem once, and produces a partition according to the solution. However, this approach may lead to an accumulated error between the final assignment and the relaxed optimal solution.

Then, we propose a Relaxation-based Iterative Rounding Algorithm. The main idea of our RIRA is as follows.

First, we define the average execution cycle of task τ_i as

$$AEC_i = \frac{1}{m}\sum_{j=1}^{m}\frac{C_i}{\lambda_{i,j}}, \tag{1}$$

and sort the tasks in the order $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_n}$, such that $AEC_{i_1} \geq AEC_{i_2} \geq \cdots \geq AEC_{i_n}$. This is also the order that we will assign tasks in. The intuition here is that the task with the greatest average execution requirement can be

considered as the most "influential" task in terms of both schedulability and energy consumption, and thus, it should be considered first.

Then, we formulate the problem under consideration as a binary integer programming problem, and then we relax it as a convex optimization problem and solve it; based on the optimal solution for the relaxed problem, we assign the most "influential" task to the corresponding processor. After that, we update the optimization problem (since we have already assigned one task, both the objective function and constraints have changed) and relax it again to achieve the solution which will guide the assignment of the next most "influential" task. The above process is repeated until we finish assigning $(n-1)$ tasks. For the last task τ_n, we just select the assignment that achieves the minimal overall energy consumption among all possible assignments of the last task.

In the following sections, we address the problem of scheduling frame-based tasks on heterogeneous platforms with the goal of achieving minimal energy consumption while meeting all of the timing constraints. Dependent platforms without and with runtime adjusting are considered in Sections 3 and 4, respectively. Scheduling on heterogeneous independent platforms is presented in Section 5.

3 Scheduling on Heterogeneous Dependent Multiprocessor Platforms without Runtime Adjusting

3.1 Problem Analysis

We first consider the optimal frequency setting if we have already had a task set partition. Let binary variables $x_{i,j}$ be 1 if task τ_i is assigned to processor M_j, and 0 otherwise. A given partition can be represented by a binary matrix $x_{n\times m}$. We denote the shared frequency among all of the processors during the whole time by f. Then, the time when processor M_j will complete its workload can be calculated as

$$\frac{1}{f}\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}. \tag{2}$$

The shared frequency should guarantee that each processor finishes the tasks on it before the deadline. Thus, to reach minimal energy consumption, the shared frequency can be slowed down as much as possible, as long as all processors' completion times are less than or equal to the deadline D

$$\frac{1}{f}\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}} \le D, \forall j = 1, 2, \ldots, m. \tag{3}$$

The energy consumption on the jth processor M_j is

$$E_j = f^3\left(\frac{1}{f}\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right) = f^2\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}. \tag{4}$$

Thus, to achieve a partition with the objective of saving energy, the problem can be formulated as

$$\begin{aligned}
\min \quad & E_{total} = f^2\sum_{j=1}^{m}\left(\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right)\\
s.t. \quad & \sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}} - fD \le 0, \forall j = 1, 2, \ldots, m,\\
& x_{i,j} = 0 \text{ or } 1, \forall i = 1, 2, \ldots, n; j = 1, 2, \ldots, m,\\
& \sum_{j=1}^{m} x_{i,j} = 1, \forall i = 1, 2, \ldots, n,
\end{aligned} \tag{5}$$

where the optimization variables are the shared frequency f and the binary matrix $x_{n\times m}$. However this binary integer programming problem is hard to solve directly. We relax the binary variables $x_{i,j}$'s to be any fraction in $[0, 1]$. The above optimization problem can be reformulated as

$$\begin{aligned}
\min \quad & E_{total} = f^2\sum_{j=1}^{m}\left(\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right)\\
s.t. \quad & \sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}} - fD \le 0, \forall j = 1, 2, \ldots, m,\\
& 0 \le x_{i,j} \le 1, \forall i = 1, 2, \ldots, n; j = 1, 2, \ldots, m,\\
& \sum_{j=1}^{m} x_{i,j} = 1, \forall i = 1, 2, \ldots, n.
\end{aligned} \tag{6}$$

Denote this relaxed optimization problem by P_1, which is a convex optimization problem that can be efficiently solved by the well-known interior point method [34]. The optimization variables of P_1 are the shared frequency f and the relaxed assignment matrix $x_{n\times m}$. Details of this method will be provided later in Section 3.4. Here, $x_{i,j}$ can be any number between 0 and 1, and it represents the percentage of task τ_i that should be assigned to processor M_j to achieve the minimal overall energy consumption.

3.2 A Simple Algorithm: RNRA

Our first intuition is that if we assign tasks in a way that is "closest" to the optimal solution (for the relaxed problem), we will achieve a better partition in terms of overall energy consumption. One possible way is a naive rounding method to partition the task set. It solves the relaxed problem once, and then assigns the tasks according to this single solution. Basically, it assigns each task, τ_i, to processor M_{j^*}, such that x_{i,j^*} is the maximum among $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$. Algorithm 1 describes this Relaxation-based Naive Rounding Algorithm.

3.3 An Improved Algorithm: RIRA

However, we notice that this approach may lead to an accumulated error between the final assignment and the relaxed optimal solution because the condition for optimal energy consumption changes after we have assigned some tasks. Thus, assigning follow-up tasks according to the original solution may not be optimal in terms of overall energy consumption. Taking this aspect into consideration, we propose the Relaxation-based Iterative Rounding Algorithm. We will describe our RIRA in detail.

In the first step, we sort tasks such that their average execution cycles AEC_i's are in descending order. Without loss of generality and for notional brevity, from now on, we will

assume that all of the tasks are already sorted in our desired order, i.e., $AEC_1 \geq AEC_2 \geq \cdots \geq AEC_n$.

Algorithm 1 RNRA

Input:
The task set $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ and associated WCECs, $C_1, C_2, \cdots, C_n$; processor efficiency matrix, $\lambda_{n\times m}$;

Output:
Binary matrix $Assign_{n\times m}$ indicating the final assignment;

1: Initialize the the assignment matrix: $Assign_{i,j} = 0$ $(\forall i = 1, 2, \cdots, n; j = 1, 2, \cdots, m)$;
2: Solve the relaxed optimization problem P_1. Thus, get the relaxed assignment matrix $x_{n\times m}$;
3: **for** $i := 1$ *to* n **do**
4: $\quad x_{i,j^*} = max(x_{i,1}, x_{i,2}, \cdots, x_{i,m})$;
5: $\quad Assign_{i,j^*} = 1$;
6: **end for**
7: **return** $Assign_{n\times m}$;

Then, relax the original optimization problem as problem P_1. Since tasks are already in our desired order, the solutions $x_{1,1}, x_{1,2}, \ldots, x_{1,n}$ indicate the optimal assignment of the most influential task τ_1. Then, we find the maximum among $x_{1,1}, x_{1,2}, \ldots, x_{1,n}$, denoted by x_{1,j^*}, and assign τ_1 to processor M_{j^*}. Denote the final assignment matrix for the task set by $Assign_{n\times m}$. Then, we have $Assign_{1,j} = 0, \forall j \neq j^*$and $Assign_{1,j^*} = 1$.

Before assigning the next most influential task τ_2, we need to update the optimization problem first. In this process, we should always keep in mind that we have already assigned task τ_1 to processor M_{j^*}, which means that $x_{1,j} = 0, \forall j \neq j^*$ and $x_{1,j^*} = 1$. The expressions for the completion time and the energy consumption on each processor are almost the same as those for problem P_1. Thus, the updated optimization problem can be formulated as

$$\begin{aligned} \min \quad & E_{total} = f^2 \sum_{i=1}^{m} \left(\sum_{i=1}^{n} \frac{x_{i,j} C_i}{\lambda_{i,j}} \right) \\ s.t. \quad & \sum_{i=1}^{n} \frac{x_{i,j} C_i}{\lambda_{i,j}} - fD \leq 0, \forall j = 1, 2, \ldots, m, \\ & 0 \leq x_{i,j} \leq 1, \forall i = 2, \ldots, n; \forall j = 1, 2, \ldots, m, \\ & \sum_{j=1}^{m} x_{i,j} = 1, \forall i = 2, \ldots, n. \end{aligned} \tag{7}$$

Denote this optimization problem by P_2, since it will provide the solution for assigning task τ_2. Notice that, even though the appearance of this updated optimization problem is very similar to the original one (P_1), P_2 is quite different from P_1 because now, $x_{1,1}, x_{1,2}, \ldots, x_{1,m}$ have fixed values, namely, $x_{1,j} = 0, \forall j \neq j^*$, and $x_{1,j^*} = 1$. The optimization variables in P_2 only includes the optimal frequency f, and $x_{2,1}, x_{2,2}, \ldots, x_{2,m}$, $x_{3,1}, x_{3,2}, \ldots, x_{3,m}, \ldots, x_{n,1}, x_{n,2}, \ldots, x_{n,m}$. After solving P_2, we can assign τ_2 according to $x_{2,1}, x_{2,2}, \ldots, x_{2,m}$ (solved for P_2), which is similar to what we do to assign τ_1. Namely, find $x_{2,j'^*} = max(x_{2,1}, x_{2,2}, \ldots, x_{2,m})$, then set $x_{2,j} = 0, \forall j \neq j'^*$, and $x_{2,j'^*} = 1$. Then, we can update the optimization problem as P_3, keeping in mind that we have already assigned task τ_1 and τ_2; solve it and assign task τ_3. Then, solve P_4 to assign τ_4;...; solve P_i to assign τ_i; .... Repeat the process until we finish assigning $(n-1)$ tasks. We notice that, in some cases, assigning the last task according to this iterative scheme may not be optimal. Thus, for the last task, we just select the assignment which can achieve the minimal overall energy consumption among all possible assignments for the last task. Algorithm 2 shows our Relaxation-based Iterative Rounding Algorithm.

Algorithm 2 RIRA

Input:
The sorted task set $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ and associated WCECs, $C_1, C_2, \cdots, C_n$; processor efficiency matrix, $\lambda_{n\times m}$;

Output:
Binary matrix $Assign_{n\times m}$ indicating the final assignment;

1: Initialize the the assignment matrix: $Assign_{i,j} = 0, \forall i = 1, 2, \cdots, n; j = 1, 2, \cdots, m$;
2: **for** $i := 1$ *to* $n-1$ **do**
3: $\quad$ Solve Optimization problem P_i;
4: $\quad x_{i,j^*} = max(x_{i,1}, x_{i,2}, \cdots, x_{i,m})$;
5: $\quad$ **for** $j := 1$ *to* m **do**
6: $\quad\quad x_{i,j} = 0$;
7: $\quad$ **end for**
8: $\quad x_{i,j^*} = 1$; $Assign_{i,j^*} = 1$;
9: $\quad$ Update the optimization problem to be P_{i+1};
10: **end for**
11: Assign the last task τ_n such that the final assignment achieves the minimal energy consumption among all possible assignments for the last task. Denote this by $Assign_{n,j'} = 1$;
12: **return** $Assign_{n\times m}$;

3.4 Algorithm Analysis

As has been mentioned, we apply the interior point method to solve the problems $P_i, \forall i = 1, 2, \ldots, n$. Interior-point methods solve a convex optimization problem or its corresponding KKT conditions by applying Newton's method to a sequence of equality constrained problems, or to a sequence of modified versions of the KKT conditions [34]. To gain a clear understanding of RIRA's complexity and accuracy, we apply a particular interior point algorithm: the barrier method to solve the optimization problems, P_i's. The barrier method uses a logarithmic barrier function to approximate the inequality constrained problem as an equality constrained problem to which Newton's method can be applied. The logarithmic barrier function is with the form:

$$\hat{I}_(u) = -\frac{1}{t}\log(-u) \tag{8}$$

where $t > 0$ is a parameter that sets the accuracy of the approximation. The total number of Newton iterations to solve one optimization problem is

$$N = \left\lceil \frac{\log(\overline{m}/(t^{(0)}\epsilon))}{\log \mu} \right\rceil \left(\frac{\overline{m}(\mu - 1 - \log \mu)}{\gamma} + c \right), \tag{9}$$

TABLE 3
An Illustration Example

i	C_i	$\lambda_{i,j}$			$t_{i,j}$		
		$j=1$	$j=2$	$j=3$	$j=1$	$j=2$	$j=3$
1	7	.7	.4	.1	10	17.5	70
2	8	.5	.2	.3	16	40	26.67
3	3	.4	.1	.2	7.5	30	15
4	5	.5	.2	.4	10	25	12.5
5	9	.6	.9	.7	15	10	12.86
6	5	.8	.3	.5	6.25	16.67	10
7	4	.3	.9	.6	13.33	4.44	6.67
8	4	.4	.6	.8	10	6.67	5

TABLE 4
Assignment by RNRA

i	Relaxed Assignment Matrix $x_{8\times3}$			$Assign_{8\times3}$		
	$j=1$	$j=2$	$j=3$	$j=1$	$j=2$	$j=3$
1	0.2920	0.7080	0	0	1	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	0	1	0	0	1
5	0	1	0	0	1	0
6	0.0665	0	0.9335	0	0	1
7	0	1	0	0	1	0
8	0	0	1	0	0	1

where $\overline{m}$ is the total number of inequalities in the problem (in P_1, $\overline{m} = m + nm$); $t^{(0)} > 0$ is the initial value of t for the logarithmic barrier function; $\mu > 1$ is the increase extent for $t$in the iteration, namely, $t^{(1)} = \mu t^{(0)}$, $t^{(2)} = \mu t^{(1)}$; γ and c are parameters used in the Newton's method; ϵ is the tolerance parameter. If we choose parameters μ, c, and γ as constants, the complexity for solving one convex problem can be simplified as: $O(mn(\log m + \log n + \log \frac{1}{\epsilon}))$. Since our proposed RIRA involves iteratively solving n optimization problems, the complexity of RIRA is

$$O\left(mn^2\left(\log m + \log n + \log \frac{1}{\epsilon}\right)\right). \tag{10}$$

3.5 Illustration Example

An illustration example is provided, which considers scheduling eight tasks to three processors. Tasks' WCECs, $C_1, C_2, \ldots, C_8$ and the processor efficiency matrix, $\lambda_{8\times3}$ are given in Table 3. For the need of some other techniques, a reference time matrix is derived as $t_{8\times3}$, where $t_{i,j} = C_i/\lambda_{i,j}$, which is also provided in the same table. The common deadline for this task set is $D = 100$. Notice that, in this example, tasks are already in the required order for RIRA.

Fig. 2a shows the partition by the min-min heuristic. Fig. 2b shows the partition by the max-min heuristic. The RNRA solves P_1 (for this example) and gets the relaxed assignment matrix $x_{8\times3}$, as shown in Table 4. Obviously, by the naive rounding scheme, it can easily achieve the final assignment matrix $Assign_{8\times3}$. Fig. 2c shows the partition by RNRA. Our RIRA first solves the original optimization problem P_1; assign τ_1 according to solutions $x_{1,1}, x_{1,2}, x_{1,3}$ (solved for P_1). Then, it updates the optimization problem as P_2, solves it, and assigns τ_2 according to solutions $x_{2,1}, x_{2,2}, x_{2,3}$ (solved for P_2). Repeat the above process until it assigns seven tasks; for the last task, it selects the assignment that achieves the minimal energy consumption. Relevant solutions are shown in Table 5. Fig. 2d shows the partition by our proposed RIRA. In each subfigure in Fig. 2, the number behind a task is the order in which this task is assigned; for example, $\tau_7(1)$ in Fig. 2a means that τ_7 is the first task that is assigned.

After a partition, the processors are slowed down dependently such that the processor with the greatest completion time meets the predefined deadline $D = 100$ exactly. In this example, the shared frequencies for the min-min, max-min, RNRA and RIRA partitions are 0.3975, 0.3417, 0.3194, and 0.3194, respectively. Note that, though the shared frequencies for RNRA and RIRA partitions are the same, the actual partitions are not the same. The energy consumption for these four partitions can be calculated as follows: 11.3 for the min-min heuristic, 10.7 for the max-min heuristic, 8.464 for RNRA, and 8.08 for our RIRA. Obviously, our proposed RIRA achieves the best partition in terms of overall energy consumption.

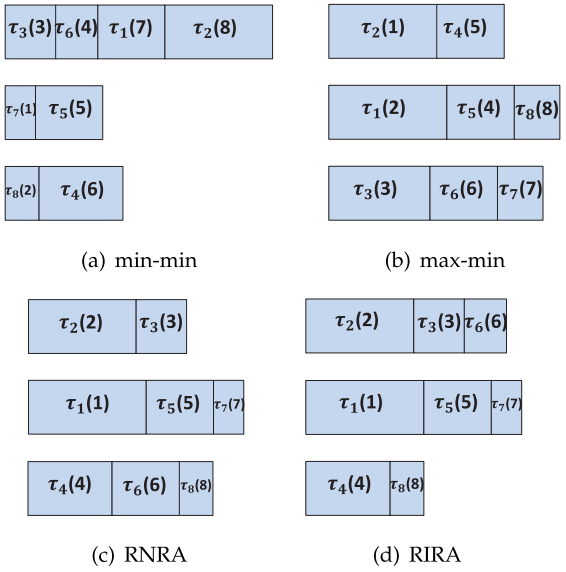


Fig. 2. Partitions by different approaches.

4 Scheduling on Heterogenous Dependent Platforms with Runtime Adjusting

4.1 Problem Analysis

For dependent platforms with runtime adjusting, the approach of Yang et al. [11] is applied here to determine the optimal frequency scheduling during different time

TABLE 5
Assignment by RIRA

i	Relaxed Assignment $x_{i,j}$ for P_i			$Assign_{8\times3}$		
	$j=1$	$j=2$	$j=3$	$j=1$	$j=2$	$j=3$
1	0.2920	0.7080	0	0	1	0
2	1	0	0	1	0	0
3	0.99984	0.00001	0.00015	1	0	0
4	0.00013	0.00001	0.99986	0	0	1
5	0	0.5379	0.4621	0	1	0
6	0.6504	0	0.3496	1	0	0
7	0	0.5062	04938	0	1	0
8	-	-	-	0	0	1

intervals after we have achieved a partition. Given a partition, denoted by an assignment matrix $x_{n\times m}$, again, $x_{i,j} = 1$ if task τ_i is assigned to processor M_j, and is 0 otherwise. We define the normalized effective execution cycles assigned to processor M_j as

$$U_j = \sum_{i=1}^{n} \frac{x_{i,j} C_i}{\lambda_{i,j}}. \tag{11}$$

Without loss of generality, we assume that processors are in ascending order of their U_j values, i.e., $U_1 \le U_2 \le \cdots \le U_m$. Because tasks do not have precedence constraints, each processor can execute its workload continuously without any interruption. Since all of the active processors must share a common frequency (though the shared frequency can vary with time), the processor with a less U_j value will complete its tasks earlier than that with a greater U_j value. Introduce $U_0 = 0$; we say that the jth time interval is the interval between the time when U_{j-1} is completed and the time when U_j is completed. Assume that the shared frequency of all running processors during the jth interval is f_j. The length of the j time interval is

$$t_j = (U_j - U_{j-1})/f_j, \tag{12}$$

where f_j is the shared frequency of the running processors during this time interval. We can notice that, during the first interval, all processors are running. During the second interval, the first processor, which has the least effective execution cycles, has finished its tasks; thus, only $(m-1)$ processors are running. Similarly, during the jth interval, only $(m-j+1)$ processor(s) are running. Thus, the energy consumption during the jth time interval is

$$E_j^{'} = (m-j+1) f_j^2 (U_j - U_{j-1}), \tag{13}$$

where $(m-j+1)$is the number of processors that are in run mode during time interval t_j. Thus, the frequency setting problem can be formulated as

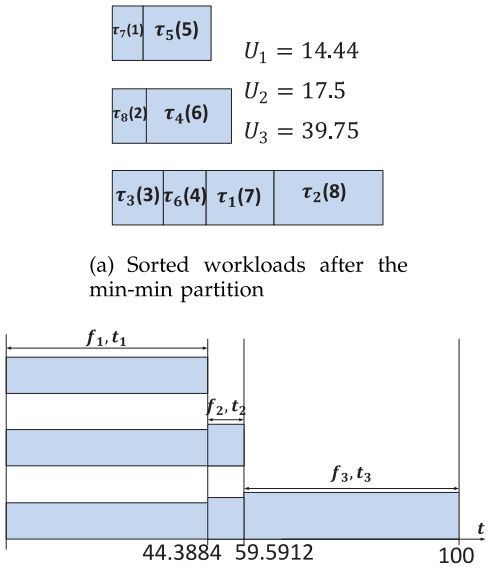
$$\begin{aligned} \min \quad & E_{total} = \sum_{j=1}^{m} (m-j+1) f_j^2 (U_j - U_{j-1}), \\ s.t. \quad & \sum_{j=1}^{m} (U_j - U_{j-1})/f_j \le D. \end{aligned} \tag{14}$$

Intuitively, it is more energy-efficient to execute at a lower frequency when more processors still have workloads, while use a higher frequency when less processors still have workloads. Actually, the above optimization problem can be solved by the Lagrange Multiplier Method directly, and the optimal frequency for the jth time interval can be achieved [11]:

$$f_j = \frac{\sum_{j=1}^{m} (U_j - U_{j-1}) \sqrt[3]{m-j+1}}{D \sqrt[3]{m-j+1}}. \tag{15}$$

4.2 Approach

Although the optimal f_j can be solved analytically, it is based on sorting the workloads on different processors first. Thus, the above analysis does not contribute to finding an energy-efficient partition in the first place. Since we are still considering a dependent platform, in our approach, we adopt the same partition, derived by the RIRA algorithm for dependent platforms without runtime adjusting, as the partition for the dependent platforms with runtime adjusting.



(a) Sorted workloads after the min-min partition

(b) Runtime frequency adjusting, $f_1 = 0.3254$, $f_2 = 0.3725$, $f_3 = 0.4693$, $t_1 = 44.3884$, $t_2 = 8.2028$, $t_3 = 47.4088$

Fig. 3. Runtime frequency adjusting for min-min partition.

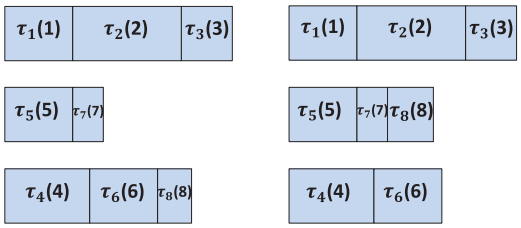
4.3 Example

Take the partition by the min-min heuristic in Fig. 2a as an illustration example to demonstrate this frequency adjusting procedure. The runtime frequency adjusting procedures are shown in Fig. 3. Fig. 3a shows the sorted workloads among the three processors. After this, we can determine the optimal frequencies for the three time intervals. For this partition, by the Lagrange Multiplier Method, we can get: $f_1 = 0.3254$, $f_2 = 0.3725$, $f_3 = 0.4693$, $t_1 = 44.3884$, $t_2 = 8.2028$, $t_3 = 47.4088$. The final scheduling for this partition is shown in Fig. 3b. The overall energy consumption is reduced from 11.3 to 10.3375.

After applying this runtime frequency adjusting scheme for all four partitions in Fig. 2, their overall energy consumptions can be achieved: 10.3375 for the partition by the min-min heuristic, 10.4740 for the max-min heuristic, 8.1617 for RNRA, and 7.8776 for our RIRA. Our proposed RIRA also achieves the best partition. This verifies that our partition method can also provide a good solution on dependent platforms with runtime adjusting.

5 Scheduling on Heterogeneous Independent Multiprocessor Platforms

In this section, we will apply our approaches for energy-aware scheduling frame-based tasks on heterogeneous independent multiprocessor platforms. Since many of the procedures are similar to that of Section 3, we will omit some details.



(a) RNRA (b) RIRA

Fig. 4. Partitions by RNRA and RIRA on independent platforms.

5.1 Problem Analysis

Similar to what we do in Section 3, we first consider the optimal frequency setting after we have had a partition, denoted by a binary matrix $x_{n\times m}$. Since we assume independent platforms here, in order to achieve minimal energy consumption, the optimal frequency for the jth processor can be determined as

$$f_j = \frac{1}{D}\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}. \tag{16}$$

Then, the energy consumption on the jth processor M_j is

$$E_j^{''} = f_j^3 D = \frac{1}{D^2}\left(\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right)^3. \tag{17}$$

Thus, to achieve the energy-optimal partition, it is equivalent to solve the following optimization problem:

$$\begin{aligned} \min \quad & E_{total} = \frac{1}{D^2}\sum_{j=1}^{m}\left(\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right)^3 \\ s.t. \quad & \sum_{j=1}^{m} x_{i,j} = 1, \forall i = 1,2,\ldots,n \\ & x_{i,j} = 0 \text{ or } 1, \forall i = 1,2,\ldots,n, \forall j = 1,2,\ldots,m. \end{aligned} \tag{18}$$

Again, we relax $x_{i,j}$to be any fraction within $[0,1]$, the problem is transformed as follows:

$$\begin{aligned} \min \quad & E_{total} = \frac{1}{D^2}\sum_{j=1}^{m}\left(\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right)^3, \\ s.t. \quad & \sum_{j=1}^{m} x_{i,j} = 1, \forall i = 1,2,\ldots,n, \\ & 0 \le x_{i,j} \le 1, \forall i = 1,2,\ldots,n, \forall j = 1,2,\ldots,m. \end{aligned} \tag{19}$$

Denote the above optimization problem as $P_1^{'}$.

5.2 Algorithms

RNRA solves problem $P_1^{'}$, and adopts the same process as in Algorithm 1 to assign all of the tasks. Our RIRA solves problem $P_1^{'}$ first, and assigns task τ_1 according to solutions $x_{1,1}, x_{1,2}, \ldots, x_{1,m}$; then, it updates the optimization problem as $P_2^{'}$:

TABLE 6
The Frequency Setting of the Final Schedulings

Processor	Frequency Settings			
	min-min	max-min	RNRA	RIRA
M_1	0.3975	0.2600	0.3350	0.3350
M_2	0.1444	0.3417	0.1444	0.2111
M_3	0.1750	0.3167	0.2750	0.2250

$$\begin{aligned} \min \quad & E_{total} = \frac{1}{D^2}\sum_{j=1}^{m}\left(\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}\right)^3, \\ s.t. \quad & \sum_{j=1}^{m} x_{i,j} = 1, \forall i = 2,\ldots,n, \\ & 0 \le x_{i,j} \le 1, \forall i = 2,\ldots,n; \forall j = 1,2,\ldots,m. \end{aligned} \tag{20}$$

Assign task τ_2 according to the solutions $x_{2,1}, x_{2,2}, \ldots, x_{2,m}$ (solved for $P_2^{'}$). Notice that the optimization variables of $P_2^{'}$ only includes $x_{2,1}, x_{2,2},\ldots, x_{2,m}$, $x_{3,1}, x_{3,2},\ldots, x_{3,m}$, $\ldots$, $x_{n,1}, x_{n,2},\ldots, x_{n,m}$, since τ_1 has already been assigned; in other words, $x_{1,1}, x_{1,2}, \ldots, x_{1,m}$ have fixed values. Repeat updating and assigning in the same way as in Algorithm 2; the only difference is that the jth relaxed optimization problem for the independent platform is denoted by $P_i^{'}$. By this way, an energy-efficient partition for independent platforms can be achieved. Given this partition, each processor reduces adjusting its execution frequency independently such that it finishes its assigned workload exactly at the deadline D, to reduce the energy consumption.

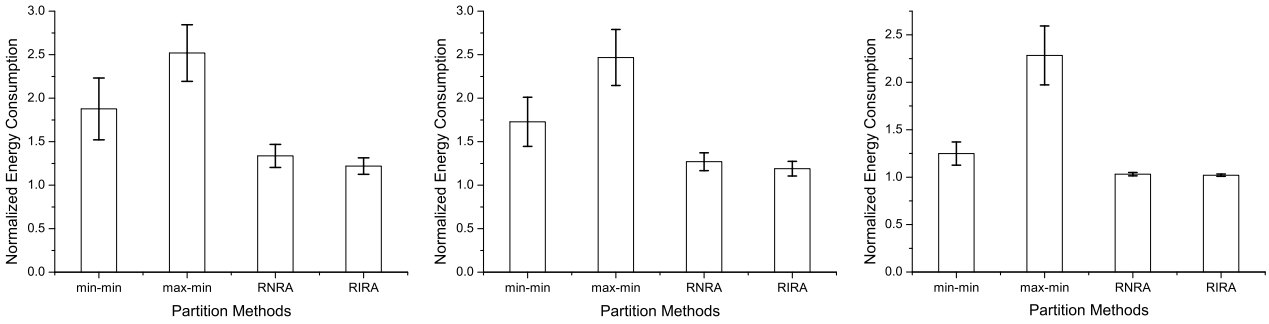
5.3 Example

For the example in Table 3, obviously, the min-min and max-min heuristics will produce the same partitions as in Figs. 2a and 2b, respectively. However, RNRA and RIRA will produce partitions different from those in Figs. 2c and 2d, respectively. Using the same example on independent platforms, the partitions derived by RNRA and RIRA are shown in Figs. 4a and 4b, respectively. For each given partition, to achieve minimal energy consumption, all of the three processors should adjust their execution frequency independently such that their workloads finish exactly at the deadline $D = 100$. The frequency setting for the three processors in the four partition methods are shown in Table 6.

The energy consumption for the four partitions on independent platforms are: 7.11 for the min-min heuristic, 8.92 for the max-min heuristic, 6.14 for RNRA and 5.84 for our RIRA. Our proposed RIRA still achieves the best partition in terms of overall energy consumption.

6 Performance Evaluation

In this section, we conduct extensive simulations and experiments to evaluate our RIRA. For each task assignment problem on a multiprocessor platform, the four described partitioning methods, namely, the min-min heuristic, the max-min heuristic, RNRA, and RIRA, are applied to three types of platform assumptions. We normalize energy consumption for each case by their corresponding optimal energy consumption, which is the solution for the first relaxed optimization problems, namely, P_1, and $P_1^{'}$.



(a) Type I platform (b) Type II platform (c) Type III platform

Fig. 5. Setting I: given task set.

6.1 Simulation Settings

From the first setting to the third setting, simulations are done for scheduling 24 tasks on six processors. In the fourth setting, we consider scheduling 64 tasks on 16 processors. We design the fifth setting to reveal the relation of the execution time of our proposed algorithm with respect to the tolerance parameter ϵ. In the sixth setting, we study the scalability of RIRA by conducting experiments for various numbers of processors and numbers of tasks. In all settings, the unit of execution requirements is 10^6. We conduct all the simulations in MATLAB (Version 7.9.0.529 (2009b)) [35] on a 32-bit Windows XP system with Intel® Core 2 Duo CPU.

In the first setting, we evaluate the performance of our proposed RIRA for different processor efficiency matrices and a fixed task set. Specifically, we choose the fixed set of tasks with execution requirements: $C = [5, 5, 5, 5, 5, 5, 5, 5; 10, 10, 10, 10, 10, 10, 10, 10;\ 15, 15, 15, 15, 15, 15, 15, 15]$. We randomly generate 50 processor efficiency matrices $\lambda_{24\times 6}$. Within each matrix, the $\lambda_{i,j} (i = 1, 2, \ldots, 24; j = 1, 2, \ldots, 6)$ values are uniformly distributed in $[0.1, 1]$. Note that we are not emphasizing the actual distribution pattern of the efficiency values, and just choose the simple uniform distribution. [0.1, 1] is chosen to reflect the heterogeneity among the processors. Efficiency values less than 0.1 are not considered, because practical multiprocessor systems are not likely to have efficiency values differing too much from each other. For each processor efficiency matrix, we can achieve the normalized energy consumption (normalized to the optimal energy consumption for the first relaxed problem) of the four partition methods under a given platform assumption. We compare each partition method's 50 normalized energy consumption values by computing their means and standard deviations.

In the second setting, we evaluate the performance of our proposed RIRA for different task sets and a fixed processor efficiency matrix. More specifically, we consider the special case where processors have different efficiencies, while one processor has the same efficiency for different tasks. We consider the following example: $\lambda_{i,1} = 1$, $\lambda_{i,2} = 0.82$, $\lambda_{i,3} = 0.64$, $\lambda_{i,4} = 0.46$, $\lambda_{i,5} = 0.28$, $\lambda_{i,6} = 0.1$, $\forall i = 1, 2, \ldots, 24$. In a sense, these values are uniformly distributed in $[0.1, 1]$. We randomly generate 50 task sets. For each task set, the C_i values are uniformly distributed between $[5, 15]$. For each task set, we can achieve the normalized energy consumption (normalized to the optimal energy consumption for the first relaxed problem) of the four partition methods under a given platform assumption. We compare each partition method's 50 normalized energy consumptions by computing their means and standard deviations.

In the third setting, we randomly generate 20 processor efficiency matrices and 20 task sets. In each of the 20×20 cases, all $\lambda_{i,j}$ values and C_i values are uniformly distributed in $[0.1, 1]$ and $[5, 15]$, respectively. For a given processor efficiency matrix and a platform assumption, we average the normalized energy consumption over the normalized energy consumption of the 20 randomly generated task sets, and then compare these 20 average normalized energy consumptions.

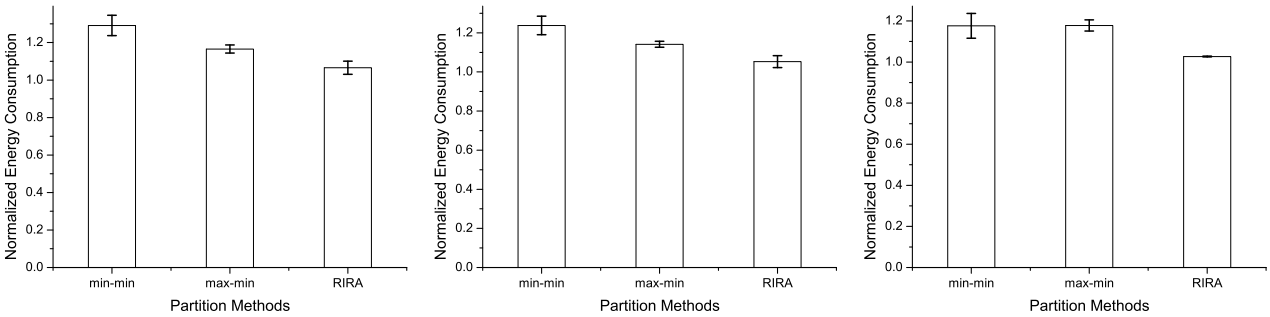
In the fourth setting, we conduct simulations for scheduling 64 tasks on 16 processors. For each platform type, we randomly generate five processor efficiency matrices and 20 task sets. For a given processor efficiency matrix and a platform assumption, we average the normalized energy consumption over the normalized energy consumption of the 20 randomly generated task sets, and then compare these five average normalized energy consumptions.

We also design the fifth setting to investigate the performance and complexity of our RIRA, without comparing it to other approaches. The result is done for scheduling 24 tasks on six processors. We fix $C = [5, 5, 5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 10, 10, 10, 15, 15, 15, 15, 15, 15, 15, 15]$ and set $\lambda_{i,1} = 1$, $\lambda_{i,2} = 0.82$, $\lambda_{i,3} = 0.64$, $\lambda_{i,4} = 0.46$, $\lambda_{i,5} = 0.28$, $\lambda_{i,6} = 0.1$, $\forall i = 1, 2, \ldots, 24$. Under these settings, we vary the tolerance parameter, ϵ, as $10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$ and evaluate the elapsed time to derive a partition, and the normalized energy consumption, by the RIRA approach.

In the sixth setting, we vary the numbers of processors and the numbers of tasks to verify the scalability of our proposed RIRA. We vary the number of processors from six to 16, with a step size of two; given a number of processors, we vary the number of tasks from 24 to 88, with a step size of 16. Detailed simulation configurations are omitted due to page limit.

6.2 Simulation Results and Analysis

For the ease of presentation, we use "Type I", "Type II", and "Type III" platforms to denote dependent platforms without runtime adjusting, dependent platforms with runtime adjusting, and independent platforms, respectively. Results for the first setting is provided in Fig. 5. In each figure of



(a) Type I platform (b) Type II platform (c) Type III platform

Fig. 6. Setting II: given processor efficiency matrix $\lambda_{24\times 6}$.

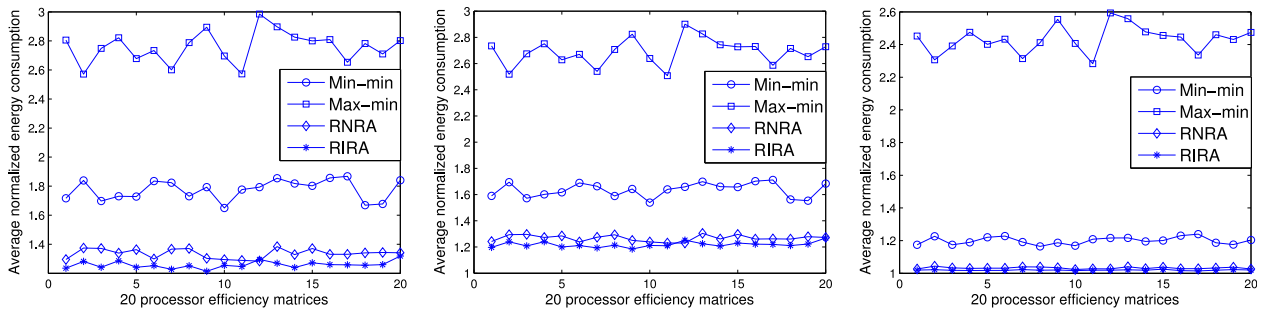
Fig. 5, the horizontal axis represents the partition methods as indicated. The vertical axis represents the normalized energy consumption for 50 randomly generated processor efficiency matrices. Obviously, our RIRA achieves the best performance in all of the three platform types. For dependent platforms without runtime adjusting, RIRA reaches 1.2195 times that of the optimal energy consumption; for dependent platforms with runtime adjusting, RIRA reaches 1.1893 times that of the optimal energy consumption; for independent platforms without runtime adjusting, RIRA reaches 1.0205 times that of the optimal energy consumption, which is only about 2 percent greater than the optimal energy consumption. For this setting, the normalized energy consumption of RIRA achieves the smallest standard deviation. Thus, our RIRA is superior in comparison to the other three methods, in terms of both average performance and stability.

Results for the second setting are provided in Fig. 6. In each figure of Fig. 6, the vertical axis represents the normalized energy consumption for 50 randomly generated task sets. We notice that, in this special case, the RNRA attempts to assign all of the tasks to the most efficient processor, i.e., processor M_1. Thus, the normalized energy consumption of RNRA (about six) is much greater than the other three methods (less than two). For clear comparisons between our RIRA and the min-min and max-min heuristics, we do not show the results for the RNRA method in Fig. 6. It can be seen that our RIRA still provides the best performance. For the three types of platforms, the average normalized energy consumptions are only 1.0665, 1.0528, and 1.0267, respectively, which means that the overall energy consumption is within 10 percent greater than the optimal energy consumption, and can be considered extremely good, though its standard deviation may be slightly greater than some of the other methods.

Also, it can be noticed that the min-min heuristic achieves a better performance than max-min in the first setting, while max-min achieves a better performance than min-min in the second setting. Thus, neither min-min nor max-min is a pure winner when compared with each other.

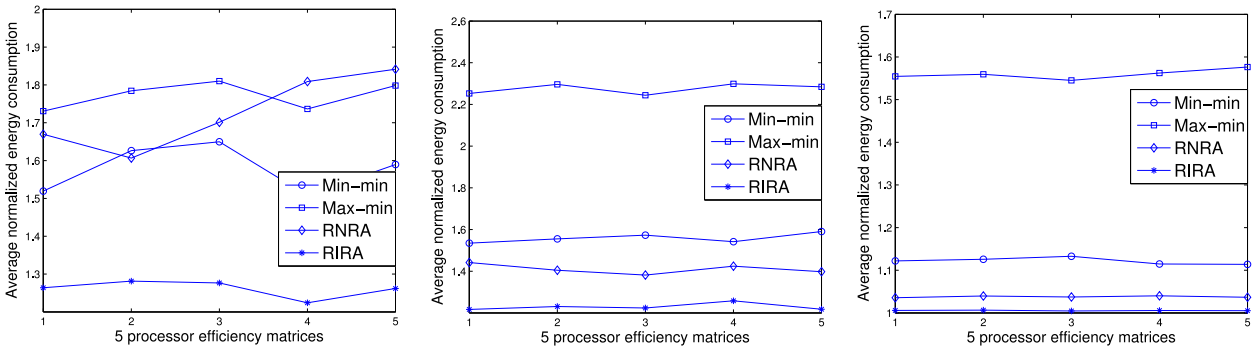
Results for the third setting are shown in Fig. 7. In each type of platform in Fig. 7, the horizontal axis represents the 20 randomly generated processor efficiency matrices. The vertical axis represents the average normalized energy consumption of the 20 randomly generated sets of tasks, given a processor efficiency matrix. Results for all of the 20 randomly generated processor efficiency matrices are provided. This general setting further verifies that the average performance of our proposed RIRA is clearly better than other methods and is also stable under various cases. Results for the fourth setting are shown in Fig. 8. RIRA still arrives at the best solution in terms of energy consumption. Again, it demonstrates its good average performance and high stability.

Fig. 9 provides the results for the fifth setting. The normalized energy consumption is the same as previously defined. The "normalized elapsed time" is normalized by the elapsed time when the tolerance parameter is set as $\epsilon = 10^{-6}$, which is actually around 8 seconds on a common PC. As can be seen, the running time of RIRA decreases linearly with respect to $\log(\epsilon)$, which justifies the complexity analysis provided in Section 3.4. Besides, we can notice that, even though the tolerance parameter increases from 10^{-6} to 10^{-2}, the final scheduling produced by the RIRA approach almost remains unchanged. This is because in our RIRA, we



(a) Type I platform (b) Type II platform (c) Type III platform

Fig. 7. Setting III: both task sets and processor efficiency matrices are randomly generated.



(a) Type I platform (b) Type II platform (c) Type III platform

Fig. 8. Setting IV: Simulation for scheduling 64 tasks on 16 processors.

actually do not need very accurate solutions for the relaxed optimization problems; in fact, we only need a "rough solution" as long as it can guide our assigning decision. Thus, we can increase the tolerance parameter when solving the relaxed optimization problems to reduce its computational complexity.

The results of the sixth setting are provided in Table 7. When we have a set of 88 frame-based tasks to schedule on 16 processors, the execution time of RIRA is about 1,500 s. We can also see that the number of tasks has a much greater influence on the execution time than the number of processors. The results verify that RIRA is a polynomial time algorithm and has satisfiable scalability, and is especially applicable for moderate and small size problems.

6.3 Simulation Summary

From all of the above experiments and comparisons, we can see that our proposed RIRA method outperforms all other methods in terms of overall energy consumption and achieves a near optimal solution for various situations, especially on independent platforms. Our RIRA achieves a good performance mainly because of the two techniques it applies. The first technique is the ranking scheme, namely, considering the most "influential" task first, since such a task potentially has the greatest influence on the overall energy consumption. The second technique is the iterative rounding scheme. When applying this technique, whenever we consider assigning a task, the assignment that is closest to the optimal solution can be chosen. Thus, our RIRA finally produces a partition with an extremely good performance in terms of overall energy consumption.

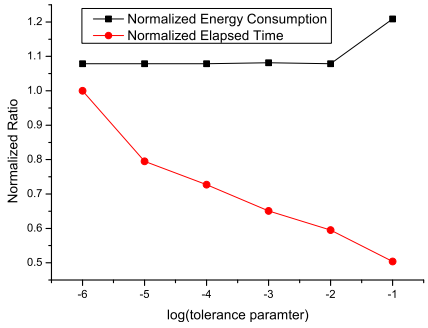


Fig. 9. Elapsed time and NEC versus ϵ

6.4 Applying Our Approach to Practical Power Configurations

By now, we have assumed that processors' dynamic power consumption is $p = f^3$; however, our approach is not limited by this assumption. Actually, our proposed RIRA works well for the general assumption that the power consumption of processor M_j is $p_j \propto f^{\alpha}$, where α is a positive number greater than two. Besides numerical simulation, we also consider a multiprocessor platform with practical power configuration. We are aware that practical processing cores are only able to execute on a set of discrete frequency values, instead of arbitrary continuous values. For a practical multi-core processor, we first apply the curve-fitting technique for the frequency and power characteristics using the form of $p(f) = \gamma f^{\alpha}$. We assume that each processor has the same power characteristics of the Intel Xscale processor, as shown in Table 8 [36]. Then, we apply our method and derive the optimal frequency setting. After this, we round each derived frequency value to the closest higher frequency. Though other techniques that use both the closest lower frequency and the closest high frequency can be used, we choose the simple rounding up strategy to show the advantage of our final practical scheduling against other scheduling methods. Applying the curve-fitting technique, we achieve a fitting function: $p(f) = 1{,}598 f^{2.595}$. We also consider scheduling 24 tasks on six such processors. We fix the processor efficiency matrix as that in the second

TABLE 7
Execution Time (in seconds) of RIRA for Different Problem Sizes

m	n				
	24	40	56	72	88
6	8.6	24.8	56.4	94.8	165.4
8	12.1	34.6	83.4	154.2	290.6
10	15.1	43.5	150.4	271.8	499.8
12	20.4	60.1	185.4	321.4	721.8
14	29.8	93.2	231.6	478.2	997.5
16	31.6	136.0	310.7	618.3	1501.1

TABLE 8
Power Characteristics of the Intel XScale Processor

k	1	2	3	4	5
frequency, f_k (GHz)	0.15	0.40	0.60	0.80	1.00
power, p_k (mW)	80	170	400	900	1600

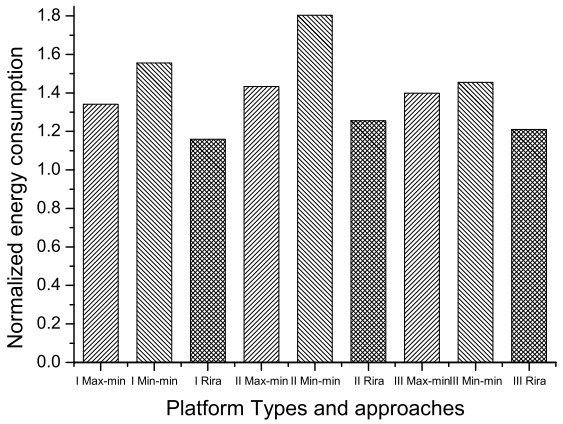


Fig. 10. Applying our method to practical power configurations.

setting and generate tasks' WCEC values such that the optimal frequency settings are well below 1 GHz. For the same reason stated in the second setting, we do not include RNRA in the comparison. The results for all of the three types of platforms are shown in Fig. 10. Each bar represents the normalized energy consumption for the specified platform type and the partition method. We can see that our RIRA also outperforms max-min and min-min.

6.5 Additional Remarks

For the ease of presentation, in most cases, we have assumed $p_j \propto f^3$. However, as has already been shown, our proposed RIRA works well for the general assumption that the power consumption of processor M_j is $p_j \propto f^{\alpha}$, where α is a positive number greater than 2.

It should also be noticed that our scheduling by the RIRA approach is an offline static scheme. The static scheduling overhead can be considered negligible compared to tasks' execution times. Once the static scheduling is produced by our proposed RIRA, little runtime/online scheduling overhead is required. Thus, though iterative rounding and solving are introduced in our algorithm, it only increases the complexity of finding the task partition and frequency setting, but does not increase the runtime scheduling overhead. Specifically, the task partition and frequency setting will be fixed after their initial setting for dependent platforms without runtime adjusting, and no online scheduling will be involved. Thus, the online scheduling overhead is very limited. This low scheduling overhead property is also enjoyed by our approach that attacks the problem on independent platforms. As we can see also, for dependent platforms with runtime adjusting, the frequency read justing only occurs $m-1$ times, which also demonstrates a low scheduling overhead. These aspects make RIRA a practically-applicable scheduling scheme.

7 Conclusion and Future Work

In this paper, we addressed the problem of scheduling frame-based tasks on heterogeneous platforms with the goal of minimizing overall energy consumption. We proposed a Relaxation-based Iterative Algorithm for three types of heterogeneous platforms, namely, dependent platforms without runtime adjusting, dependent platforms with runtime adjusting, and independent platforms. We notice that a "workload-balanced" partition is not optimal in terms of overall energy consumption. In our algorithm, when assigning each task, we always place the overall energy consumption at the highest priority. Thus, our proposed RIRA produces a better performance than existing methods, and achieves a near optimal solution for most cases. Experiments and comparisons from three different angles verify the strength of our algorithm.

As we have pointed out, the ranking method (always considering the most influential task first) and iterative rounding approach also have their merits when we come to various integer, especially binary integer, programming problems. We only consider frame-based tasks in our work, while it is also applicable to use our iterative rounding scheme to partition periodic tasks on heterogeneous multiprocessor platforms. The proposed RIRA approach does not work for the situation where tasks have different deadlines. Addressing energy-aware scheduling for tasks with different deadlines are beyond the scope of this paper, and is left for future work.

Acknowledgments

This research was supported in part by NSF Grants ECCS 1231461, ECCS 1128209, CNS 1065444, and CCF 1028167.

References

[1] K. Li, "Power allocation task scheduling on multiprocessor computers with energy time constraints," in *Energy-Efficient Distributed Computing Systems*. Hoboken, NJ, USA: Wiley 2012.

[2] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, Nov. 2002, pp. 721–725.

[3] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 1, pp. 211–230, Feb. 2005.

[4] M. Li and F. F. Yao, "An efficient algorithm for computing optimal discrete voltage schedules," *SIAM J. Comput.*, vol. 35, no. 3, pp. 658–671, Sep. 2005.

[5] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proc. 36th Annu. Symp. Found. Comput. Sci.*, 1995, pp. 374–382.

[6] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc. Des. Autom. Conf.*, 2001, pp. 828–833.

[7] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. 36th Des. Autom. Conf.*, 1999, pp. 134–139.

[8] H.-S. Yun and J. Kim, "On energy-optimal voltage scheduling for fixed-priority hard real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 2, no. 3, pp. 393–430, Aug. 2003.

[9] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proc. Int. Parallel and Distrib. Process. Symp.*, pp. 113–121, 2003.

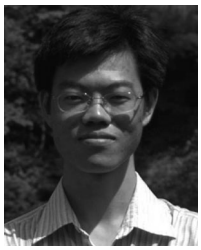
[10] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–6.

[11] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *Proc. Des., Autom. Test Eur.*, Mar. 2005, vol. 1, pp. 468–473.

[12] 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 *Proc. 16th Euromicro Conf. Real-Time Syst.*, Jun.-Jul. 2004, pp. 101–108.

[13] J.-J. Chen and T.-W. Kuo, "Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics," in *Proc. Int. Conf. Parallel Process.*, Jun. 2005, pp. 13–20.

[14] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, Mar. 2011, pp. 1–6.
[15] D.-S. Zhang, F.-Y. Chen, H.-H. Li, S.-Y. Jin, and D.-K. Guo, "An energy-efficient scheduling algorithm for sporadic real-time tasks in multiprocessor systems," in *Proc. IEEE 13th Int. Conf. High Perform. Comput. Commun.*, 2011, pp. 187–194.
[16] V. Nelis, J. Goossens, R. Devillers, and N. Navet, "Power-aware real-time scheduling upon identical multiprocessor platforms," in *Proc. IEEE Int. Conf. Sens. Netw., Ubiquitous Trustworthy Comput.*, 2008, pp. 209–216.
[17] V. Nelis and J. Goossens, "Mora: An energy-aware slack reclamation scheme for scheduling sporadic real-time tasks upon multiprocessor platforms," in *Proc. 15th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2009, pp. 210–215.
[18] N.-T. Fong, C.-Z. Xu, and L. Y. Wang, "Optimal periodic remapping of dynamic bulk synchronous computations," *J. Parallel Distrib. Comput.*, vol. 63, no. 11, pp. 1036–1049, 2003.
[19] X. Tang, K. Li, Z. Zeng, and B. Veeravalli, "A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 1017–1029, Jul. 2011.
[20] M. A. Awan and S. M. Petters, "Energy-aware partitioning of tasks onto a heterogeneous multi-core platform," in *Proc. IEEE 19th Real-Time Embedded Tech. Appl. Symp.*, Apr. 2013, pp. 205–214.
[21] E. M. Saad, M. H. Awadalla, M. Shalan, and A. Elewi, "Energy-aware task partitioning on heterogeneous multiprocessor platforms," *CoRR*, vol. abs/1206.0396, 2012.
[22] W. Sun and T. Sugawara, "Heuristics and evaluations of energy-aware task mapping on heterogeneous multiprocessors," in *Proc. Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 599–607.
[23] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in *Proc. Des., Autom. Test Eur. Conf. Exhibition*, Apr. 2009, pp. 694–699.
[24] J.-J. Chen and L. Thiele, "Task partitioning and platform synthesis for energy efficiency," in *Proc. 15th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2009, pp. 393–402.
[25] 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 *Proc. 27th IEEE Int. Real-Time Syst. Symp.*, Dec. 2006, pp. 303–312.
[26] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *Proc. 9th IEEE/ACM Int. Symp. Cluster Comput. Grid*, May 2009, pp. 92–99.
[27] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorithm for grid task scheduling," in *Proc. 3rd IEEE/IFIP Int. Conf. Central Asia Internet*, Sep. 2007, pp. 1–7.
[28] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 59, pp. 107–131, Nov. 1999.
[29] H. Izakian, A. Abraham, and V. Snasel, "Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments," in *Proc. Int. Joint Conf. Comput. Sci. Optimization*, Apr. 2009, vol. 1, pp. 8–12.
[30] D. Li and J. Wu, "Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms," in *Proc. 41st Int. Conf. Parallel Process.*, Sep. 2012, pp. 430–439.
[31] S. Albers, A. Antoniadis, and G. Greiner, "On multi-processor speed scaling with migration," in *Proc. 23rd ACM Symp. Parallelism Algorithms Architectures*, 2011, pp. 279–288.
[32] E. Angel, E. Bampis, F. Kacem, and D. Letsios, "Speed scaling on parallel processors with migration," in *Proc. 18th Int. Conf. Parallel Process.*, 2012, pp. 128–140.
[33] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *Des., Autom. Test Eur. Conf. Exhibi.*, 2011, pp. 1–6.
[34] S. Boyd and L. Vandenberghe, in *Convex Optimization*, Cambridge, UK, Cambridg. Univ. Press, 2004.
[35] *MATLAB, Version 7.9.0.529 (R2009b)*, The MathWorks Inc., Natick, MA, USA, 2010.
[36] Intel xscale microarchitecture. (2003). [Online]. Available: http://developer.intel.com/design/intelxscale/benchmarks.htm



Dawei Li received the bachelor's degree from the Advanced Class, Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, Hubei, P.R. China. He is working toward the PhD degree in the Department of Computer and Information Sciences at Temple University since September 2011. His research interests include energy-aware task scheduling on multi-cores/multiprocessors, application mapping and scheduling on network-on-chip -based multiprocessor system-on-chips, and design and analysis of data center networks.

Jie Wu is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, he was a program director at the National Science Foundation and an distinguished professor at Florida Atlantic University. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including *IEEE Transactions on Computers*, *IEEE Transactions on Service Computing*, and *Journal of Parallel and Distributed Computing*. He was a general chair for IEEE IPDPS 2008 and IEEE ICDCS 2013 and a program co-chair/chair for IEEE INFOCOM 2011 and CCF CNCC 2013. Currently, he is serving as a general chair for ACM MobiHoc 2014. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a CCF Distinguished Speaker and a fellow of the IEEE.