Energy-aware Scheduling for Frame-based Tasks on Heterogeneous Multiprocessor Platforms

Dawei Li and Jie Wu
Department of Computer and Information Sciences
Temple University
Philadelphia, USA
{dawei.li, jiewu}@temple.edu

***Abstract*—Modern computational systems have adopted heterogeneous multiprocessors to increase their computation capability. As the performance increases, the energy consumption in these systems also increases significantly. Dynamic Voltage and Frequency Scaling (DVFS), which allows processors to dynamically adjust their supply voltages and execution frequencies to work on different power/energy levels, is considered an efficient scheme to achieve the goal of saving energy. In this paper, we consider scheduling 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 all of these three platforms, we first introduce a Relaxation-based Naive Rounding Algorithm (RNRA), which can produce good solutions for some cases, but may be unstable under other situations. Then, we propose a Relaxation-based Iterative Rounding Algorithm (RIRA). Experiments and comparisons show that our RIRA produces a better performance than RNRA and other existing methods, and achieves near-optimal scheduling under most cases.**

***Index Terms*—Heterogeneous multiprocessor platforms, dynamic voltage and frequency scaling (DVFS), task partitioning, energy-aware scheduling**

I. Introduction

A. Background

To meet the increasing requirements of computation, modern computational systems are adopting multiprocessor platforms. As the computational performance increases, energy consumption in these systems also increases significantly. Dynamic Voltage and Frequency Scaling (DVFS) [1], 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.

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.

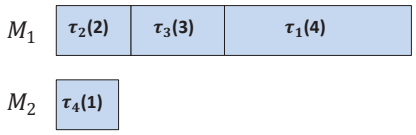
In this paper, we consider partition-based energy-aware scheduling for frame-based tasks on heterogeneous DVFS multiprocessor platforms. For the same problem, widely used methods derive an "energy-efficient" partition that tries to achieve balanced workloads among all processors. It is believed that a balanced partition also has a good performance in terms of overall energy consumption. For example, in [2], the min-min heuristic is considered an energy-aware method for mapping tasks on heterogeneous platforms; in [3], 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 [2], 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 [4], [3], [5].

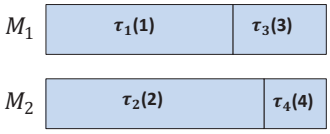
Max-min: this heuristic is very similar to the min-min heuristic. It also begins with the set of all unassigned tasks. The only difference is that, in the second phase, the task with the overall maximum expected completion time among all of the unassigned tasks is chosen and assigned to the corresponding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

B. Motivational Example

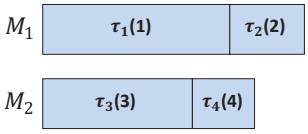
In the following, we will show that workload-balanced partitioning methods do not work well on heterogeneous platforms in terms of overall energy consumption. Due to differences between tasks' characteristics and between processors, different processors may have different execution efficiencies for different 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 tasks and two processors, where $t_{1,1} = 30$, $t_{1,2} = 50$, $t_{2,1} = 12$, $t_{2,2} = 35$, $t_{3,1} = 15$, $t_{3,2} = 24$, $t_{4,1} = 12$,



(a) Partitioned by the min-min heuristic



(b) Partitioned by the max-min heuristic



(c) Partitioned by our RIRA approach

Fig. 1. Task partition using different methods

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 Fig. 1 (a) and Fig. 1 (b), respectively. Our proposed Relaxation-based Iterative Rounding Algorithm (RIRA) always places overall energy consumption at the highest priority and achieves the partition in Fig. 1 (c). In each figure in Fig. 1, the number behind a task is the order in which this task is assigned; for example, $\tau_1(4)$ in Fig. 1 (a) means that τ_1 is the fourth task that is assigned.

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 $e = f^3 t$. For the partition in Fig. 1 (a), which is on a dependent platform without runtime adjusting, both of the two processors should operate at $0.57 f_s$. The overall energy consumption on the two processors is $21.7683 f_s^3$. By similar calculations, the overall energy consumption for the three different partitions on the three types of platforms can be achieved and are listed in Table I, in which "Type I" represents dependent platforms without runtime adjusting, "Type II" represents dependent platforms with runtime adjusting, and "Type III" represents independent platforms. Readers might want to refer to Section II-B for clear definitions of these platform types. 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, however our RIRA will produce different optimal partitions (in terms of overall energy consumption) for dependent platforms and independent platforms, respectively.

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

TABLE I
THE OVERALL ENERGY CONSUMPTION OF DIFFERENT PARTITIONS ON DIFFERENT PLATFORMS

C. 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.

- Firstly, 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, different instruction set architectures, etc. On the other hand, different tasks/applications may have various different characteristics. Thus, this general assumption is practical on real platforms.
- Secondly, most previous work derives 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 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.
- Finally, we propose a Relaxation-based Iterative Rounding Algorithm (RIRA) for partitioning task sets 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. Besides, we believe this "iterative rounding" technique also has its merits when we come to various similar integer, especially binary integer, programming problems.

D. Paper Organization

The organization of this paper is as follows. Section II gives the system model and problem definition; some existing methods for the same problem are also presented; besides, the main idea of our approach for the problem is briefly introduced. In Section III, our proposed approach is applied to scheduling frame-based tasks on dependent platforms; so-

lutions for dependent platforms without and with runtime adjusting are provided in subsections A and B, respectively. Section IV applies our approach to independent platforms. Experiments and comparisons are provided in Section V. A brief conclusion is made in Section VI.

II. System Model and Problem Definition

In this section, we describe the system model and the problem we are considering.

A. Task Model

In this paper, we consider scheduling a set of independent frame-based tasks $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ 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. 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, can be calculated as $C_i/(\lambda f)$.

B. Platform Model

The platform under consideration consists of m heterogeneous processors. All processors are DVFS-enabled processors that can adjust their supply voltages and execution frequencies. We define $\lambda_{i,j}$ as the execution efficiency of processor M_j when it is executing task τ_i. This platform appears in various practical situations, 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 $[0, +\infty)$. The power consumption model that we consider in this paper is widely adopted by existing work and is said to be a good approximation for some practical platforms. Processors can operate in two modes: one is *run* mode, where the power consumption only consists of dynamic power $p = f^3$; 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 without any overhead.

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*.

C. Problem Definition

Given a set of frame-based tasks $\{\tau_1, \tau_2, \cdots, \tau_n\}$, our goal is to schedule all of the tasks on m heterogeneous processors, $M_1, M_2, \cdots, M_m$, such that the overall energy consumption 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 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.

D. Related Work

Much work has been done regarding energy-aware scheduling on both homogeneous platforms and heterogeneous platforms [6], [7].

For homogeneous platforms, several typical works are described as follows. [8] 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. [9] considers scheduling on independent platforms with the consideration of task migration. [10] studies scheduling with the consideration of application-specific power consumption on independent platforms. [11] handles scheduling on partitioned multi-core platforms, where cores within the same partition are dependent and cores from different partitions are independent.

Also, quite a few work has been done for energy-aware scheduling on heterogeneous multiprocessor platforms. In [2], 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. [12] considers energy-aware scheduling on a heterogeneous platform with one non-DVFS Processing Unit (PU) and one DVFS processor. [13] studies platforms with a fixed number of heterogeneous processors. [14] investigates platforms with a fixed number of heterogeneous processor types, while one processor type may still have multiple processors. A Minimum Average Energy First (MAEF) strategy is adopted to tackle the task assignment and processor allocation problem. [15] considers scheduling precedence constrained tasks/applications.

Our work in this paper contributes to energy-aware scheduling on heterogeneous multiprocessor platforms. The main difference between these above-mentioned work and ours is, that our proposed method has a strong theoretical foundation

to produce energy-efficient scheduling, as we will show later. Various experiments and comparisons verify the strength of our approach.

E. 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 (RIRA). The main idea of our RIRA is as follows.

Firstly, we define the average execution cycle of task τ_i as $AEC_i = \frac{1}{m}\sum_{j=1}^{m} C_i/(\lambda_{i,j})$ and sort the tasks in the order $\tau_{i_1}, \tau_{i_2}, \cdots, \tau_{i_n}$, 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.

Secondly, 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.

III. Energy-aware Scheduling on Heterogeneous Dependent Multiprocessor Platforms

In this section, we address the problem of scheduling frame-based tasks on dependent 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 subsections A and B, respectively.

A. Dependent Platforms without Runtime Adjusting

We first consider the optimal frequency setting if we have already had a task 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 $\frac{1}{f}\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}}$. 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:

$$\frac{1}{f}\sum_{i=1}^{n}\frac{x_{i,j}C_i}{\lambda_{i,j}} \leq D, \forall j = 1, 2, \cdots, m.$$

The energy consumption on the jth processor M_j is:

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

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

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

where the optimization variables are the shared frequency f and the binary matrix $x_{n\times m}$. Relax the binary variables $x_{i,j}$'s to be any fraction in $[0, 1]$. The above optimization problem can be reformulated as:

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

Denote this relaxed optimization problem by P_1, which is a convex optimization problem that can be solved by the well-known interior point method in polynomial time (in terms of the input problem size under a given precision requirement) [16]. The optimization variables of P_1 are the shared frequency f and the relaxed assignment matrix $x_{n\times m}$. Here, $x_{i,j}$ represents the percentage of task τ_i that should be assigned to processor M_j, to achieve the minimal overall energy consumption.

Our 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 task τ_i to processor M_{j^*}, such that x_{i,j^*} is the maximum among $x_{i,1}, x_{i,2}, \cdots, x_{i,m}$. Algorithm 1 describes this Relaxation-based Naive Rounding Algorithm (RNRA).

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;

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

However, this approach may lead to an accumulated error between the final assignment and the optimal solution because the condition for optimal energy consumption changes after we assign 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 (RIRA). 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., $AEC_1 \geq AEC_2 \geq \cdots \geq AEC_n$.

Then, relax the original optimization problem as problem P_1. Since tasks are already in our desired order, the solutions $x_{1,1}, x_{1,2}, \cdots, x_{1,n}$ indicate the optimal assignment of the most influential task τ_1. Then, we find the maximum among $x_{1,1}, x_{1,2}, \cdots, x_{1,n}$, 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:

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

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, $x_{1,1}, x_{1,2}, \cdots, x_{1,m}$ 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 $x_{2,1}, x_{2,2}, \cdots, x_{2,m}$, $x_{3,1}, x_{3,2}, \cdots, x_{3,m}$, $\cdots$, $x_{n,1}, x_{n,2}, \cdots, x_{n,m}$. After solving P_2, we can assign τ_2 according to $x_{2,1}, x_{2,2}, \cdots, x_{2,m}$ (solved for P_2), which is similar to what we do to assign τ_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;$\cdots$; solve P_i to assign τ_i; $\cdots$. 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 (RIRA).

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;

Initialize the the assignment matrix: $Assign_{i,j} = 0, \forall i = 1, 2, \cdots, n; j = 1, 2, \cdots, m$;
for $i := 1$ *to* $n-1$ **do**
  Solve Optimization problem P_i;
  $x_{i,j^*} = max(x_{i,1}, x_{i,2}, \cdots, x_{i,m})$;
  for $j := 1$ *to* m **do**
    $x_{i,j} = 0$;
  end for
  $Assign_{i,j^*} = 1$;
  $x_{i,j^*} = 1$;
  Update the optimization problem to be P_{i+1};
end for
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 $Assign_{n,j'} = 1$;
return $Assign$;

An illustrative example is provided, which considers assigning 8 tasks to 3 processors. Tasks' WCECs, $C_1, C_2, \cdots, C_8$ and the processor efficiency matrix, $\lambda_{8\times 3}$ are given in Table II. For the need of some other techniques, a reference time matrix is derived as $t_{8\times 3}$, where $t_{i,j} = C_i/\lambda_{i,j}$, which is also provided in the same table. Notice that, in this example, tasks are already in the required order for RIRA.

Fig. 2 (a) shows the partition by the min-min heuristic. Fig. 2 (b) shows the partition by the max-min heuristic. The RNRA solves P_1 (for this example) and gets the relaxed

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 II
EXAMPLE

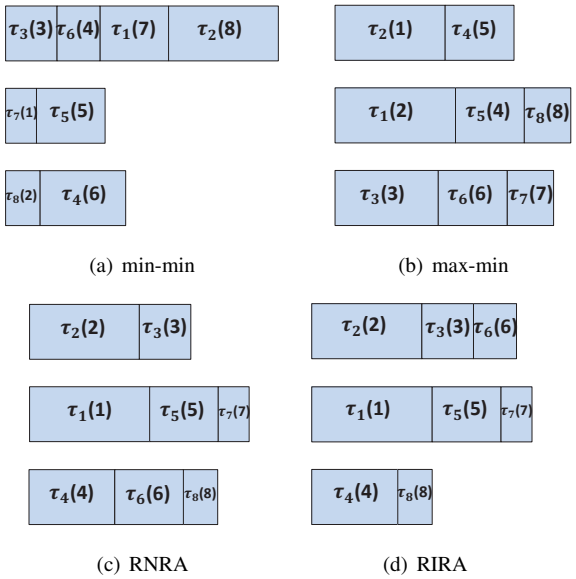
i	Relaxed Assignment Matrix $x_{8\times3}$			$Assign_{8\times3}$		
	$j=1$	$j=2$	$j=3$	$j=1$	$j=2$	$j=3$
1	0.2920	0.7080	0	0	1	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	0	1	0	0	1
5	0	1	0	0	1	0
6	0.0665	0	0.9335	0	0	1
7	0	1	0	0	1	0
8	0	0	1	0	0	1

TABLE III
ASSIGNMENT BY RNRA

assignment matrix $x_{8\times3}$, as shown in Table III. Obviously, by the naive rounding scheme, it can easily achieve the final assignment matrix $Assign_{8\times3}$. Fig. 2 (c) 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 7 tasks; for the last task, it selects the assignment that achieves the minimal energy consumption. Relevant solutions are shown in Table IV. Fig. 2 (d) shows the partition by our proposed RIRA. In each figure in Fig. 2, the number behind a task is the order in which this task is assigned; for example, $\tau_7(1)$ in Fig. 2 (a) 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 exactly. Assume that the common deadline is 100. The energy consumption for these four partitions are 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.

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

TABLE IV
ASSIGNMENT BY RIRA



(a) min-min (b) max-min

(c) RNRA (d) RIRA

Fig. 2. Partitions by different approaches

B. Dependent Platforms with Runtime Adjusting

For dependent platforms with runtime adjusting, an approach similar to that in [8] is applied here to determine the optimal frequency scheduling during different time 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 $U_j = \sum_{i=1}^{n} \frac{x_{i,j}C_i}{\lambda_{i,j}}$. Without loss of generality, we assume that processors are in ascending order of their U_j values, i.e., $U_1 \leq U_2 \leq \cdots \leq U_m$. Since all of the processors must share a common frequency (though the shared frequency can vary with time), the processor with a lesser U_j value will complete its tasks earlier than that with a greater U_j value. Introduce $U_0 = 0$; the time interval between the time when U_{j-1} is completed and the time when U_j is completed, can be calculated as $t_j = (U_j - U_{j-1})/f_j$, where f_j is the shared frequency of the running processors during this time interval. Thus, the energy consumption during the jth time interval is:

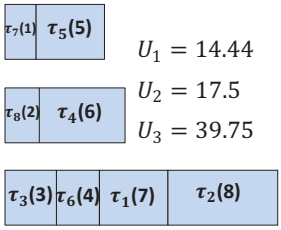
$$E_j^{'} = (m - j + 1) f_j^2 (U_j - U_{j-1})$$

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

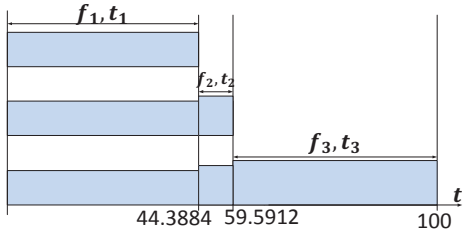
$$\begin{aligned} min \quad & E_{total} = \textstyle\sum_{j=1}^{m} (m - j + 1) f_j^2 (U_j - U_{j-1}) \\ s.t. \quad & \textstyle\sum_{j=1}^{m} \frac{U_j - U_{j-1}}{f_j} \leq D \\ & 0 \leq \frac{U_j - U_{j-1}}{f_j} \leq D, f_j \geq 0, \forall j = 1, 2, \cdots, m. \end{aligned}$$

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:

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



(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

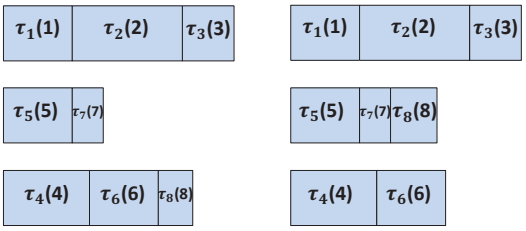
Let's take the partition by the min-min heuristic in Fig. 2 (a) as an illustrative example. The runtime frequency adjusting procedures are shown in Fig. 3. Fig. 3 (a) shows the sorted workloads among the three processors. After this, we can determine the optimal frequencies for the three time intervals. 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 overall energy consumption is reduced from 11.3 to 10.3375.

After applying this runtime adjusting scheme for all of the 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.

IV. Energy-aware Scheduling on Heterogeneous Independent Multiprocessor Platforms

In this section, we will apply our approaches for energy-aware scheduling frame-based tasks on independent multi-processor platforms. Again, we firstly 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:

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



(a) RNRA (b) RIRA

Fig. 4. Partitions by RNRA and RIRA

Then, the energy consumption on the jth processor M_j is:

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

Thus, to achieve the energy-optimal partition, it is equivalent to solve the following optimization problem, denoted by $P_1^{'}$:

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

For simplicity, the binary optimization variables $x_{i,j}$'s have already been relaxed. RNRA solves problem $P_1^{'}$, and adopts the same process as in Algorithm I to assign all of the tasks. Our RIRA solves problem $P_1^{'}$ first, and assigns task τ_1 according to solutions $x_{1,1}, x_{1,2}, \cdots, x_{1,m}$; then, it updates the optimization problem as $P_2^{'}$:

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

Assign task τ_2 according to the solutions $x_{2,1}, x_{2,2}, \cdots, x_{2,m}$ (solved for $P_2^{'}$). Notice that the optimization variables of $P_2^{'}$ only includes $x_{2,1}, x_{2,2}, \cdots, x_{2,m}$, $x_{3,1}, x_{3,2}, \cdots, x_{3,m}$, $\cdots$, $x_{n,1}, x_{n,2}, \cdots, x_{n,m}$, since τ_1 has already been assigned; in other words, $x_{1,1}, x_{1,2}, \cdots, x_{1,m}$ 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.

For the example in Table II, obviously, the min-min and max-min heuristics will produce the same partitions as in Fig. 2 (a) and Fig. 2 (b). However, RNRA and RIRA will produce partitions different from those in Fig. 2 (c) and Fig. 2 (d). Using the same example on independent platforms, the partitions derived by RNRA and RIRA are shown in Fig. 4 (a) and Fig. 4 (b), respectively. 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.

V. Performance Evaluation

In this section, we will evaluate our RIRA from three different angles under three settings. 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 $P_1^{'}$. All experiments and comparisons are done for scheduling 24 tasks on 6 processors.

A. Simulation Settings

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 $\lambda_{i,j}(i = 1, 2, \cdots, 24; j = 1, 2, \cdots, 6)$ values are uniformly distributed in $[0.1, 1]$. 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 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: $\lambda_{i,1} = 1$, $\lambda_{i,2} = 0.82$, $\lambda_{i,3} = 0.64$, $\lambda_{i,4} = 0.46$, $\lambda_{i,5} = 0.28$, $\lambda_{i,6} = 0.1$, $\forall i = 1, 2, \cdots, 24$. 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 consumption 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 consumption.

B. Simulation Results

Results for the first setting is provided in Fig. 5. In each figure of 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% greater than the optimal energy consumption. For this setting, the normalized energy consumption of RIRA achieves the smallest standard deviation. Thus, our RIRA is better than all of the other three methods, in terms of both average performance and stability.

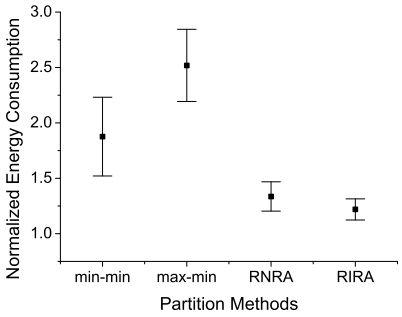
Results for the second setting is provided in Fig. 6. In each figure of Fig. 6, the vertical axis represents the normalized energy consumption for 50 randomly generated task sets. We notice that, in this special case, the RNRA attempts to assign all of the tasks to the most efficient processor, i.e., processor M_1. Thus, the normalized energy consumption of RNRA (about 6) is much greater than the other three methods (less than 2). 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% 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 better performance than max-min in the first setting, while max-min achieves 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.

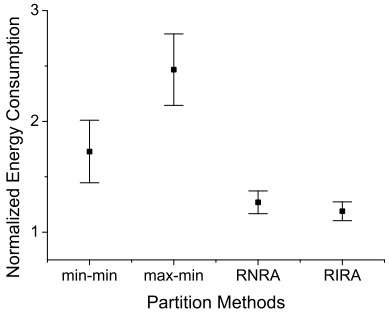
Results for the third setting are shown in Fig. 7. In each type of platforms in Fig. 7, the horizontal axis represents the 20 randomly generated processor efficiency matrices. The vertical axis represents the average normalized energy consumption of the 20 randomly generated sets of tasks, given a processor 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.

C. Simulation Summary

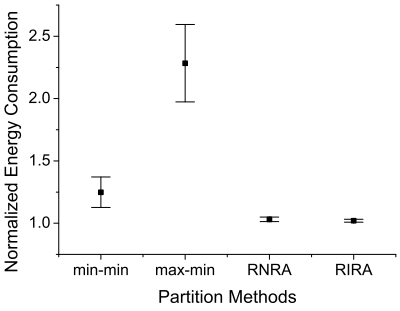
From all of the above experiments and comparisons, we can see that our proposed RIRA method outperforms all other methods in terms of overall energy consumption and achieves a near optimal solution for various situations, especially on independent platforms. Our RIRA achieves a good performance mainly because of the two techniques it applies. The first technique is the ranking scheme, namely, considering



(a) Dependent Platform without Runtime Adjusting

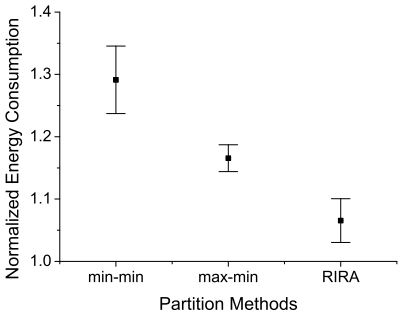


(b) Dependent Platform with Runtime Adjusting

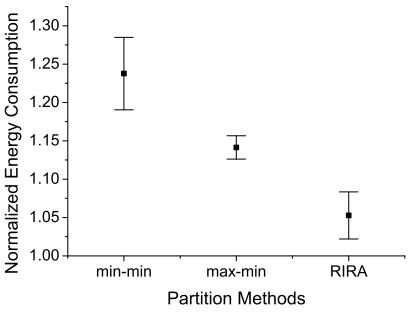


(c) Independent Platform

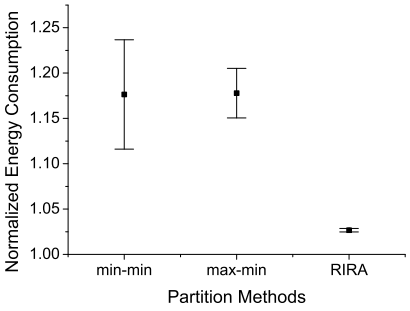
Fig. 5. Setting I: given task set



(a) Dependent Platform without Runtime Adjusting



(b) Dependent Platform with Runtime Adjusting



(c) Independent Platform

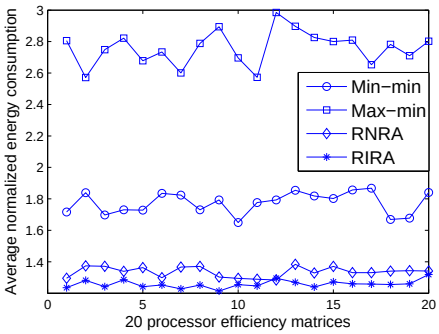
Fig. 6. Setting II: given processor efficiency matrix $\lambda_{24\times 6}$

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.

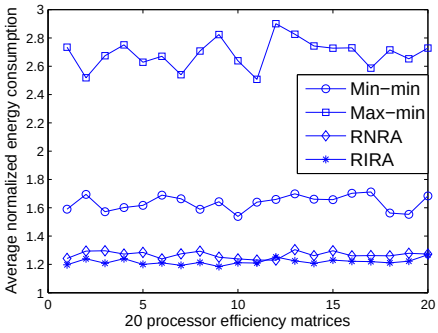
D. Additional Remarks

One important aspect we should notice is that, though RIRA requires high computational complexity compared to other methods, it is still a polynomial time algorithm, for the interior point method is in polynomial time in terms of input problem size, given a specific solution accuracy [16]. Besides, in our RIRA, we actually do not need 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 reduce the accuracy requirement when solving the relaxed optimization problems to reduce its computational complexity. Thirdly, it should also be noticed that our scheduling 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. These aspects make RIRA a practically-applicable scheduling scheme.

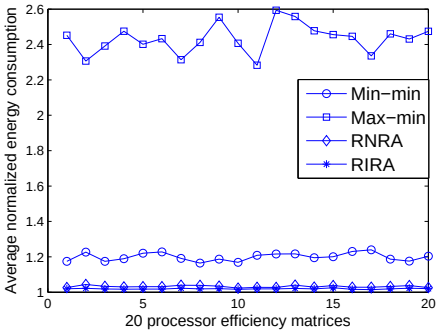
Besides, in our work, we assume that processors' dynamic power consumption is $p = f^3$; our approach is not limited by this assumption. Actually, our proposed RIRA works well for the general assumption that the power consumption of processor M_j is $p_j = \beta_j f^\alpha$, where β_j is a positive constant factor for processor M_j, and α is a positive number greater



(a) Dependent Platform without Runtime Adjusting



(b) Dependent Platform with Runtime Adjusting



(c) Independent Platform

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

than 2. Additionally, 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. This aspect is what we will consider in the future.

Finally, as we have pointed out, this 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.

VI. Conclusion

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 (RIRA) 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 near optimal solution for most cases. Experiments and comparisons from three different angles verify the strength of our algorithm.

References

[1] 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 *IEEE/ACM International Conference on Computer Aided Design*.

[2] W. Sun and T. Sugawara, "Heuristics and evaluations of energy-aware task mapping on heterogeneous multiprocessors," in *International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, May 2011, pp. 599 –607.

[3] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," in *the 3rd IEEE/IFIP International Conference in Central Asia on Internet*, September 2007, pp. 1 –7.

[4] 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, November 1999.

[5] H. Izakian, A. Abraham, and V. Snasel, "Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments," in *International Joint Conference on Computational Sciences and Optimization*, vol. 1, April 2009, pp. 8–12.

[6] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in *the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2007, pp. 28–38.

[7] J.-J. Chen, C.-Y. Yang, T.-W. Kuo, and C.-S. Shih, "Energy-efficient real-time task scheduling in multiprocessor dvs systems," in *Asia and South Pacific Design Automation Conference*, January 2007, pp. 342–349.

[8] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *Proceedings of Design, Automation and Test in Europe*, March 2005, pp. 468–473 Vol. 1.

[9] 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 *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, June-July 2004, pp. 101 – 108.

[10] J.-J. Chen and T.-W. Kuo, "Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics," in *International Conference on Parallel Processing*, June 2005, pp. 13–20.

[11] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *Design, Automation Test in Europe Conference and Exhibition*, March 2011, pp. 1–6.

[12] 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 International Real-Time Systems Symposium*, December 2006, pp. 303–312.

[13] 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 *Design, Automation Test in Europe Conference and Exhibition*, April 2009, pp. 694–699.

[14] J.-J. Chen and L. Thiele, "Task partitioning and platform synthesis for energy efficiency," in *the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009, pp. 393–402.

[15] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2009, pp. 92 –99.

[16] S. Boyd and L. Vandenberghe, "Convex optimization," in *Cambridge University Press*, 2004.