Energy-Efficient Scheduling with Time and Processors Eligibility Restrictions

Xibo Jin[1,2], Fa Zhang[1], Ying Song[1], Liya Fan[3], and Zhiyong Liu[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] IBM China Research Laboratory, Beijing, China
{jinxibo,zhangfa,songying,zyliu}@ict.ac.cn, fanliya@cn.ibm.com

Abstract. While previous work on energy-efficient algorithms focused on assumption that tasks can be assigned to any processor, we initially study the problem of task scheduling on restricted parallel processors. The objective is to minimize the overall energy consumption while speed scaling (SS) method is used to reduce energy consumption under the execution time constraint (Makespan C_{max}). In this work, we discuss the speed setting in the continuous model that processors can run at arbitrary speed in $[s_{min}, s_{max}]$. The energy-efficient scheduling problem, involving task assignment and speed scaling, is inherently complicated as it is proved to be NP-Complete. We formulate the problem as an Integer Programming (IP) problem. Specifically, we devise a polynomial time optimal scheduling algorithm for the case tasks have an uniform size. Our algorithm runs in $O(mn^3logn)$ time, where m is the number of processors and n is the number of tasks. We then present a polynomial time algorithm that achieves an approximation factor of $2^{\alpha-1}(2-\frac{1}{m^{\alpha}})$ (α is the power parameter) when the tasks have arbitrary size work.

1 Introduction

Energy consumption has become an important issue in the parallel processor computational systems. Dynamic Speed Scaling (SS) is a popular approach for energy-efficient scheduling to reduce energy consumption by dynamically changing the speeds of the processors according to the work they need to perform. The well-known relationship between speed and power is the cube-root rule, more precisely, that is the power of a processor is proportional to s^3 when it runs at speed s [1, 2]. Most research literatures [3, 4, 5, 6, 7, 8, 9, 10] have assumed a more general power function s^{α}, where $\alpha > 1$ is a constant power parameter. Note that it is a convex function of the processor's speed. Obviously, energy consumption is the power integrated over duration time. Higher speeds allow for faster execution, at the same time, result in higher energy consumption. In the past few years, energy-efficient scheduling has received much attention from single processor to parallel processors environment. In algorithmic, the approaches can (in general) be categorized into the following two classes for reducing the energy usage [5, 7]: (1) *Dynamic speed scaling* and (2) *Power-down management*. Our paper focuses on energy-efficient scheduling via dynamic speed scaling strategy.

F. Wolf, B. Mohr, and D. an Mey (Eds.): Euro-Par 2013, LNCS 8097, pp. 66–77, 2013.

In this policy, the goals of scheduling are either to minimize the total energy consumption or to trade off the conflicting objectives of energy and performance. The main difference is the former reduces the total energy consumption as long as the timing constraint is not violated, while the later seeks the best point between the energy cost and performance metric (such as makespan and flow time).

Speed scaling has been widely studied to save energy consumption initiated by Yao et al. [3]. The previous work consider that a task can be assigned to any processor. But it is natural to consider the restricted scheduling in modern computational systems. The reason is that the systems evolve over time, such as cluster, then the processors of the system differ from each other in their functionality (For instance, the processors have different additional components). This leads to the task can only be assigned to the processors, which has the task's required component. I.e., it leads to different affinities between tasks and processors. In practice, certain tasks may have to be allocated for certain physical resources (such as GPU) [11]. It is also pointed out that some processors whose design is specialized for particular types of tasks, then tasks should be assigned to a processor best suited for them [12]. Furthermore, when considering tasks and input data, tasks need to be assigned on the processors containing their input data. In other words, a part of tasks can be assigned on processors set A_i, and a part of tasks can be assigned on processors set A_j, but $A_i \neq A_j$, $A_i \cap A_j \neq \emptyset$. Another case in point is the scheduling with processing processor restrictions aimed at minimizing the makespan has been studied extensively in algorithmic (See [13] for an excellent survey). Therefore, it is significant to study the scheduling with processor restrictions from both of practical and algorithmic requirements.

Previous Work: Yao et al. [3] were the first to explore the problem of scheduling a set of tasks with the smallest amount of energy on single processor environment via speed scaling. They proposed an optimal offline greedy algorithm and two bounded online algorithms named *Optimal Available* and *Average Rate*. Ishihara et al. [4] formulated the minimization-energy of dynamical voltage scheduling (DVS) as an integer linear programming problem when all tasks were ready at the beginning and shared common finishing time. They showed that in the optimal solution a processor only runs at two adjacent discrete speeds when it can use only a small number of discrete processor speeds.

Besides studying variant of the speed scaling problems on single processor, researchers also carried out studies on parallel processors environment. Chen et al. [6] considered energy-efficient scheduling with and without task migration over multiprocessor. They proposed approximation algorithm for different settings of power characteristics where no task was allowed to migrate. When task migration is allowed and migration cost is assumed being negligible, they showed that there is an optimal real-time task scheduling algorithm. Albers et al. [7] investigated the basic problem of scheduling a set of tasks on multi-processor settings with an aim to minimize the total energy consumption. First they studied the case that all tasks were unit size and proposed a polynomial time algorithm for agreeable deadlines. They proved it is NP-Hard for arbitrary release time and deadlines

and gave a $\alpha^{\alpha}2^{4\alpha}$-approximation algorithm. For scheduling tasks with arbitrary processing size, they developed constant factor approximation algorithms. Aupy et al. [2] studied the minimization of energy on a set of processors for which the tasks assignment had been given. They investigated different speed scaling models. Angel et al. [10] consider the multiprocessor migratory and preemptive scheduling problem with the objective of minimizing the energy consumption. They proposed an optimal algorithm in the case where the jobs have release dates, deadlines and the power parameter $\alpha > 2$.

There were also some literatures to research the performance under an energy bounded. Pruhs et al. [8] discussed the problem of speed scaling to optimize makespan under an energy budget in a multiprocessor environment where the tasks had precedence constraints ($Pm|prec, energy|C_{max}$, m is the number of processors). They reduced the problem to the $Qm|prec|C_{max}$ and obtained a poly-log(m)-approximation algorithm assuming processors can change speed continuously over time. Greiner et al. [9] studied the trade off between energy and delay, i.e., their objective was to minimize the sum of energy cost and delay cost. They suggested a randomized algorithm $\mathcal{RA}$ for multiple processors: each task was assigned uniformly at random to the processors, then a single processor algorithm $\mathcal{A}$ was applied separately by each processor. They proved that the approximation factor of $\mathcal{RA}$ was βB_{α} without task migration when $\mathcal{A}$ was a β-approximation algorithm (B_{α} is the α-th Bell number). They also showed that any β-competitive online algorithm for a single processor yields a randomized βB_{α}-competitive online algorithm for multiple processors without migration. Using the method of conditional expectations, the results could be transformed to a derandomized version with additional running time. Angel et al. [10] also extended their algorithm, which considered minimizing the energy consumption, to obtain an optimal algorithm for the problem of maximum lateness minimization under a budget of energy.

However, all of these results were established without taking into account the restricted parallel processors. More formally, let the set of tasks $\mathcal{J}$ and the set of processors $\mathcal{P}$ construct a bipartite graph $G = (\mathcal{J} + \mathcal{P}, E)$, where the edge of E denotes a task can be assigned to a processor. The previous work study G is a *complete* bipartite graph, i.e., for any two vertices, $v_1 \in \mathcal{J}$ and $v_2 \in \mathcal{P}$, the edge v_1v_2 is in G. We study the energy-efficient scheduling that G is a *general* bipartite graph, i.e., v_1v_2 may be not an edge of G.

Our Contribution: In this paper, we address the problem of task **S**cheduling with the objective of **E**nergy **M**inimization on **R**estricted **P**arallel **P**rocessors (SEMRPP). It assumes all tasks are ready at time 0 and share a common deadline (a real-time constraint) [2, 4, 6, 7]. In this work, We discuss the continuous speed settings that processors can run at arbitrary speed in $[s_{min}, s_{max}]$. In Section 2, we provide the formal description of model. Section 3 discusses some preliminary results and reformulate the problem as an Integer Programming (IP) problem. In Section 4, we devise a polynomial time optimal scheduling algorithm in the

case where the tasks have an uniform size. For the general case that the tasks have non-uniform computational work, in Section 5, we present a $2^{\alpha-1}(2-\frac{1}{m^{\alpha}})$-approximation algorithm, where α is the power parameter and m is the number of processors. Finally we conclude the paper in Section 6. To the best of our knowledge, our work may be the initial attempt to study energy optimization on the restricted parallel processors.

2 Problem and Model

We model the SEMRPP problem of scheduling a set $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ of n independent tasks on a set $\mathcal{P} = \{P_1, P_2, ..., P_m\}$ of m processors. Each task J_j has an amount of computational work w_j which is defined as the number of the required CPU cycles for the execution of J_j [3]. We refer to the set $\mathcal{M}_j \subseteq \mathcal{P}$ as eligibility processing set of the task J_j, that is, J_j needs to be scheduled on one of its eligible processors $\mathcal{M}_j(\mathcal{M}_j \neq \phi)$. We also say that J_j is allowable on processor $P_i \in \mathcal{M}_j$, and is not allowed to migrate after it is assigned on a processor. A processor can process at most one task at a time and all processors are available at time 0.

At any time t, the speed of J_j is denoted as s_{jt}, and the corresponding processing power is $P_{jt} = (s_{jt})^{\alpha}$. The amount of CPU cycles w_j executed in a time interval is the speed integrated over duration time and energy consumption E_j is the power integrated over duration time, that is, $w_j = \int s_{jt}dt$ and $E_j = \int P_{jt}dt$, following the classical models of the literature [2, 3, 4, 5, 6, 7, 8, 9, 10]. Note that in this work we focus on speed scaling and all processors are alive during the whole execution, so we do not take static energy into account [2, 8]. Let c_j be the time when the task J_j finishes its execution. Let x_{ij} be an $0-1$ variable which is equal to one if the task J_j is processed on processor P_i and zero otherwise. We note that $x_{ij} = 0$ if $P_i \notin \mathcal{M}_j$. Our goal is to schedule the tasks on processors to minimize the overall energy consumption when each task could finish before the given common deadline C and be processed on its eligible processors. Then the SEMRPP problem is formulated as follows:

$$(\mathbf{P_0}) \qquad min \sum_{j=1}^{n} \int P_{jt}dt$$

$$s.t. \qquad c_j \leq C \quad \forall J_j,$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall J_j,$$

$$x_{ij} \in \{0,1\} \quad \forall J_j, P_i \in \mathcal{M}_j,$$

$$x_{ij} = 0 \quad \forall J_j, P_i \notin \mathcal{M}_j.$$

3 Preliminary Lemma

We start by giving preliminary lemmas for reformulating the SEMRPP problem.

Lemma 1. *If S is an optimal schedule for the SEMRPP problem in the continuous model, it is optimal to execute each task at a unique speed throughout its execution.*

Proof. Suppose S is an optimal schedule that some task J_j does not run at a unique speed during its execution. We denote J_j's speeds by $s_{j1}, s_{j2}, ..., s_{jk}$, the power of each speed i is $(s_{ji})^\alpha, i = (1, 2, ..., k)$, and the execution time of the speeds are $t_{j1}, t_{j2}, ..., t_{jk}$, respectively. So, its energy consumption is $\sum_{i=1}^{k} t_{ji}(s_{ji})^\alpha$. We average the k speeds and keep the total execution time unchanged, i.e., $\bar{s}_j = (\sum_{i=1}^{k} s_{ji}t_{ji})/(\sum_{i=1}^{k} t_{ji})$. Because the power function is a convex function of speed, according to convexity [14] (In the rest of paper, it will use convexity in many place but will not add reference [14]), we have

$$\sum_{i=1}^{k} t_{ji}(s_{ji})^\alpha = (\sum_{i=1}^{k} t_{ji})(\sum_{i=1}^{k} \frac{t_{ji}}{\sum_{i=1}^{k} t_{ji}}(s_{ji})^\alpha)$$
$$\geq (\sum_{i=1}^{k} t_{ji})(\sum_{i=1}^{k} \frac{t_{ji}s_{ji}}{\sum_{i=1}^{k} t_{ji}})^\alpha = (\sum_{i=1}^{k} t_{ji})(\bar{s}_j)^\alpha = \sum_{i=1}^{k} t_{ji}(\bar{s}_j)^\alpha$$

So the energy consumption by unique speed is less than a task run at different speeds. I.e. , if we do not change J_j's execution time and its assignment processor (satisfying restriction), we can get a less energy consumption scheduling, which is a contradiction to that S is an optimal schedule.

Corollary 1. *There exists an optimal solution for SEMRPP in the continuous model, for which each processor executes all tasks at a uniform speed, and finishes its tasks at time C.*

All tasks on a processor run at a unique speed can be proved like *Lemma 1.* If some processor finishes its tasks earlier than C, it can lower its speed to consume less energy without breaking the time constraint and the restriction. Furthermore there will be no gaps in the schedule [8].

Above discussion leads to a reformulation of the SEMRPP problem in the continuous model as following:

$$(\mathbf{P_1}) \qquad min \sum_{i=1}^{m} \frac{(\sum_{j=1}^{n} x_{ij}w_j)^\alpha}{C^{\alpha-1}}$$

$$s.t. \qquad \sum_{j=1}^{n} x_{ij}w_j \leq s_{max}C \quad \forall P_i, \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall J_j, \tag{2}$$

$$x_{ij} \in \{0, 1\} \quad \forall J_j, P_i \in \mathcal{M}_j, \tag{3}$$

$$x_{ij} = 0 \quad \forall J_j, P_i \notin \mathcal{M}_j. \tag{4}$$

The objective function is from that a processor P_i runs at speed $\frac{\Sigma_{J_j on P_i} w_j}{C} = \frac{\Sigma_{j=1}^{n} x_{ij} w_j}{C}$, that is each task on P_i will run at this speed, and P_i will complete all the tasks on it at time C (It assumes that, in each problem instance, the computational cycles of the tasks on one processor is enough to hold the processor will not run at speed $s_i < s_{min}$. Otherwise we are like to turn off some processors). Constraint (1) follows since a processor can not run at a speed higher than s_{max}. Constraint (2) relates to that if a task has assigned on a processor it will not be assigned on other processors, i.e, non-migratory. Constraint (3) and (4) are the restrictions of the task on processors.

Lemma 2. *Finding an optimal schedule for SEMRPP problem in the continuous model is NP-Complete in the strong sense.*

Lemma 3. *There exists a polynomial time approximation scheme (PTAS) for SEMRPP problem in the continuous model, when $\mathcal{M}_j = \mathcal{P}$ and s_{max} is fast enough.*

Note that we give detailed proofs (Due to the space limit, we omit the proof. See our report [15] for details) of Lemma 2 and Lemma 3 that were similarly stated as observations in the work [7], and we mainly state the conditions when they are established in the restricted environment. (such as the set of restricted processors and the upper speed s_{max} that we discuss below in the paper)

4 Uniform Tasks

We now propose an optimal algorithm for a special case of SEMRPP problem in which all tasks have equal execution cycles (uniform) (denoted as ECSEMRPP_Algo algorithm). We set $w_j = 1, \forall J_j$ and set $C = C/w_j$ in $(\mathbf{P_1})$ without loss of generality. Given the set of tasks $\mathcal{J}$, the set of processors $\mathcal{P}$ and the sets of eligible processors of tasks $\{\mathcal{M}_j\}$, we construct a network $G = (V, E)$ as follow: the vertex set of G is $V = \mathcal{J} \cup \mathcal{P} \cup \{s, t\}$ (s and t correspond to a source and a destination, respectively), the edge set E of G consists of three subsets: $(1)(s, P_i)$ for all $P_i \in \mathcal{P}$; $(2)(P_i, J_j)$ for $P_i \in \mathcal{M}_j$; $(3)(J_j, t)$ for all $J_j \in \mathcal{J}$. We set unit capacity to edges (P_i, J_j) and (J_j, t), (s, P_i) have capacity c (initially we can set $c = n$). Define $L^* = min\{max\{L_i\}\}(i = 1, 2, ..., m)$, L_i is the load of processor P_i and it can be achieved by ***Algorithm 1***.

Lemma 4. *The algorithm BS_Algo solves the problem of finding minimization of maximal load of processor for restricted parallel processors in $O(n^3 logn)$ time, if all tasks have equal execution cycles.*

Its proof can mainly follow from the Maximum-flow in [16]. The computational complexity is equal to the time $O(n^3)$ to find Maximum-flow multiple $logn$ steps, i.e, $O(n^3 logn)$.

We construct our ECSEMRPP_Algo algorithm (***Algorithm 2***) through finding out the min-max load vector $\boldsymbol{l}$ that is a strongly-optimal assignment defined in [17, 18].

Algorithm 1. BS_Algo(G, n)

input : (G, n)
output: L^*, P_i that have the maximal load, the set $\mathcal{J}_i$ of tasks that load on P_i
1: Let variable $l = 1$ and variable $u = n$;
2: If $l = u$, then the optimal value is reached: $L^* = l$, return the P_i and $\mathcal{J}_i$, stop;
3: Else let capacity $c = \lfloor \frac{1}{2}(l + u) \rfloor$. Find the Maximum-flow in the network G. If the value of Maximum-flow is exact n, namely $L^* \leq c$, then set $u = c$ and keep P_i, $\mathcal{J}_i$ by the means of the Maximum-flow. Otherwise, the value of Maximum-flow is less than n, namely $L^* > c$, we set $l = c + 1$. Go back to 2.

Algorithm 2. ECSEMRPP_Algo

1: Let $G_0 = G(V, E)$, $\mathcal{P}^H = \phi$, $\mathcal{J}^H = \{\phi_1, ..., \phi_m\}$;
2: Call $BS\_Algo(G_0, n)$;
3: Set maximal load sequence index $i = i + 1$. According to the scheduling returned by step 2, we denote the processor that has actual maximal load as P_i^H and denote the tasks set assigned on it as $\mathcal{J}_i^H$. $\mathcal{E}_i^H$ corresponds to the related edges of P_i^H and $\mathcal{J}_i^H$. We set $G_0 = \{V \backslash P_i^H \backslash \mathcal{J}_i^H, E \backslash \mathcal{E}_i^H\}$, $\mathcal{P}^H = \mathcal{P}^H \cup \{P_i^H\}$, $\phi_i = \mathcal{J}_i^H$. If $G_0 \neq \phi$, go to step 2;
4: We assign the tasks of $\mathcal{J}_i^H$ to P_i^H and set all tasks at speed $\frac{\Sigma_{J_j \in \mathcal{J}_i^H} w_j}{C}$ on P_i^H. Return the final schedule H.

Definition 1. *Given an assignment H denote by S_k the total load on the k most load of processors. We say that an assignment is strongly-optimal if for any other assignment H' (S_k' accordingly responds to the total load on the k most load of processors) and for all $1 \leq k \leq m$ we have $S_k \leq S_k'$.*

Theorem 1. *Algorithm ECSEMRPP_Algo finds the optimal schedule for the SEMRPP problem in the continuous model in $O(mn^3 logn)$ time, if all tasks have equal execution cycles.*

Proof. First we prove the return assignment H of ECSEMRPP_Algo is a strongly-optimal assignment. We set $H = \{L_1, L_2, ..., L_m\}$, L_i corresponds to the load of processor P_i in non-ascending order. Suppose H' is another assignment that $H' \neq H$ and $\{L_1', L_2', ..., L_m'\}$ corresponds to the load. According to the ECSEMRPP_Algo algorithm, we know that H' can only be the assignment that P_i moves some tasks to $P_j (j < i)$, because P_i can not move some tasks to $P_{j'} (j' > i)$ otherwise it can lower the L_i which is a contradiction to ECSEMRPP_Algo algorithm. We get $\Sigma_{k=1}^{i} L_i \leq \Sigma_{k=1}^{i} L_i'$, i.e., H is a strongly-optimal assignment by the definition. It turns out that there does not exist any assignment that can reduce the difference between the loads of the processors in the assignment H. I.e., there are not other assignment can reduce our aim as it is convexity. So the optimal scheduling is obtained.

Every time we discard a processor, so the total cost time is $m \times O(n^3 logn) = O(mn^3 logn)$ according to *Lemma* 4, which completes the proof.

5 General Tasks

As it is NP-Complete in the strong sense for general tasks (*Lemma* 2), we aim at getting an approximation algorithm for the SEMRPP problem. First we relax the equality (3) of ($\mathbf{P_1}$) to

$$0 \leq x_{ij} \leq 1 \qquad \forall J_j, P_i \in \mathcal{M}_j \tag{5}$$

After relaxation, the SEMRPP problem transforms to a convex program. The feasibility of the convex program can be checked in polynomial time to within an additive error of ϵ (for an arbitrary constant $\epsilon > 0$) [19], and it can be solved optimally [14]. Suppose x^* be an optimal solution to the relaxed SEMRPP problem. Now our goal is to convert this fractional assignment to an integral one $\bar{x}$. We adopt the dependent rounding introduced by [18, 20].

Define a bipartite graph $G(x^*) = (V, E)$ where the vertices of G are $V = \mathcal{J} \cup \mathcal{P}$ and $e = (i, j) \in E$ if $x^*_{ij} > 0$. The weight on edge (i, j) is $x^*_{ij} w_j$. The rounding iteratively modifies x^*_{ij}, such that at the end x^*_{ij} becomes integral. There are mainly two steps as following:

i. Break cycle:

1.While($G(x^*)$ has cycle $C = (e_1, e_2, ..., e_{2l-1}, e_{2l})$)

2.Set $C_1 = (e_1, e_3, ..., e_{2l-1})$ and $C_2 = (e_2, e_4, ..., e_{2l})$.

Find minimal weight edge of C, denoted as e^C_{min} and its weight $\epsilon = min_{e \in C_1 || e \in C_2} e$;

3.If $e^C_{min} \in C_1$ then every edge in C_1 subtract ϵ and every edge in C_2 add ϵ;

4.Else every edge in C_1 add ϵ and every edge in C_2 subtract ϵ;

5.Remove the edges with weight 0 from G.

ii. Rounding fractional tasks:

1.In the first rounding phase consider each integral assignment if $x^*_{ij} = 1$, set $\bar{x}_{ij} = 1$ and discard the corresponding edge from the graph. Denote again by G the resulting graph;

2.While($G(x^*)$ has connected component C)

3.Choose one task node from C as root to construct a tree Tr, match each task node with any one of its children. The resulting matching covers all task nodes;

4.Match each task to one of its children node (a processor) such that $P_i = argmin_{P_i \in \mathcal{P}} \Sigma_{\bar{x}_{ij}=1} \bar{x}_{ij} w_j$, set $\bar{x}_{ij} = 1$, and $\bar{x}_{ij} = 0$ for other children node respectively.

We denote above algorithm as Relaxation-Dependent-Rounding. Next we analyse the approximation factor it can find.

Theorem 2. *(i) Relaxation-Dependent-Rounding finds a* $2^{\alpha-1}(2 - \frac{1}{p^\alpha})$*-approximation to the optimal schedule for the SEMRPP problem in the continuous model in polynomial time, where* $p = max_{\mathcal{M}_j} |\mathcal{M}_j| \leq m$. *(ii) For any processor* P_i, $\Sigma_{\mathcal{J}} \bar{x}_{ij} w_j < \Sigma_{\mathcal{J}} x^*_{ij} w_j + max_{\mathcal{J}: x^*_{ij} \in (0,1)} w_j$, x^*_{ij} *is the fractional task assignment at the beginning of the second phase. (i.e., extra maximal execution cycles of linear constraints are violated only by* $max_{\mathcal{J}: x^*_{ij} \in (0,1)} w_j$*)*

Proof. (i) Denote the optimal solution for the SEMRPP problem as OPT, H^* as the fractional schedule obtained after breaking all cycles and $\bar{H}$ as the schedule returned by the algorithm. Moreover, denote by H_1 the schedule consisting of the tasks assigned in the first step, i.e., $x^*_{ij} = 1$ right after breaking the cycles and by H_2 the schedule consisting of the tasks assigned in the second rounding step, i.e., set $\bar{x}_{ij} = 1$ by the matching process. We have $\|H_1\|_\alpha \leq \|H^*\|_\alpha \leq \|OPT\|_\alpha$[1], where the first inequality follows from H_1 is a sub-schedule of H^* and the second inequality results from H^* being a fractional optimal schedule compared with OPT which is an integral schedule. We consider $\|H_1\|_\alpha \leq \|H^*\|_\alpha$ carefully. If $\|H_1\|_\alpha = \|H^*\|_\alpha$, that is all tasks have been assigned in the first step and the second rounding step is not necessary, then we have $\|H_1\|_\alpha = \|H^*\|_\alpha = \|OPT\|_\alpha$. Such that the approximation is 1. Next we consider $\|H_1\|_\alpha < \|H^*\|_\alpha$, so there are some tasks assigned in the second rounding step, w.l.o.g., denote as $\mathcal{J}_1 = \{J_1, ..., J_k\}$. We assume the fraction of task J_j assigned on processor P_i is f_{ij} and the largest eligible processor set size $p = max_{\mathcal{M}_j} |\mathcal{M}_j| \leq m$. Then we have

$$
\begin{aligned}
(\|H^*\|_\alpha)^\alpha &= \sum_{i=1}^{m} (\Sigma_{J_j : x^*_{ij}=1} w_j + \Sigma_{J_j \in \mathcal{J}_1} f_{ij})^\alpha \\
&\geq \sum_{i=1}^{m} (\Sigma_{J_j : x^*_{ij}=1} w_j)^\alpha + \sum_{i=1}^{m} (\Sigma_{J_j \in \mathcal{J}_1} f_{ij})^\alpha \\
&= (\|H_1\|_\alpha)^\alpha + \sum_{i=1}^{m} (\Sigma_{J_j \in \mathcal{J}_1} f_{ij})^\alpha \geq (\|H_1\|_\alpha)^\alpha + \sum_{i=1}^{m} \sum_{j=1}^{k} (f_{ij})^\alpha \qquad (6) \\
&= (\|H_1\|_\alpha)^\alpha + \sum_{j=1}^{k} \sum_{i=1}^{m} (f_{ij})^\alpha \geq (\|H_1\|_\alpha)^\alpha + \sum_{j=1}^{k} (\frac{\sum_{i=1}^{m} f_{ij}}{p})^\alpha \\
&= (\|H_1\|_\alpha)^\alpha + \frac{1}{p^\alpha} \sum_{j=1}^{k} (w_j)^\alpha
\end{aligned}
$$

From the fact that H_2 schedules only one task per processor, thus optimal integral assignment for the subset of tasks it assigns and certainly has cost smaller than any integral assignment for the whole set of tasks. In a similar way we have

$$
(\|H_2\|_\alpha)^\alpha = \sum_{j=1}^{k} (w_j)^\alpha \leq (\|OPT\|_\alpha)^\alpha \qquad (7)
$$

So the inequality (6) can be reduced to

$$
(\|H^*\|_\alpha)^\alpha \geq (\|H_1\|_\alpha)^\alpha + \frac{1}{p^\alpha} (\|H_2\|_\alpha)^\alpha \qquad (8)
$$

[1] In H_1 schedule, when the loads of m processors is $\{l^{h1}_1, l^{h1}_2, ..., l^{h1}_m\}$, $\|H_1\|_\alpha$ means $((l^{h1}_1)^\alpha + (l^{h1}_2)^\alpha + ... + (l^{h1}_m)^\alpha)^{\frac{1}{\alpha}}$.

then

$$\begin{aligned}(\|\bar{H}\|_\alpha)^\alpha &= (\|H_1 + H_2\|_\alpha)^\alpha \leq (\|H_1\|_\alpha + \|H_2\|_\alpha)^\alpha \\ &= 2^\alpha(\frac{\|H_1\|_\alpha + \|H_2\|_\alpha}{2})^\alpha \leq 2^\alpha(\frac{1}{2}(\|H_1\|_\alpha)^\alpha + \frac{1}{2}(\|H_2\|_\alpha)^\alpha) \\ &\leq 2^{\alpha-1}((\|H^*\|_\alpha)^\alpha - \frac{1}{p^\alpha}(\|H_2\|_\alpha)^\alpha + (\|H_2\|_\alpha)^\alpha) \\ &\leq 2^{\alpha-1}(2 - \frac{1}{p^\alpha})(\|OPT\|_\alpha)^\alpha\end{aligned}$$

So

$$\frac{(\|\bar{H}\|_\alpha)^\alpha}{(\|OPT\|_\alpha)^\alpha} \leq 2^{\alpha-1}(2 - \frac{1}{p^\alpha})$$

Which concludes the proof that the schedule $\bar{H}$ guarantees a $2^{\alpha-1}(2 - \frac{1}{p^\alpha})$-approximation to optimal solution for the SEMRPP problem and can be found in polynomial time.

(*ii*) Seen from above, we also have

$$\Sigma_{J_j \in \mathcal{J}} \bar{x}_{ij} w_j < \Sigma_{J_j \in \mathcal{J}} x^*_{ij} w_j + max_{J_j \in \mathcal{J}: x^*_{ij} \in (0,1)} w_j, \forall P_i$$

Where the inequality results from the fact that the load of processor P_i in $\bar{H}$ schedule is the load of H^* plus the weight of task matched to it. Because we match each task to one of its child node, i.e., the execution cycle of the adding task $\bar{w}_j < max_{J_j \in \mathcal{J}: x^*_{ij} \in (0,1)} w_j$.

Now we discuss the s_{max}. First we give Proposition 1 to feasible and violation relationship.

Proposition 1. *If* $(\mathbf{P_1})$ *has feasible solution for the SEMRPP problem in the continuous model, we may hardly to solve* $(\mathbf{P_1})$ *without violating the constraint of the limitation of the maximal execution cycles of processors.*

Obviously, if $(\mathbf{P_1})$ has a unique feasible solution, i.e., the maximal execution cycles of processors is set to the OPT solution value. Then if we can always solve $(\mathbf{P_1})$ without violating the constraint, this means we can easily devise an exact algorithm for $(\mathbf{P_1})$. But we have proof that $(\mathbf{P_1})$ is NP-Complete in the strong sense. Next, we give a guarantee speed which can be regarded as fast enough on the restricted parallel processors scheduling in the dependent rounding.

Lemma 5. *Dependent rounding can get the approximation solution without violating the maximal execution cycles of processors constraint when* $s_{max} C \geq max_{P_i \in \mathcal{P}} L_i + max_{J_j \in \mathcal{J}} w_j$, *where* $L_i = \Sigma_{J_j \in \mathcal{J}_i} \frac{1}{|\mathcal{M}_j|} w_j$, $\mathcal{J}_i$ *is the set of tasks that can be assigned to processor* P_i.

Proof. First we denote a vector $\boldsymbol{H} = \{H_1, H_2, ..., H_m\}$ in non-ascending sorted order as the execution cycles of m processors at the beginning of the second step. We also denote a vector $\boldsymbol{L} = \{L_1, L_2, ..., L_m\}$ in non-ascending sorted order as the execution of m processors that $L_i = \Sigma_{J_j \in \mathcal{J}_i} \frac{1}{|\mathcal{M}_j|} w_j$. Now we need to prove

$H_1 \leq L_1$. Suppose we have $H_1 > L_1$, w.l.o.g., assume that the processor P_1 has the execution cycles of H_1. We denote the set of tasks assigned on P_1 as $\mathcal{J}_1^H$. Let $\mathcal{M}_1^H$ be the set of processors to which a task, currently fractional or integral assigned on processor P_1, can be assigned, i.e., $\mathcal{M}_1^H = \bigcup_{J_j \in \mathcal{J}_1^H} \mathcal{M}_j$. Similarly we denote the set of tasks can process on $\mathcal{M}_1^H$ as $\mathcal{J}^H$ and the set of processors $\mathcal{M}^H$ for every task in $P_i \in \mathcal{M}_1^H$ can be assigned. We have $\mathcal{M}^H = \bigcup_{J_j \in \mathcal{J}^H} \mathcal{M}_j$. W.l.o.g, we denote $\mathcal{M}^H$ as a set $\{h_1, h_2, ..., h_k\} (1 \leq k \leq m)$ and also denote a set $\{l_1, l_2, ..., l_k\} (1 \leq k \leq m)$ as its corresponding processors set in $\boldsymbol{L}$. According to the convexity of the objective, we get $H_{h_1} = H_{h_2} = ... = H_{h_k}$. By our assumption, $H_{h_p} > L_{l_q}, \forall p, \forall q$. Then

$$\Sigma_p H_{h_p} > \Sigma_q L_{l_q} \tag{9}$$

Note that each integral task (at the beginning of the second step) in the left part of inequality (9) can also have its respective integral task in the right part, but the right part may have some fractional task. So $\Sigma_q L_{l_q} - \Sigma_p H_{h_p} \geq 0$, i.e., $\Sigma_p H_{h_p} \leq \Sigma_q L_{l_q}$, a contradiction to inequality (9). The assumption is wrong, we have $H_1 \leq L_1$. By Theorem 2's the maximal execution cycles of dependent rounding $\bar{H}_{max}$, we have following process to finish the proof:

$$\begin{aligned} \bar{H}_{max} &< H_1 + max_{J_j \in \mathcal{J}: x_{ij}^* \in (0,1)} w_j \leq L_1 + max_{J_j \in \mathcal{J}: x_{ij}^* \in (0,1)} w_j \\ &\leq L_1 + max_{J_j \in \mathcal{J}} w_j = max_i L_i + max_{J_j \in \mathcal{J}} w_j \quad . \end{aligned}$$

6 Conclusion

In this paper we explore algorithmic instruments leading to reduce energy consumption on restricted parallel processors. We aim at minimizing the sum of energy consumption while the speed scaling method is used to reduce energy consumption under the execution time constraint (C_{max}). We first assess the complexity of scheduling problem under speed and restricted parallel processors settings. We present a polynomial-time approximation algorithm with a $2^{\alpha-1}(2 - \frac{1}{p^\alpha})$-approximation ($p = max_{\mathcal{M}_j} |\mathcal{M}_j| \leq m$) factor for the general case that the tasks have arbitrary size of execution cycles. Specially, when the tasks have an uniform size, we propose an optimal scheduling algorithm with time complexity $O(mn^3 logn)$. (We omit the evaluation results here due to the space limit, see our report [15] for details.)

Acknowledgement. This work was supported by grants of National Natural Science Foundation for China (No. 61020106002, No. 61161160566) and Spanish MINECO/MICINN grant TEC2011-29688-C02-01.

References

1. Mudge, T.: Power: A first-class architecture design constraint. Journal of Computer 34(4), 52–58 (2001)
2. Aupy, G., Benoit, A., Dufossé, F., Robert, Y.: Reclaiming the energy of a schedule: Models and algorithms. INRIA Research report RR-7598 (April 2011); Short version appeared in SPAA 2011

3. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of the IEEE Symposium on Foundation of Computer Science (FOCS 1995), pp. 374–382 (1995)
4. Ishihara, T., Yasuura, H.: Voltage scheduling problem for dynamically variable voltage processors. In: Proceeding of the International Symposium on Low Power Electronics and Design (ISLPED 1998), pp. 197–202 (1998)
5. Irani, S., Shukla, S., Gupta, R.: Algorithms for power savings. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 37–46 (2003)
6. Chen, J., Kuo, W.: Multiprocessor energy-efficient scheduling for real-time jobs with different power characteristics. In: International Conference on Parallel Processing (ICPP 2005), pp. 13–20 (2005)
7. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2007), pp. 289–298 (2007)
8. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory of Computing System 43(1), 67–80 (2008)
9. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: Proceedings of the 21st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2009), pp. 11–18 (2009)
10. Angel, E., Bampis, E., Kacem, F., Letsios, D.: Speed scaling on parallel processors with migration. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 128–140. Springer, Heidelberg (2012)
11. Srikantaiah, S., Kansal, A., Zhao, F.: Energy aware consolidation for cloud computing. In: Proceedings of the Conference on Power Aware Computing and Systems (Hotpower 2008) (2008)
12. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling heterogeneous processors isn't as easy as you think. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), pp. 1242–1253 (2012)
13. Leung, J., Li, L.: Scheduling with processing set restrictions: A survey. International Journal of Production Economics 116(2), 251–262 (2008)
14. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
15. Jin, X., Zhang, F., Song, Y., Fan, L., Liu, Z.: Energy-efficient Scheduling with Time and Processors Eligibility Restrictions. CoRR-abs/1301.4131 (2013), http://arxiv.org/abs/1301.4131
16. Lin, Y., Li, W.: Parallel machine scheduling of machine-dependent jobs with unit-length. European Journal of Operational Research 156(1), 261–266 (2004)
17. Alon, N., Azar, Y., Woeginger, G., Yadid, T.: Approximation schemes for scheduling. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 1997), pp. 493–500 (1997)
18. Azar, Y., Epstein, L., Richter, Y., Woeginger, G.: All-norm approximation algorithms. Journal of Algorithms 52(2), 120–133 (2004)
19. Nesterov, Y., Nemirovskii, A.: Interior-point polynomial algorithms in convex programming. SIAM Studies in Applied Mathematics. SIAM (1994)
20. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding in bipartite graphs. In: Proceedings of the IEEE Symposium on Foundation of Computer Science (FOCS 2002), pp. 323–332 (2002)