ISAECC: An Improved Scheduling Approach for Energy Consumption Constrained Parallel Applications on Heterogeneous Distributed Systems

Ting Ye[†], Zhi-Jie Wang[‡,#], Zhe Quan[†], Song Guo[⊥], Kenli Li[†], and Keqin Li[§]
[†] *College of Information Science and Engineering, Hunan University, Changsha, China*
[‡] *School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China*
[#] *Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, China*
[⊥] *Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong*
[§] *Department of Computer Science, State University of New York, United States*
ytcaro@hnu.edu.cn, wangzhij5@mail.sysu.edu.cn, quanzhe@gmail.com,
song.guo@polyu.edu.hk, lkl510@263.net, lik@newpaltz.edu

***Abstract*—Power-aware task scheduling on processors has been a hot topic. In this paper, we study the problem of minimizing the schedule length for energy consumption constrained parallel applications on heterogeneous distributed systems. Previous work (solving this problem) adopts a policy that preassigns the *minimum* energy consumption for each unassigned task. Nevertheless, our analysis reveals that such a preassignment policy could be unfair, and it may not achieve an optimistic *schedule length*. Motivated by this, we propose a new task scheduling algorithm that suggests a weight-based mechanism to preassign energy consumption for unassigned tasks. We theoretically prove that our preassignment mechanism can guarantee the energy consumption constraint. Also, we have conducted extensive experiments based on two real parallel applications. The results consistently demonstrate that, compared to state-of-the-art algorithms, our approach can achieve smaller schedule length while satisfying the energy consumption constraint.**

***Keywords*-Distributed system, energy consumption, parallel application, task scheduling, preassignment strategy**

I. Introduction

Computers have been developed to achieve higher performance over the past seven decades. While the performance has increased dramatically, power consumption in computer systems has also increased [1, 2]. Such increased energy consumption causes severe economic, ecological, and technical problems [2]. Power conservation is critical in many computation and communication environments and has attracted much attention [1–8].

Generally, there are two kinds of approaches to reduce power consumption in computing systems [2]: (i) using the thermal-aware hardware design; and (ii) using the power-aware software design. As for the latter, there is a well-known mechanism called *dynamic voltage and frequency scaling* (DVFS), which dynamically tunes the energy-delay tradeoff [3, 9]. Owning to this mechanism, power-aware task scheduling on processors with variable voltages and frequencies has been extensively studied [3, 10–13]. There are two main considerations in dealing with the energy-delay tradeoff: (i) in high performance computing systems, techniques and algorithms usually aim to maximize performance under certain energy consumption constraints; and (ii) in low-power devices and systems, techniques and algorithms usually *aim to* minimize energy consumption while still meeting certain performance goals. These two lines of works can be witnessed in [14–17].

Recently, the problem of minimizing schedule length of an energy consumption constrained application with precedence constrained *sequential* tasks was studied in [2] by Li. Later, the author extended this problem to the context of constrained *parallel* tasks [18]. These works were interested in homogeneous systems with shared memory and they cannot be applied to heterogeneous distributed systems. Different from the above works, Xiao et al. [19] studied a variant problem that aims to minimize schedule length of an energy consumption constrained parallel application on heterogeneous distributed systems. To solve this problem, they developed an algorithm called MSLECC. The basic idea of their method is to preassign the minimum energy consumption for each unassigned task to satisfy the energy consumption constraint, and then minimize the schedule length in a heuristic manner. With this idea, their algorithm achieves favourable performance. Nevertheless, we observe that, for the low priority tasks, the preassignment policy in MSLECC could be not fair, which may lead to less optimistic results. To address this issue, this paper develops a new approach that achieves a better performance.

The main contributions of this paper are as follows.

- We design a preassignment strategy that adopts a *weight-based mechanism* (Section III-A), and provide the rigorous proof to show its feasibility (Section III-B).
- We develop a new task scheduling algorithm to minimize the schedule length while considering the energy consumption constraint (Section III-C).
- We evaluate our approach and the experimental results show that it can achieve much shorter schedule length while satisfying the energy consumption constraint (Section IV).

In next section, we introduce some preliminaries, and finally we conclude the paper in Section V.

II. Preliminaries

In this section, we first introduce the models (Section II-A), and then describe the problem to be studied (Section II-B), and finally we review state-of-the-art method for this problem and reveal its limitation (Section II-C).

A. Models

Following prior works [5, 19–22], we use the *directed acyclic graph* (DAG) to represent the application model. Let $U = \{u_1, u_2, ..., u_{|U|}\}$ denote the set of processors, where $|U|$ is the number of processors. The DAG application model is defined as $G = \{N, M, C, W\}$, where N denotes the set of nodes in G, M denotes the set of communication edges, C denotes the set of communication time, W is a matrix with size $|N| \times |U|$. In addition, we define the followings: (i) each node $n_i \in N$ denotes a task; (ii) each edge $m_{i,j} \in M$ denotes the communication message from task n_i to n_j; (iii) $c_{i,j} \in C$ denotes the communication time of $m_{i,j}$ when task n_i and n_j are assigned to different processors; (iv) $w_{i,k}$ denotes the execution time of task n_i running on the processor u_k with the maximum frequency; (v) $pred(n_i)$ and $succ(n_i)$ denote the set of direct predecessor tasks and the set of direct successor tasks of task n_i, respectively; (vi) n_{entry} and n_{exit} denote the task without predecessor and without successor, respectively.

On the other hand, the power model used in this paper follows that in [19, 23]. Specifically, the system power consumption at frequency f is defined as:

$$P(f) = P_s + h(P_{ind} + P_d) = P_s + h(P_{ind} + C_{ef} f^m)$$

where P_s denotes static power (similar to [19, 23], in this paper we also do not consider it, since it is unmanageable), P_{ind} and P_d denote frequency-independent and frequency-dependent dynamic power, respectively, h denotes the system state ($h = 1$ means the system is active, and $h = 0$ means it is inactive), C_{ef} denotes the effective capacitance, and m denotes the dynamic power exponent.

The minimum energy-efficient frequency, denoted by f_{ee}, is defined as

$$f_{ee} = \sqrt[m]{\frac{P_{ind}}{(m-1)C_{ef}}}$$

Clearly, if the frequency of a processor ranges from the minimum value f_{min} to the maximum value f_{max}, then the actual frequency f should be in the interval $[f_{low}, f_{max}]$, where $f_{low} = max(f_{min}, f_{ee})$. In addition, since the processors in system are heterogeneous, we can define the following sets:

- The set of P_{ind}: $\{P_{1,ind}, P_{2,ind}, ..., P_{|U|,ind}\}$;
- The set of P_d: $\{P_{1,d}, P_{2,d}, ..., P_{|U|,d}\}$;
- The set of C_{ef}: $\{C_{1,ef}, C_{2,ef}, ..., C_{|U|,ef}\}$;
- The set of m : $\{m_1, m_2, ..., m_{|U|}\}$;
- The set of actual efficient frequencies:

$$\left\{ \begin{array}{c} \{f_{1,low}, f_{1,\alpha}, ..., f_{1,max}\}, \\ \{f_{2,low}, f_{2,\alpha}, ..., f_{2,max}\}, \\ ..., \\ \{f_{|U|,low}, f_{|U|,\alpha}, ..., f_{|U|,max}\} \end{array} \right\}$$

This way, we can compute the energy consumption of task n_i executed on the processor u_k with frequency $f_{k,h}$ based on the following:

$$E(n_i, u_k, f_{k,h}) = P_{k,h} \times w_{i,k} \times \frac{f_{k,max}}{f_{k,h}} \quad (1)$$

where $P_{k,h} = P_{k,ind} + C_{k,ef} \times (f_{k,h})^{m_k}$.

B. Problem Description

For ease of understanding the problem, we first clarify several definitions.

Definition 1: Given a task n_i executed on processor u_k with frequency $f_{k,h}$, its earliest start time (EST) is denoted as $EST(n_i, u_k, f_{k,h})$, which is computed as

$$\begin{cases} EST(n_{entry}, u_k, f_{k,h}) = 0 \\ EST(n_i, u_k, f_{k,h}) = max\left(avail[k], \max\limits_{n_j \in pred(n_i)} \{AFT(n_j) + c'_{i,j}\}\right) \end{cases}$$

where $avail[k]$ is the earliest available time while processor u_k is ready for executing a task, $AFT(n_j)$ represents the actual finish time of task n_j. $c'_{i,j}$ denotes the communication time between task n_i and n_j. $c'_{i,j} = 0$ if n_i and n_j are assigned to the same processor, otherwise, $c'_{i,j} = c_{i,j}$.

Definition 2: The earliest finish time (EFT) of task n_i executed on processor u_k with frequency $f_{k,h}$ is denoted as $EFT(n_i, u_k, f_{k,h})$, which is computed as

$$EFT(n_i, u_k, f_{k,h}) = EST(n_i, u_k, f_{k,h}) + w_{i,k} \times \frac{f_{k,max}}{f_{k,h}} \quad (2)$$

We now describe the problem to be addressed. Specifically, the scheduling problem discussed in this paper is to find a proper processor and frequency for each task in application G, so as to (i) generate the minimum schedule length $SL(G)$, where $SL(G) = AFT(n_{exit}) = min\{EFT(n_{exit})\}$; and (ii) ensure the actual energy consumption of G, denoted by $E(G)$, is no larger than its given energy consumption constraint $E_{given}(G)$. That is,

$$E(G) = \sum_{i=1}^{|N|} E(n_i, u_{pr(i)}, f_{pr(i),hz(i)}) \leq E_{given}(G) \quad (3)$$

where $u_{pr(i)}$ and $f_{pr(i),hz(i)}$ denote the processor and frequency assigned to task n_i respectively, $f_{pr(i),hz(i)} \in [f_{pr(i),low}, f_{pr(i),max}]$, $u_{pr(i)} \in U$ and $1 \leq i \leq |N|$.

Let $E_{min}(G)$ and $E_{max}(G)$ represent the minimum and maximum energy consumption of application G, respectively. They are calculated as

$$E_{min}(G) = \sum_{i=1}^{|N|} E_{min}(n_i) \tag{4}$$

$$E_{max}(G) = \sum_{i=1}^{|N|} E_{max}(n_i) \tag{5}$$

where the minimum and maximum energy consumption of task n_i are computed as

$$E_{min}(n_i) = \min_{u_k \in U} E(n_i, u_k, f_{k,low}) \tag{6}$$

$$E_{max}(n_i) = \max_{u_k \in U} E(n_i, u_k, f_{k,max}) \tag{7}$$

Note that, throughout this paper, we assume $E_{min}(G) \leq E_{given}(G) \leq E_{max}(G)$.

C. State-Of-The-Art

In this subsection we review the existing method closest to ours, and reveal its limitation through a running example.

► *The MSLECC algorithm.* In the literature, the state-of-the-art method for this problem is proposed in [19]. Their method, called the MSLECC algorithm, consists of several major steps: (i) it gets the sequence of tasks sorted by the *upward rank values* (defined later); (ii) it preassigns the minimum energy consumption for each unscheduled task to satisfy the energy consumption constraint (i.e., $E_{given}(G)$); (iii) it transfers the energy consumption constraint to that of each task; and (iv) it traverses all processors and frequencies to select a proper processor with the minimum EFT for each task in the sequence.

Definition 3: The upward rank value $(rank_u)$ of a task reflects its priority among all the tasks in the application [24]. It is computed as

$$rank_u(n_i) = \frac{\sum_{k=1}^{|U|} w_{i,k}}{|U|} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_u(n_j)\}$$

It is worth noting that, Step (ii) mentioned above is the core of their method, since it assures that the energy consumption constraint is satisfied.

Without loss of generality, one can use $\{n_{s(1)}, n_{s(2)}, ..., n_{s(|N|)}\}$ to denote the sequence of tasks ranked by the $rank_u$ values, and assume that $n_{s(j)}$ is the task that should be assigned currently. Accordingly, one can use $\{\{n_{s(1)}, n_{s(2)}, ..., n_{s(j-1)}\}$ to denote the set of tasks that have been assigned, and $\{n_{s(j+1)}, n_{s(j+2)}, ..., n_{s(|N|)}\}$ to denote the set of tasks that are unassigned. Then, when scheduling the task $n_{s(j)}$, the energy consumption of application G is computed as

$$\begin{aligned} E_{s(j)}(G) = &\sum_{x=1}^{j-1} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ &+ E(n_{s(j)}, u_k, f_{k,h}) + \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \end{aligned} \tag{8}$$

where $E_{pre}(n_{s(y)})$ denotes the preassigned energy consumption for task $n_{s(y)}$.

Fact 1: For any task $n_{s(j)}$ ($j \in [1, ..., |N|]$), if

$$E_{s(j)}(G) \leq E_{given}(G), \tag{9}$$

then the actual energy consumption $E(G) \leq E_{given}(G)$ (cf., Formulate 3) can be satisfied.

Besides Step (ii), another important step is to transfer energy consumption constraint of G to that of each task, recall Step (iii). It is based on the followings. Firstly, by Eqs. 8 and 9, one can have

$$\begin{aligned} E(n_{s(j)}, u_k, f_{k,h}) \leq\ & E_{given}(G) \\ & - \sum_{x=1}^{j-1} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ & - \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \end{aligned}$$

Let the energy consumption constraint of task $n_{s(j)}$ be

$$\begin{aligned} E_{given}(n_{s(j)}) =& E_{given}(G) \\ & - \sum_{x=1}^{j-1} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ & - \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \end{aligned} \tag{10}$$

Further, considering the upper bound $E_{max}(n_{s(j)})$, one can set $E_{given}(n_{s(j)}) = min\{E_{given}(n_{s(j)}), E_{max}(n_{s(j)})\}$. Hence, when processing task $n_{s(j)}$, one just needs to consider the following constraint (instead of the total energy consumption constraint):

$$E(n_{s(j)}, u_k, f_{k,h}) \leq E_{given}(n_{s(j)})$$

Under this constraint, one can assign each task to a processor with the minimum EFT to obtain the minimum schedule length. The above idea essentially transfers the energy consumption constraint of G to that of each task.

► *The limitation of MSLECC.* We now introduce the concept of "extra energy", which will be used in analysing the limitation of MSLECC.

Definition 4: The extra energy refers to the difference between the energy consumption constraint of a task and its preassigned energy consumption. It is computed as

$$\triangle E_{ex}(n_i) = E_{given}(n_i) - E_{pre}(n_i) \tag{11}$$

Note that, for the MSLECC algorithm, it sets $E_{pre}(n_i) = E_{min}(n_i)$, and initially, the "total" extra energy of G, denoted by $\triangle E_{ex}(G)$, can be computed as $\triangle E_{ex}(G) = E_{given}(G) - \sum_{i=1}^{|N|} E_{pre}(n_i)$.

To examine the limitation of MSLECC, we execute a preliminary experiment running the application example shown in Fig. 1. In this experiment, the parallel application with 10

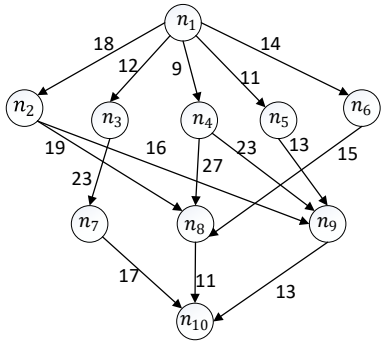


Figure 1. An example of the DAG application model.

Table I
POWER PARAMETERS OF PROCESSORS

u_k	$P_{k,ind}$	$C_{k,ef}$	m_k	$f_{k,low}$	$f_{k,max}$
u_1	0.03	0.8	2.9	0.26	1.0
u_2	0.04	0.8	2.5	0.26	1.0
u_3	0.07	1.0	2.5	0.29	1.0

Table II
EXECUTION TIME OF EACH TASK.

$Task$	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
u_1	14	13	11	13	12	13	7	5	18	21
u_2	16	19	13	8	13	16	15	11	12	7
u_3	9	18	19	17	10	9	11	14	20	16

Table III
SCHEDULING RESULTS

n_i	$E_{given}(n_i)$	$E_{pre}(n_i)$	$\triangle E_{ex}(n_i)$
n_1	13.44	2.48	10.96
n_3	20.33	1.95	18.38
n_4	18.19	2.08	16.11
n_2	19.26	2.30	16.96
n_5	10.92	2.13	8.79
n_6	13.44	2.30	11.14
n_9	5.44	3.12	2.32
n_7	1.32	1.24	0.08
n_8	0.8874	0.8863	0.0011
n_{10}	1.8204	1.8193	0.0011
$E(G) = 80.98, SL(G) = 129.3660$			

tasks is executed on 3 processors; the maximum frequency of each processor is set to 1.0; the frequency precision is set to 0.01; the energy consumption constraint is set as $E_{given}(G) = E_{max}(G) \times 0.5 = 80.995$, and the parameters of all processors are shown in Table I. Then, the scheduling sequence of tasks is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_7, n_8, n_{10}\}$, Table II shows the execution time of each task on three processors with maximum frequency, and Table III shows part of scheduling results.

One can see from Table III that the tasks with higher priorities usually have more extra energy than those with low priorities (cf., the fourth column). For example, the extra energy of task n_3 is 18.38 while task n_7 is just 0.08. The underlying reason could be that MSLECC uses the policy of preassigning the minimum energy consumption for each unscheduled task; this leads to the vast majority of total extra energy to be shared by the tasks with higher priories. On the contrary, the low priority tasks have to find the processors that have low energy consumption (since the available energy consumption is little); this leads to less chance to choose optimistic schedule length. The phenomenon above essentially implies that, the preassignment policy in MSLECC could be somewhat extreme, and so it could be nice if one can have a more competitive task scheduling algorithm that allows us to obtain smaller schedule length while satisfying the energy consumption constraint. This is just the focus of our paper.

III. OUR SOLUTION

The central idea of our approach is to preallocate the energy consumption for unscheduled tasks by a weight mechanism, instead of directly preallocating the minimum energy consumption for them. In what follows, we first show how to preassign energy consumption based on the so-called weight mechanism (Section III-A), and then prove that such a preassignment method can always satisfy the energy consumption constraint (Section III-B). Finally, Section III-C presents the improved scheduling approach for energy consumption constrained parallel applications (ISAECC).

A. Preassigning Energy Consumption

We first present several concepts, which benefit to understand our preassignment strategy.

Definition 5: Given $E_{min}(G)$ and $E_{given}(G)$, the improvable energy, denoted by $E_{ie}(G)$, is computed as

$$E_{ie}(G) = E_{given}(G) - E_{min}(G) \tag{12}$$

Definition 6: Given a task n_i, its energy consumption level $E_{ave}(n_i)$ is defined as the average of its *maximum* and *minimum* energy consumption, i.e., $E_{ave}(n_i) = \frac{E_{max}(n_i)+E_{min}(n_i)}{2}$.

Like Definition 6, we can define the *energy consumption level* of *an application* G by $E_{ave}(G)$ if replacing n_i with G.

Definition 7: Given a task n_i, the **weight** of its energy consumption level, denoted by $el(n_i)$, is defined as

$$el(n_i) = \frac{E_{ave}(n_i)}{E_{ave}(G)} \tag{13}$$

Here $\sum_{i=1}^{|N|} el(n_i) = 1$. In the sequel, we show how to preassign the energy consumption for each task n_i based on the weight.

Specifically, in our approach the preassigned energy consumption $E_{pre}(n_i)$ is computed as

$$E_{pre}(n_i) = min\{E_{wa}(n_i), E_{max}(n_i)\} \tag{14}$$

where $E_{wa}(n_i)$ is computed based on the following:

$$E_{wa}(n_i) = E_{ie}(G) \times el(n_i) + E_{min}(n_i) \tag{15}$$

where $E_{ie}(G)$ refers to the improvable energy in terms of G (recall Eq. 12). Note that, Eq. 15 could reflect the basic idea of our method. Regarding Eq. 14, it is mainly for assuming that, in the extreme case the preassigned energy consumption is no larger the upper bound $E_{max}(n_i)$.

To this step, a natural question is "does the above preassignment mechanism satisfy the energy consumption constraint?" Next, we address this question positively.

B. Feasibility of The Preassignment Mechanism

To prove the feasibility, we only need to show the following theorem holds.

Theorem 1: Given an application G, and assume we preassign the energy consumption for unscheduled tasks by the weight mechanism, then each task $n_{s(j)}$ can always find a processor to satisfy Eq. 9.

Proof: We prove it by *induction*. Firstly, for the first task $n_{s(1)}$, all other $|N|-1$ tasks in application G are unassigned. Then, by Eqs. 8, 12, 13, 15, and 14, we have

$$\begin{aligned} E_{s(1)}(G) &= E(n_{s(1)}, u_k, f_{k,h}) + \sum_{y=2}^{|N|} E_{pre}(n_{s(y)}) \\ &\leq E(n_{s(1)}, u_k, f_{k,h}) + \sum_{y=2}^{|N|} E_{wa}(n_{s(y)}) \\ &= E(n_{s(1)}, u_k, f_{k,h}) + \sum_{y=1}^{|N|} E_{wa}(n_{s(y)}) - E_{wa}(n_{s(1)}) \\ &= E(n_{s(1)}, u_k, f_{k,h}) + E_{given}(G) - E_{wa}(n_{s(1)}) \end{aligned}$$

and $E_{wa}(n_{s(1)}) \geq E_{min}(n_{s(1)})$. This means that, $n_{s(1)}$ at least can find a processor that satisfies its minimum energy consumption $E_{min}(n_{s(1)})$. In other words, when $E(n_{s(1)}, u_k, f_{k,h}) = E_{min}(n_{s(1)})$, we have

$$\begin{aligned} E_{s(1)}(G) &= E(n_{s(1)}, u_k, f_{k,h}) + E_{given}(G) - E_{wa}(n_{s(1)}) \\ &\leq E_{given}(G) \end{aligned}$$

This essentially shows that Eq. 9 is satisfied for $n_{s(1)}$.

Secondly, without loss of generality, assume that for the jth task $n_{s(j)}$ it can find a processor $u_{pr(s(j))}$ and frequency $f_{pr(s(j)),hz(s(j))}$ to satisfy the Eq. 9. That is,

$$\begin{aligned} E_{s(j)}(G) &= \sum_{x=1}^{j-1} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ &\quad + E(n_{s(j)}, u_{pr(s(j))}, f_{pr(s(j)),hz(s(j))}) \\ &\quad + \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \\ &= \sum_{x=1}^{j} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ &\quad + \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \\ &\leq E_{given}(G) \end{aligned}$$

The above formulation can be written as

$$\begin{aligned} &\sum_{x=1}^{j} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ &\qquad \leq E_{given}(G) - \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \end{aligned} \tag{16}$$

Thirdly, for the $(j+1)$th task $n_{s(j+1)}$, the energy consumption of application G is

$$\begin{aligned} E_{s(j+1)}(G) = &\sum_{x=1}^{j} E(n_{s(x)}, u_{pr(s(x))}, f_{pr(s(x)),hz(s(x))}) \\ &+ E(n_{s(j+1)}, u_k, f_{k,h}) + \sum_{y=j+2}^{|N|} E_{pre}(n_{s(y)}) \end{aligned} \tag{17}$$

Combing Eqs. 16 and 17, we get

$$\begin{aligned} E_{s(j+1)}(G) &\leq E_{given}(G) - \sum_{y=j+1}^{|N|} E_{pre}(n_{s(y)}) \\ &\quad + E(n_{s(j+1)}, u_k, f_{k,h}) + \sum_{y=j+2}^{|N|} E_{pre}(n_{s(y)}) \\ &= E_{given}(G) + E(n_{s(j+1)}, u_k, f_{k,h}) - E_{pre}(n_{s(j+1)}) \end{aligned}$$

By Eqs. (15), (14), we can know $E_{pre}(n_{s(j+1)}) \geq E_{min}(n_{s(j+1)})$. That means, when the energy consumption $E(n_{s(j+1)}, u_k, f_{k,h})$ assigned to task $n_{s(j+1)}$ is in the interval $\left[E_{min}(n_{s(j+1)}), E_{pre}(n_{s(j+1)})\right]$, we have:

$$E_{s(j+1)}(G) \leq E_{given}(G) \tag{18}$$

Putting all together, hence Theorem 1 holds. ■

C. The Proposed Algorithm

Our approach (i.e., ISAECC) is shown in Algorithm 1. In brief, Lines 2-6 are for calculating some values (e.g., energy consumption level) for each task and also for the application G, while Lines 7-8 are to calculate the preassigned energy consumption for each task. Lines 9-22 are to select processor and frequency for each task. In Lines 13-22, all processors and frequencies are traversed for mapping the task to the processor with the minimum EFT. Finally, Lines 23-24 are to calculate the actual energy consumption $E(G)$ and the final schedule length $SL(G)$.

Theorem 2: The time complexity of the ISAECC Algorithm is $O(|N|^2 \times |U| \times |F|)$, where $|F|$ represents the maximum number of discrete frequencies from $f_{k,low}$ to $f_{k,max}$.

Proof: For each task in the list, selecting the processor with the minimum EFT has complexity $O(|N| \times |U| \times |F|)$, and traversing all tasks needs a time complexity of $O(|N|)$. Thus, the total time is $O(|N|^2 \times |U| \times |F|)$. ■

Algorithm 1 The ISAECC Algorithm

Input: G=(N,M,C,W),U,$E_{given}(G)$
Output: SL(G),E(G)

Sort tasks in a list dl by descending order of $rank_u$;
for $(\forall i, n_i \in N)$ **do**
  Compute $E_{min}(n_i)$ and $E_{max}(n_i)$; // Eqs. 6 and 7
  Compute $E_{ave}(n_i)$;
Compute $E_{min}(G)$ and $E_{max}(G)$; // Eqs. 4 and 5
Compute $E_{ave}(G)$;
for $(\forall i, n_i \in N)$ **do**
  Compute $E_{pre}(n_i)$; // Eq. 14
while (dl is not empty) **do**
  $n_i = dl$.out();
  $AFT(n_i) = \infty$;
  Compute $E_{given}(n_i)$; // Eq. 10
  for all $(u_k \in U)$ **do**
    for all h,$f_{k,h} \in [f_{k,low}, f_{k,max}]$ **do**
      Compute $E(n_i, u_k, f_{k,h})$; // Eq. 1
      if $E(n_i, u_k, f_{k,h}) > E_{given}(n_i)$ **then**
        continue;
      Compute $EFT(n_i, u_k, f_{k,h})$; // Eq. 2
      if $(EFT(n_i, u_k, f_{k,h}) < AFT(n_i))$ **then**
        Let $pr(i) = k$, and $f_{pr(i),hz(i)} = f_{k,h}$;
        $E(n_i, u_{pr(i)}, f_{pr(i),hz(i)}) = E(n_i, u_k, f_{k,h})$;
        $AFT(n_i) = EFT(n_i, u_k, f_{k,h})$;
Compute actual energy consumption $E(G)$; // Eq. 3
Compute the schedule length $SL(G) = AFT(n_{exit})$;
return $E(G)$, $SL(G)$

► *A running example.* We still consider the application example described in Fig 1. For the fair comparison, all parameters are the same as in II-C. Table IV shows the task scheduling results generated by ISAECC.

From Table IV, we can see that the total energy consumption $E(G) = 75.3619$, which is less than $E_{given}(G)$ and the value 80.9939 got by MSLECC. The final schedule length $SL(G) = 86.4233$, which is better than 129.3600 obtained by MSLECC. In addition, compared with Table III, the "extra energy" of different tasks with different priorities does not show large differences. All these results show us that our method should be effective and relatively fair for all tasks.

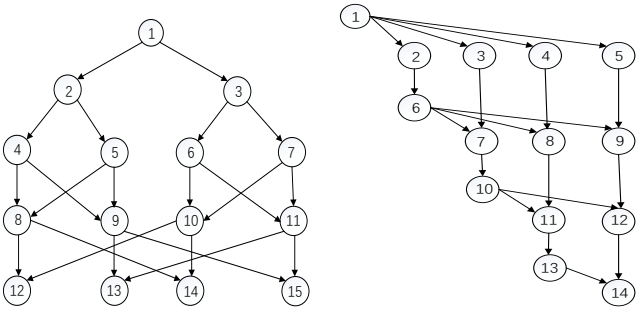
IV. Experiments

A. Experimental Settings

We compare the performance of our algorithm with the HEFT [24] and the MSLECC [19] using the metrics $E(G)$ and $SL(G)$. The HEFT is a precedence-constrained application scheduling algorithm for minimizing the schedule length on heterogeneous systems. Note that, this method does not consider the power consumption constraint. In contrast, the MSLECC is much more close to our algorithm, since both of us solve the same problem (with the same constraints), recall Section II-C.

Table IV
TASK ASSIGNMENT OF APPLICATION IN FIG.1 USING ISAECC

n_i	$E_{given}(n_i)$	$u(n_i)$	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$	$\triangle E_{ex}(n_i)$
n_1	7.7815	u_3	0.84	0.0	10.7143	7.6789	0.0000
n_3	9.4689	u_1	1.0	22.7143	33.7143	9.1300	0.1027
n_4	9.1651	u_2	1.0	19.7143	27.7143	6.7200	0.3389
n_2	11.9277	u_3	0.67	10.7143	37.5800	11.7521	2.4451
n_5	6.6457	u_2	0.68	27.7143	46.8320	6.5964	0.1577
n_6	7.5945	u_3	0.83	37.5800	48.4233	7.5645	0.0493
n_9	11.3104	u_2	1.0	53.5800	65.5800	10.0800	0.0300
n_7	7.0785	u_1	1.0	33.7143	40.7143	5.8100	1.2304
n_8	7.4362	u_1	1.0	63.4233	68.4233	4.1500	1.2685
n_{10}	11.5131	u_2	1.0	79.4233	86.4233	5.8800	3.2862
$E(G) = 75.3619, SL(G) = 86.4233$							



(a) FFT application with $\rho = 4$ (b) GE application with $\rho = 5$

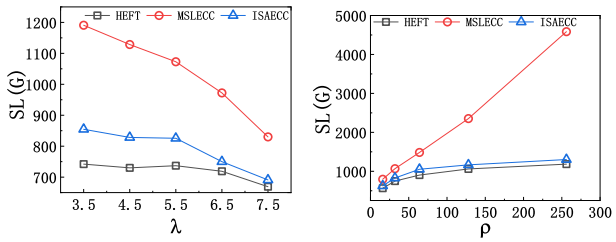
Figure 2. Example of real parallel applications.

In our experiments, we select two real parallel applications: fast Fourier transform (FFT) and Gaussian elimination (GE) for tests. Fig. 2 (a) shows an example of FFT parallel application with $\rho = 4$, where ρ is a parameter representing the size of application. For the FFT graph, the total number of tasks is $|N| = (2 \times \rho - 1) + \rho \times \log_2 \rho$, where $\rho = 2^y$ for some integer y. Note that, the FFT parallel application with the size ρ has ρ "exit" tasks; see e.g., the tasks numbered as 12, 13, 14 and 15 in Fig. 2 (a). In order to match the application model (recall Section II-A), we add a "dummy" `exit` task, whose execution time is zero; and we connect the dummy exit task to the last ρ exit tasks, and set their communication time to 0. On the other hand, the size of a GE application is $|N| = \frac{\rho^2+\rho-2}{2}$. Fig. 2 (b) shows a GE parallel application example with $\rho = 5$.

The simulated heterogeneous platform contains 64 processors. The application and processor parameters are: $10ms \leq w_{i,k} \leq 100ms$, $10ms \leq c_{i,j} \leq 100ms$, $0.03 \leq P_{k,ind} \leq 0.07$, $0.8 \leq C_{k,ef} \leq 1.2$, $2.5 \leq m_k \leq 3.0$, and $f_{k,max} = 1.0$ GHZ. The frequency precision is 0.01 GHz. For ease of observing the effectiveness of ISAECC, in our experiments we vary the sizes of energy consumption constraints and the scales of applications, respectively.

Table V
SCHEDULING RESULTS OF FFT APPLICATION WITH $\rho = 32$.

$E_{given}(G)$	HEFT		MSLECC		ISAECC	
	$E(G)$	$SL(G)$	$E(G)$	$SL(G)$	$E(G)$	$SL(G)$
2167.97	5374.18	742	2167.96	1190.73	2167.92	854.53
2648.41	5190.55	730	2648.40	1128.38	2647.24	828.31
3445.54	5221.23	737	3445.54	1027.69	3443.03	825.61
3823.87	4804.34	719	3823.87	972.00	3795.81	750.00
4561.67	5464.16	669	4561.67	830.00	4431.41	691.92

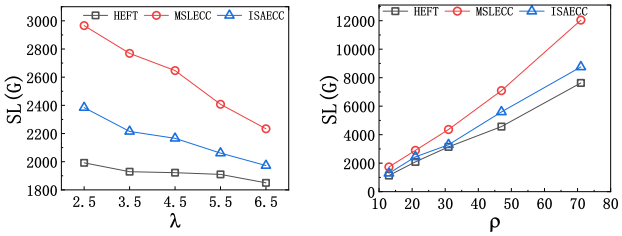


(a) (b)

Figure 3. The schedule lengths of FFT parallel application. (a) varying λ (essentially, the energy consumption constraint $E_{given}(G)$); (b) varying ρ (essentially, the scale of application).

Table VII
SCHEDULING RESULTS OF GE APPLICATION WITH $\rho = 21$.

$E_{given}(G)$	HEFT		MSLECC		ISAECC	
	$E(G)$	$SL(G)$	$E(G)$	$SL(G)$	$E(G)$	$SL(G)$
1588.53	3967.66	1992	1588.53	2966.25	1588.48	2383.98
2235.07	4281.90	1929	2235.07	2769.04	2234.13	2214.94
2885.79	4339.69	1922	2885.79	2646.75	2881.44	2166.12
3449.71	3981.84	1910	3449.71	2407.57	3449.53	2061.08
3936.61	4247.82	1850	3936.61	2233.73	3901.48	1972.72



(a) (b)

Figure 4. The schedule lengths of GE parallel application. (a) varying λ; (b) varying ρ.

B. Experimental Results for FFT Application

Exp-1. In this experiment, we compare the $E(G)$ and $SL(G)$ of the FFT application under different $E_{given}(G)$. The application size is set to $\rho = 32$ (i.e.,$|N| = 233$). $E_{given}(G)$ is set to $E_{min}(G) \times \lambda$. We vary λ from 3.5 to 7.5.

Table V shows the scheduling results. We can see that although HEFT obtains the smaller schedule length, it exceeds the energy consumption constraint in each case. As for MSLECC and ISAECC, they are always able to satisfy the given energy consumption constraint even if the given energy consumption is small. For the sake of intuition, Fig. 3(a) shows the final schedule length for varying λ (notice: it essentially varies the energy consumption constraint $E_{given}(G)$, since $E_{given}(G) = \lambda \times E_{min}(G)$). It can be seen that, compared to MSLECC, our algorithm has the obvious advantage on the final schedule length, especially when the given energy consumption is small. This is because both the preassignment policy of MSLECC and the small $E_{given}(G)$ make the available energy consumption of low priority tasks turn less, which leads to the long schedule length. In addition, as we expected, the larger $E_{given}(G)$ is, the better schedule length we can obtain.

Exp-2. In this experiment, we fix λ to 4.5 (i.e., $E_{given}(G) = E_{min}(G) \times 4.5$), and then vary the scale of the application. Specifically, we vary ρ from 16 (i.e., $|N| = 95$, small scale) to 256 (i.e.,$|N| = 2559$, large scale).

Table VI shows the scheduling results. Similar to Exp-1, although HEFT can obtain the smaller schedule length while it exceeds the energy consumption constraint in each case. Also, both MSLECC and ISAECC can always satisfy the energy consumption constraint. On the other hand, Fig. 3(b) depicts the variation tendency of the final schedule length. It can be seen that, for the MSLECC algorithm, the final schedule length dramatically increases when the scale grows, while the final schedule length obtained by our algorithm only increases slightly. This essentially illustrates that our algorithm has the better scalability.

C. Experimental Results for GE Application

Exp-3. This experiment compares the $E(G)$ and $SL(G)$ of GE parallel application under different $E_{given}(G)$. The application size is limited to $\rho = 21$ (i.e.,$|N|$=230), which

Table VI
SCHEDULING RESULTS OF FFT APPLICATION WITH $\lambda = 4.5$.

ρ	$\|N\|$	$E_{given}(G)$	HEFT		MSLECC		ISAECC	
			$E(G)$	$SL(G)$	$E(G)$	$SL(G)$	$E(G)$	$SL(G)$
16	95	1190.79	2216.57	566	1190.78	795.46	1186.68	625.85
32	233	3119.76	5479.67	749	3119.75	1067.00	3115.05	828.00
64	511	6751.48	10741.35	903	6751.48	1484.46	6747.45	1052.53
128	1151	16639.31	22912.71	1060	16639.30	2353.92	16621.53	1165.85
256	2559	31245.50	45115.35	1184	31245.50	4585.00	31241.40	1302.45

Table VIII
SCHEDULING RESULTS OF GE PARALLEL APPLICATION WITH $\lambda = 4.5$.

ρ	$\|N\|$	$E_{given}(G)$	HEFT		MSLECC		ISAECC	
			$E(G)$	$SL(G)$	$E(G)$	$SL(G)$	$E(G)$	$SL(G)$
13	90	1064.63	1793.1	1135	1064.62	1735.31	1061.57	1294.65
21	230	2879.52	3952.13	2092	2879.52	2899.03	2863.37	2442.64
31	495	6090.62	9607.13	3144	6090.62	4361.69	6090.43	3291.08
47	1127	13516.70	23625.26	4566	13516.70	7090.65	13513.44	5598.46
71	2555	30591.75	52439.49	7634	30591.75	12042.50	30591.49	8757.40

is roughly equal to that in Exp-1. We vary $E_{given}(G)$ from $E_{min}(G) \times 2.5$ to $E_{min}(G) \times 6.5$.

Fig. 4(a) plots the final schedule lengths of all cases, and Table VII shows the detailed scheduling results. Both of them indicate that ISAECC achieves better schedule lengths than MSLECC. Similar to Exp-1, MSLECC and ISAECC can always satisfy energy consumption constraints while HEFT always exceeds the given constraints. On the other hand, combining Exp-3 and Exp-1, it shows that our solution is feasible for different types of applications.

Exp-4. We fix $E_{given}(G)$ to $E_{min}(G) \times 4.5$, and vary ρ from 13 (i.e., $|N| = 90$, small scale) to 71 (i.e., $|N| = 2555$, large scale). These scales are roughly equal to those in Exp-2 for FFT parallel application.

Table VIII shows detailed scheduling results, and Fig. 4(b) plots the variation tendency of final schedule length when varying ρ. Similar to Exp-2, the actual energy consumption using ISAECC and MSLECC is still within the given constraint, and our algorithm can generate the shorter schedule length, compared against the MSLECC algorithm. Additionally, by comparing Figs. 4(b) and 3(b), we find an interesting phenomenon. That is, when ρ increases, the schedule lengths obtained by our method and HEFT increase *slightly* in Fig. 3(b), while they increase *dramatically* in Fig. 4(b). This phenomenon could be due to that the FFT parallel application has better parallelism than the GE parallel application.

V. Conclusion

In this paper, we proposed a new scheduling algorithm to minimize the schedule length for energy consumption constrained parallel applications on heterogeneous distributed systems. The central idea of our algorithm is using a weight-based mechanism to preassign the energy consumption for unassigned tasks. Extensive experiments based on two real applications consistently demonstrated that our proposed algorithm is effective and competitive, compared against state-of-the-art algorithms.

References

[1] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, 2005.

[2] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Computers*, vol. 61, no. 12, pp. 1668–1681, 2012.

[3] P. Macken, M. Degrauwe, M. V. Paemel, and H. Oguey., "A voltage reduction technique for digital systems," in *ISSCC*, 1990, pp. 238–239.

[4] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *OSDI*, 1994, pp. 13–23.

[5] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *CCGRID*, 2012, pp. 781–786.

[6] K. Gharehbaghi, F. Kocer, and H. Kulah, "Optimization of power conversion efficiency in threshold self-compensated UHF rectifiers with charge conservation principle," *IEEE Trans. on Circuits and Systems*, vol. 64-I, no. 9, pp. 2380–2387, 2017.

[7] S. Albers, "Energy-efficient algorithms," *Commun. ACM*, vol. 53, no. 5, pp. 86–96, 2010.

[8] F. Sandoval, G. Poitau, and F. Gagnon, "Hybrid peak-to-average power ratio reduction techniques: Review and performance comparison," *IEEE Access*, vol. 5, pp. 27 145–27 161, 2017.

[9] M. R. Stan and K. Skadron, "Guest editors' introduction: Power-aware computing," *IEEE Computer*, vol. 36, no. 12, pp. 35–38, 2003.

[10] F. F. Yao, A. J. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *FOCS*, 1995, pp. 374–382.

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

[12] J. R. Lorch and A. J. Smith, "PACE: A new approach to dynamic voltage scaling," *IEEE Trans. Computers*, vol. 53, no. 7, pp. 856–869, 2004.

[13] 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, 2005.

[14] D. P. Bunde, "Power-aware scheduling for makespan and flow," *J. Scheduling*, vol. 12, no. 5, pp. 489–500, 2009.

[15] S. Cho and R. G. Melhem, "On the interplay of parallelization, program performance, and energy consumption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 3, pp. 342–353, 2010.

[16] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, 2011.

[17] S. U. Khan and I. Ahmad, "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 3, pp. 346–360, 2009.

[18] K. Li, "Power and performance management for parallel computations in clouds and data centers," *J. Comput. Syst. Sci.*, vol. 82, no. 2, pp. 174–190, 2016.

[19] X. Xiao, G. Xie, R. Li, and K. Li, "Minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems," in *ISPA*, 2016, pp. 1471–1476.

[20] G. Zeng, Y. Matsubara, H. Tomiyama, and H. Takada, "Energy-aware task migration for multiprocessor real-time systems," *Future Generation Comp. Syst.*, vol. 56, pp. 220–228, 2016.

[21] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment," *J. Grid Comput.*, vol. 14, no. 1, pp. 55–74, 2016.

[22] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, 2011.

[23] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Trans. Design Autom. Electr. Syst.*, vol. 18, no. 2, pp. 23:1–23:21, 2013.

[24] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.