Task Scheduling for Energy Consumption Constrained Parallel Applications on Heterogeneous Computing Systems

Zhe Quan, Zhi-Jie Wang, Ting Ye, and Song Guo, *Senior Member, IEEE*

Abstract—Power-aware task scheduling on processors has been a research hotspot in computing systems. Given an application G containing a set N of tasks $\{n_1, ..., n_{|N|}\}$, and a system containing a set U of processors $\{u_1, ..., u_{|U|}\}$, the power-aware task scheduling generally refers to finding the appropriate processor and frequency for each task n_i, so as to make sure that all tasks can be finished efficiently and the overall energy consumption is guaranteed. In this paper, we study the problem of minimizing the *schedule length* for energy consumption constrained parallel applications on heterogeneous computing systems, where *the schedule length* refers to the time interval between starting the first task and finishing the last task. For this problem, existing work 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 for the low priority tasks, and it may not achieve an optimistic *schedule length*. Thereby, we propose a new task scheduling algorithm that suggests a weight-based mechanism to preassign energy consumption for unassigned tasks, and we provide the rigorous proof to show its feasibility. Further, we show that this idea can be extended to solve *reliability* maximization problems with energy consumption constraint or with both deadline and energy consumption constraints, where the *reliability* refers to the probability of executing application G without failures, and the deadline constraint refers to the "allowable" maximum schedule length. We have conducted extensive experiments based on real parallel applications. The experimental results consistently demonstrate that our proposed algorithms can achieve favourable performance, compared to state-of-the-art algorithms.

Index Terms—heterogeneous systems, energy consumption, parallel application, task scheduling, reliability

1 Introduction

Computers have been developed to achieve higher performance over the past seven decades, due to the rapid growing information technology (IT) demands in both industry and academia [2, 3, 4, 5, 6]. Computing systems offer powerful computing and data storage ability, and thereby they have been widely used for supporting industrial and scientific workflows [7, 8, 9]. On the other hand, although the performance of such systems has increased dramatically, the power consumption has also increased. The increased energy consumption causes severe economic, ecological, and technical issues [3, 10, 11, 12]. A typical approach to reducing power consumption in computing systems is to use the power-aware software design [2, 3, 5, 13]. A well-known mechanism, called *dynamic voltage and frequency scaling* (DVFS), dynamically tunes the energy-delay tradeoff [13, 14, 15]. As a result, power-aware task scheduling on processors with variable voltages and frequencies has incurred extensive studies [16, 17, 18, 19, 20, 21, 22, 23, 24].

- *Zhe Quan, and Ting Ye are with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China. E-mail: {quanzhe,ytcaro,lkl}@hnu.edu.cn,*
- *Zhi-Jie Wang is with (i) the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China, (ii) the Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, China, and (iii) the National Engineering Laboratory for Big Data Analysis and Applications, Beijing, China. E-mail: wangzhij5@mail.sysu.edu.cn*
- *Song Guo is with the Department of Computing, Hong Kong Polytechnic University, Kowloon 999077, Hong Kong. E-mail: song.guo@polyu.edu.hk*

In the above works, some efforts were made to *minimize* energy consumption while still meeting certain performance goals. Others were made to *maximize* or *minimize* some performance metric under certain energy consumption constraints. Our work belongs to the second category, and in particular, we study the following problem: minimizing the schedule length for energy consumption constraint parallel applications on heterogeneous computing systems. Some previous works [3, 25] considered this problem on the *homogeneous* systems with shared memory, so they cannot be applied to *heterogeneous* computing systems. Recently, Xiao *et al.* [26] studied the problem on the *heterogeneous* computing systems, and developed an algorithm called MSLECC (minimizing schedule length of energy consumption constrained). Its basic idea is to guarantee the overall energy consumption constraint by passing all energy constraint on each task and preassigning the minimum energy consumption for each unscheduled task, and then minimize the schedule length in a heuristic manner. In this way, the algorithm achieves a favourable performance. Nevertheless, we observe that, for the low priority tasks, the preassignment policy in MSLECC could be unfair, which may lead to less optimistic results. This is because such a policy could make the available energy consumption for low priority tasks far less than that for high priority tasks, and so the low priority task may have to choose the processor that could incur a long scheduling length, due to the energy consumption constraint. Motivated by this, we develop a new approach called ISAECC, whose key idea is to use a weighted energy preallocation, instead of

a fixed scheme (i.e., the minimum energy preallocation), for unscheduled tasks. The rationale behind our mechanism is to make the energy consumption allocation more relevant to the characteristics of task itself (e.g., energy consumption level) and reduce the influence of priority order on energy consumption allocation. We provide the rigorous proof to show the feasibility of our proposed approach.

In addition, we show that the above idea can be extended to solve another interesting problem — maximizing *reliability* for energy consumption constrained parallel applications on heterogeneous computing systems, where the *reliability* refers to the probability of executing application G without failures. High reliability is important and previous studies [27, 28, 29] indicated that, the DVFS technique may cause a sharp rise in transient failures of processors, consequently affecting the reliability. Although we only simply extend the above idea, the adapted algorithm can achieve a favourable performance for the reliability maximization problem. This essentially reflects, from another perspective, the effectiveness of the weighted energy preallocation mechanism. To further demonstrate the usefulness of the above idea, we also extend it to solve another reliability maximization problem which considers two constrains: deadline and energy consumption constraints, where the deadline constraint refers to the "allowable" maximum schedule length [11, 12]. The main idea for solving this problem is to apply our ISAECC algorithm (mentioned earlier) to get a preliminary scheduling result at first, and then "reclaim" the slack time between the deadline constraint and the schedule length to reallocate the tasks. In this way, one can achieve the large reliability as far as possible, while the deadline and energy consumptions are still satisfied. To summarize, the main contributions of this paper are as follows.

- We design a preassignment strategy that adopts a *weight-based mechanism*. Also, we provide the rigorous proof to show its feasibility.
- We develop a new task scheduling algorithm to minimize the schedule length while considering the energy consumption constraint.
- We extend the above idea to solve reliability maximization problems with energy consumption constraint or with both deadline and energy consumption constraints.
- We evaluate our proposed algorithms based on real parallel applications. The experimental results consistently demonstrate the superiorities and competitiveness of our algorithms.

The rest of the paper is organized as follows. Section 2 reviews prior works most related to ours. Section 3 gives some preliminaries related to the problem of minimizing the schedule length for energy consumption constrained parallel applications. In Section 4, we present our approach for this problem. In Section 5, we introduce two reliability maximization problems and present our algorithms, respectively. In Section 6, we discuss and analyze the experimental results. Finally, we conclude the paper in Section 7.

2 Related Work

Power-aware task scheduling is an important and hot topic in parallel and distributed computing. This section reviews previous works most related to ours. For ease of exposition, we classify previous works into two categories: (i) *schedule length based task scheduling*, i.e., focusing more attention on minimizing the schedule length when executing power-aware task scheduling; and (ii) *reliability based task scheduling*, i.e., focusing more attention on obtaining the high reliability when executing power-aware task scheduling.

2.1 Schedule Length Based Task Scheduling

Since the DVFS-based energy-efficient design technique was first introduced in [2], it has been widely used in energy-related task scheduling problems [3, 30, 31, 32]. For example, the work [30] considered the energy-aware task scheduling problem as a combinatorial optimization problem. The work [3] studied the problem of minimizing schedule length for energy consumption constrained *sequential* applications. Later, this problem is extended to the context of constrained *parallel* tasks [25]. Moreover, in [32] the authors considered three constraints (i.e., energy, deadline and reward). Recently, a survey [33] summarizes various scheduling strategies in data center networks. These works were mainly interested in *data center networks* or *homogeneous* systems with shared memory, and so different from our work.

Actually, many previous works [32, 34, 35] studied task scheduling problems on *heterogeneous* systems. For example, Huang *et al.* [36] proposed an enhanced energy-efficient scheduling algorithm to reduce energy consumption while meeting the service level agreement. The work [35] proposed a DVFS-enabled energy-efficient workflow task scheduling algorithm. Rusu *et al.* [32] suggested an efficient algorithm for minimizing energy consumption while meeting multiple constraints. Generally, the problems studied in these works are opposite to our problem, since we are interested in minimizing scheduling length under certain energy consumption constraints, instead of minimizing energy consumption while meeting other constraints.

Besides the above works, there are many other excellent works that could be much more related to ours. For example, Lee *et al.* [23] proposed energy-conscious scheduling (ECS) for parallel application on heterogeneous distributed systems to implement joint minimization of schedule length and energy consumption. They focused on addressing the trade-off between the quality of schedules and energy consumption. In [37], the authors suggested an approach that uses the constrained critical paths (CCPs) to accomplish better schedules. A representative work [38] proposed the Heterogeneous Earliest Finish Time (HEFT) algorithm, which was developed for minimizing the schedule length on heterogeneous systems. Their model is the same to ours, yet they did not fully consider the energy consumption constraint. The work most closet to ours could be [26], in which both the problem and the used model are the same to ours. Their paper proposed an algorithm called MSLECC. The major feature of their method is to preassign the minimum energy consumption for each unassigned task to satisfy the constraint. Instead, our work proposed a weight-based mechanism, achieving a better performance.

2.2 Reliability Based Task Scheduling

Regarding reliability based task scheduling, there are many excellent works. Some prior works [39, 40, 41, 42, 43, 44,

45] focused on leveraging backup and redundant devices to improve the reliability. These works are obviously different from ours. There are also some works that did not leverage backup and redundant devices to improve the reliability. For example, the work [29] studied the problem of maximizing reliability of real-time embedded applications under hard energy constraint. Tang *et al.* designed a reliability-driven scheduling architecture in grid systems [46]. Generally, these works are based on simple models which cannot precisely reflect heterogeneous computing systems.

Some previous works investigated the problems which are (generally) opposite to our problem. For example, the work [28] studied the energy and reliability scheduling co-design problem for real-time cyber-physical systems; their goal is to *minimize total energy* while *guaranteeing reliability constraints*. Xie *et al.* [21] studied the problem of minimizing the resource cost for a reliable parallel application on heterogeneous embedded systems. Zhu *et al.* [47] studied reliability-aware energy management schemes to minimize energy consumption while preserving the reliability.

There are also some works solving the problems similar to, but essentially different from, our problem. For example, The work [48] suggested an algorithm to pursue low energy consumption and high system reliability for workflow scheduling. In [49], the authors considered both reliability and execution time, and developed scheduling algorithms to maximize reliability and minimize execute time.

We realize that the problem studied in [27] is consistent with the problem we considered. To maximize reliability with energy conservation for parallel task scheduling, they developed a Reliability Maximization with Energy Constraint (RMEC) algorithm, which incorporates three important phases, including task priority establishment, frequency selection, and processor assignment. The central idea of RMEC is the reliability maximum energy (RME) conservative strategy. In brief, it selects a combination of processor and frequency that can maximize the RME function (which considers both energy and reliability), and then assigns such processor and frequency to the current task. Although we solve the same problem, our basic idea is to decompose it into two sub-problems and adopts the weighted preassignment mechanism. Our method is clearly different from theirs, and it achieved a better performance, as validated in our experiments.

As for the deadline constraint, there are also many works but they are more or less different from ours. For example, the work [11] proposed the deadline, reliability, resource-aware (DDR) algorithm to achieve minimum resources while satisfying the deadline and reliability constraints. Zhao *et al.* [50] presented an efficient method to maximize the reliability under deadline and energy constraints on a single processor. In contrast, our paper is interested in such a problem on heterogeneous distributed systems. Xie *et al.* [12] studied the problem of maximizing the reliability under deadline constraint, and proposed FFSV2 algorithm. Yet, their work did not consider the energy consumption constraint. This article is a full version of the preliminary work [1].

TABLE 1
Main notations and their descriptions

Notation	Definition
$c_{\{i,j\}}$	Communication time between task n_i and n_j
$w_{\{i,k\}}$	Execution time of task n_i running on the processor u_k with the maximum frequency
$pred(n_i)$	The set of direct predecessor tasks of task n_i
$succ(n_i)$	The set of direct successor tasks of task n_i
$\zeta_{\{k,h\}}$	The failure rate per time unit of processor u_k with frequency $f_{\{k,h\}}$
$EST(n_i, u_k)$	The earliest start time of task n_i executed on processor u_k
$EFT(n_i, u_k, f_{\{k,h\}})$	The earliest finish time of task n_i executed on processor u_k with frequency $f_{\{k,h\}}$
$LFT(n_i, u_k)$	The latest finish time of task n_i executed on processor u_k
$E(n_i, u_k, f_{\{k,h\}})$	The energy consumption of task n_i executed on processor u_k with frequency $f_{\{k,h\}}$
$R(n_i, u_k, f_{\{k,h\}})$	The reliability of task n_i executed on processor u_k with frequency $f_{\{k,h\}}$
$u_{pr(i)}$	The processor assigned to task n_i
$f_{\{pr(i),hz(i)\}}$	The frequency assigned to task n_i on processor $u_{pr(i)}$
$E_{given}(n_i)$	Energy consumption constraint of task n_i
$E_{pre}(n_i)$	The preassigned energy consumption for task n_i
$E_{given}(G)$	Given energy consumption constraint of application G
$D_{given}(G)$	Given deadline constraint of application G
$E(G)$	The total energy consumption of application G
$SL(G)$	The final schedule length of application G
$R(G)$	The final reliability of application G

3 PRELIMINARIES

In this section, we first introduce the models (Section 3.1), and then describe the problem formally (Section 3.2). Finally, we briefly introduce the method closest to ours, and reveal its limitation (Section 3.3). For ease of reference, Table 1 summarizes the main notations.

3.1 Models

Following prior works [23, 26, 35, 36, 51], 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 task n_i and n_j ; (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 [26, 52]. 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 (same to [26, 52], 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 the 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 $P_{\{k,h\}} = P_{\{k,ind\}} + C_{\{k,ef\}} \times (f_{\{k,h\}})^{m_k}$.

3.2 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, its earliest start time (EST) is denoted as $EST(n_i, u_k)$, which is computed as

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

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, and $c_{\{i,j\}}$ refers to the communication time between task n_i and n_j. In this paper, we assume $c_{\{i,j\}} = 0$ when n_i and n_j are assigned to the same processor.

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

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

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 $SL(G) = AFT(n_{exit})$; 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

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

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

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

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

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

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

3.3 The MSLECC Method

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

► *A brief review to the MSLECC algorithm.* The MSLECC algorithm is proposed in [26], and it 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; (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 [38]. It is computed as

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

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. For ease of presentation, let $\mathbb{S}_{have} = \{n_{s(1)}, n_{s(2)}, ..., n_{s(j-1)}\}$ denote the set of tasks that have been assigned, and let $\mathbb{S}_{not} = \{n_{s(j+1)}, n_{s(j+2)}, ..., n_{s(|N|)}\}$ denote the set of tasks that are unassigned. Then, when

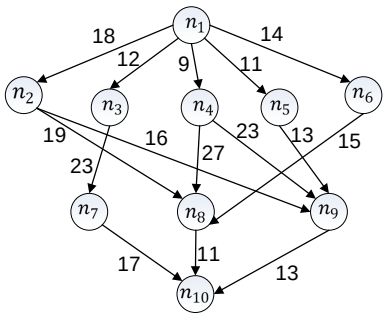


Fig. 1. An example of the DAG application model.

scheduling the task $n_{s(j)}$, the energy consumption of application G is computed as

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

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

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

then the actual energy consumption $E(G) \leq E_{given}(G)$ (cf., Formula 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

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

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

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

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.

TABLE 2
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 3
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

► *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

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

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 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 to $E_{given}(G) = E_{max}(G) \times 0.5 = 80.995$, and the parameters of all processors are shown in Table 2. 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 3 shows the execution time of each task on three processors with maximum frequency, and Table 4 shows part of scheduling results, which are ordered by the priority from highest to lowest.

One can see from Table 4 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 consume low energy (since the available energy consumption is little). This leads to less chance to achieve an 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 scheme that allows us to obtain smaller schedule length while satisfying the energy consumption constraint. This is just the focus of our paper.

TABLE 4
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
$E(G) = 80.98, SL(G) = 129.3660$			

4 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 4.1), and then prove that such a preassignment scheme can always satisfy the energy consumption constraint (Section 4.2). Finally, Section 4.3 presents the improved scheduling approach for energy consumption constrained parallel applications (ISAECC).

4.1 Preassigning Energy Consumption

We first present several concepts, which are helpful 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 ensuring 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.

4.2 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

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

and $E_{wa}(n_{s(1)}) \ge E_{min}(n_{s(1)})$. 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

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

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,

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

The above formulation can be written as

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

TABLE 5
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
$E(G) = 75.3619, SL(G) = 86.4233$							

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

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

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. (14) and (15), 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:

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

Putting all together, hence Theorem 1 holds. ■

4.3 The Proposed Algorithm for Minimizing Schedule Length

Our approach (i.e., ISAECC) for solving the problem of minimizing the schedule length 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, selecting the processor with the minimum EFT has complexity $O(|N| \times |U| \times |F|)$, and traversing all tasks consumes $O(|N|)$ time. Thus, the total time complexity 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)

```
1: Sort tasks in a list dl by descending order of rank_u;
2: for (∀i, n_i ∈ N) do
3:    Compute E_min(n_i) and E_max(n_i); // Eqs. 6 and 7
4:    Compute E_ave(n_i);
5: Compute E_min(G) and E_max(G); // Eqs. 4 and 5
6: Compute E_ave(G);
7: for (∀i, n_i ∈ N) do
8:    Compute E_pre(n_i); // Eq. 14
9: while (dl ≠ ∅ ) do
10:   n_i = dl.out();
11:   AFT(n_i) = ∞;
12:   Compute E_given(n_i); // Eq. 10
13:   for each u_k ∈ U do
14:      for each f_{k,h} ∈ [f_{k,low}, f_{k,max}] do
15:         Compute E(n_i, u_k, f_{k,h}); // Eq. 1
16:         if E(n_i, u_k, f_{k,h}) > E_given(n_i) then
17:            continue;
18:         Compute EFT(n_i, u_k, f_{k,h}); // Eq. 2
19:         if (EFT(n_i, u_k, f_{k,h}) < AFT(n_i)) then
20:            Let u_{pr(i)} = u_k and f_{pr(i),hz(i)} = f_{k,h};
21:            E(n_i, u_{pr(i)}, f_{pr(i),hz(i)}) = E(n_i, u_k, f_{k,h});
22:            AFT(n_i) = EFT(n_i, u_k, f_{k,h});
23: Compute actual energy consumption E(G); // Eq. 3
24: Compute the schedule length SL(G) = AFT(n_exit);
25: return E(G), SL(G)
```

► *The running example.* We still consider the application example described in Fig 1. For a fair comparison, all parameters are the same as in Section 3.3. Table 5 shows the task scheduling results generated by ISAECC.

From Table 5, 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 4, the "extra energy" of different tasks with different priorities does not show large differences. For intuition, Fig. 2 depicts the scheduling Gantt chart in which the arrows represent the communication message between tasks. The example above implies that our method should be effective and relatively fair for all tasks.

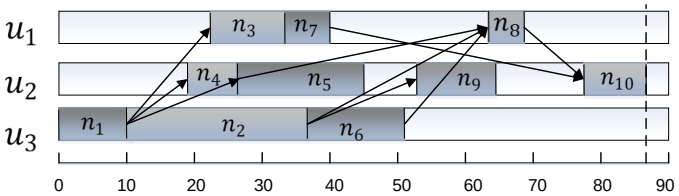


Fig. 2. Scheduling Gantt chart of application in Fig.1 using ISAECC

5 EXTENSIONS

In this section we extend the main idea presented in the previous section to solve other scheduling problems on heterogeneous computing systems : (i) maximizing the reliability under energy consumption constraint (Section

5.1); and (ii) maximizing the reliability under deadline and energy consumption constraints (Section 5.2). For short, we call them MRECC and MRDECC problems, respectively.

5.1 MRECC Problem

This problem involves with three models: (i) application model, (ii) power model, and (iii) reliability model. Since the former two are as same as that in Section 3.1, we mainly introduce the reliability model, followed by describing the problem formally (Section 5.1.1), and then we cover our proposed approach (Section 5.1.2).

5.1.1 Reliability Model and Problem Formulation

In a real system, the transition faults in the task execution phase are unpredictable and inevitable, and they often follow the probability distribution. A commonly used model [27, 28, 29, 39, 53] for the transient failure of a processor is the Poisson distribution with a parameter ζ . Let ζ_k denote the failure rate per time unit of the processor u_k. Then, the reliability of task n_i (executed on u_k with the maximum frequency in its execution phase) can be calculated as

$$R(n_i, u_k) = e^{-\zeta_k \times w_{\{i,k\}}} \tag{18}$$

For a DVFS-capable system, considering the effect of dynamic frequency scaling on transient faults, the average rate depends on the actual processing frequency [27, 29]. That is, different frequencies have different failure rates. Let $\zeta_{\{k,max\}}$ denote the failure rate per time unit of processor u_k with the maximum frequency. Then, the failure rate per time unit of u_k with the frequency $f_{\{k,h\}}$ is defined as

$$\zeta_{\{k,h\}} = \zeta_{\{k,max\}} \times 10^{\frac{d(f_{\{k,max\}} - f_{\{k,h\}})}{f_{\{k,max\}} - f_{\{k,min\}}}} \tag{19}$$

where $d > 0$ denotes the sensitivity of failure rates to voltage scaling.

According to Eqs. 18 and 19, one can compute the reliability of task n_i executed on the processor u_k with frequency $f_{\{k,h\}}$ as follows:

$$\begin{aligned} R(n_i, u_k, f_{\{k,h\}}) &= e^{-\zeta_{\{k,h\}} \times \frac{w_{\{i,k\}} \times f_{\{k,max\}}}{f_{\{k,h\}}}} \\ &= e^{-\zeta_{\{k,max\}} 10^{\frac{d(f_{\{k,max\}} - f_{\{k,h\}})}{f_{\{k,max\}} - f_{\{k,min\}}}} \times \frac{w_{\{i,k\}} \times f_{\{k,max\}}}{f_{\{k,h\}}}} \end{aligned} \tag{20}$$

Eq. 20 indicates that, in a processor the relationship between reliability and frequency is monotonically increasing. Then, for an application G in a DVFS-capable system, its reliability, denoted by $R(G)$, is computed as

$$R(G) = \prod_{i=1}^{|N|} R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) \tag{21}$$

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

Problem statement. The scheduling problem to be addressed in this section is to find a proper processor and frequency for each task such that we can (i) maximize *the reliability of the application* G, i.e., $R(G)$, and (ii) ensure *the actual energy consumption of* G, i.e., $E(G)$, is no larger than its given energy consumption constraint $E_{given}(G)$, namely, $E(G) \leq E_{given}(G)$ (cf., Formula 3) can be satisfied.

5.1.2 Our Approach for MRECC Problem

It can be seen from the problem description that, the problem contains two parts: satisfying energy consumption constraint, and maximizing reliability. Naturally, as similar as that in Section 4.3, we can also process it by decomposing it into two sub-problems, and then solve them respectively, and finally integrate them. As for the first sub-problem, it can be also solved based on the weight mechanism, since the sub-problem here is highly similar to that in Section 4. For the second sub-problem, our basic idea is to choose the processor-frequency pair (which satisfies the energy constraint) with the maximum $R(G)$ value for each task.

Algorithm 2 shows the pseudo-codes of our approach (called ISAECC*) for solving the problem of maximizing reliability under the energy consumption constraint. Firstly, it sorts the tasks in a list dl (Line 1), and then computes some basic information, e.g., $E_{min}(n_i)$ and $E_{max}(G)$, based on the equations described earlier (Lines 2-8). Next, it traverses all processors and frequencies to select a proper processor-frequency pair with maximum reliability for each task (Lines 9-23). Note that, we here need to traverse the frequencies of a processor "from f_{max} to f_{min}" (Line 14), and stop to traverse the remaining lower frequencies once a proper frequency is found (Line 23). This is because the lower frequency easily leads to low reliability (recall Eq. 20). The rest of steps are to compute $E(G)$ and $R(G)$ based on the processor-frequency pairs found before, and finally we return the results (Lines 24-26).

Algorithm 2 The ISAECC* Algorithm

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

1: Sort tasks in a list dl by descending order of $rank_u$;
2: **for** $(\forall i, n_i \in N)$ **do**
3: Compute $E_{min}(n_i)$ and $E_{max}(n_i)$; // Eqs. 6 and 7
4: Compute $E_{ave}(n_i)$;
5: Compute $E_{min}(G)$ and $E_{max}(G)$; // Eqs. 4 and 5
6: Compute $E_{ave}(G)$;
7: **for** $(\forall i, n_i \in N)$ **do**
8: Compute $E_{pre}(n_i)$; // Eq. 14
9: **while** (dl is not empty) **do**
10: $n_i = dl$.out();
11: $R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) = 0$;
12: Compute $E_{given}(n_i)$; // Eq. 10
13: **for** each $u_k \in U$ **do**
14: **for** each $f_{\{k,h\}}$ from $f_{\{k,max\}}$ to $f_{\{k,low\}}$ **do**
15: Compute $E(n_i, u_k, f_{\{k,h\}})$; // Eq. 1
16: **if** $E(n_i, u_k, f_{\{k,h\}}) > E_{given}(n_i)$ **then**
17: continue;
18: Compute $R(n_i, u_k, f_{\{k,h\}})$; // Eq. 20
19: **if** $(R(n_i, u_k, f_{\{k,h\}}) > R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}))$ **then**
20: Let $u_{pr(i)} = u_k$ and $f_{\{pr(i),hz(i)\}} = f_{\{k,h\}}$;
21: $E(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) = E(n_i, u_k, f_{\{k,h\}})$;
22: $R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) = R(n_i, u_k, f_{\{k,h\}})$;
23: break; //skip the lower frequencies
24: Compute actual energy consumption $E(G)$; // Eq. 3
25: Compute $R(G)$; //Eq. 21
26: **return** $E(G)$, $R(G)$

TABLE 6
Power parameters of processors

u_k	$P_{\{k,ind\}}$	$C_{\{k,ef\}}$	m_k	$f_{\{k,low\}}$	$f_{\{k,max\}}$	$\zeta_{\{k,max\}}$
u_1	0.03	0.8	2.9	0.26	1.0	0.00015
u_2	0.04	0.7	2.5	0.27	1.0	0.00020
u_3	0.07	1.0	2.5	0.29	1.0	0.00025

TABLE 7
RMEC vs. ISAECC* using the application in Fig. 1

n_i	$u(n_i)$		$f(n_i)$		$E(n_i)$		$R(n_i)$	
	RMEC	ISAECC*	RMEC	ISAECC*	RMEC	ISAECC*	RMEC	ISAECC*
n_1	u_1	u_1	0.26	0.65	2.4817	5.5865	0.9224	0.9904
n_3	u_1	u_1	0.26	0.86	1.9499	6.9911	0.9385	0.9970
n_4	u_2	u_2	0.27	1.0	1.9708	5.9200	0.9425	0.9984
n_2	u_1	u_1	0.26	0.83	2.3044	7.7692	0.9277	0.9960
n_5	u_1	u_1	0.26	0.67	2.1272	5.0228	0.9331	0.9925
n_6	u_1	u_1	0.26	0.67	2.3044	5.4414	0.9227	0.9919
n_9	u_2	u_2	0.27	0.96	2.9563	8.4011	0.9149	0.9972
n_7	u_1	u_1	0.26	0.83	1.2408	4.1834	0.9604	0.9979
n_8	u_1	u_1	0.26	1.0	0.8863	4.1500	0.9716	0.9993
n_{10}	u_2	u_2	0.27	1.0	1.7245	5.1800	0.9495	0.9986
RMEC: E(G)=19.9463, R(G)=0.5312 ISAECC*: E(G)=58.6455, R(G)=0.9599								

Theorem 3. The time complexity of the ISAECC* Algorithm is $O(|N|\times|U|\times|F|)$, where $|F|$ represents the maximum number of discrete frequencies from $f_{\{k,low\}}$ to $f_{\{k,max\}}$.

Proof. For each task, selecting the processor with the maximum reliability has complexity $O(|U|\times|F|)$, and traversing all tasks needs $O(|N|)$ time. Thus, the total time complexity is $O(|N|\times|U|\times|F|)$. ■

Remark. One can also extend the MSLECC algorithm in [26] to solve the problem of maximizing reliability. For clearness, we call it the MSLECC* algorithm. It can be done as follows: (i) compute upward rank values to determine the order of task scheduling; (ii) for each task, traverse all processors and frequencies to choose the processor-frequency pairs that satisfy the energy consumption; (iii) choose the processor-frequency pair with maximum $R(G)$ value from the pairs obtained in Step (ii).

► *The running example.* We still use the application described in Fig. 1 as the running example. The energy consumption constraint is set as $E_{given}(G) = 3\times E_{min}(G) = 59.8390$, and the parameters of all processors are listed in Table 6. Table 7 shows the scheduling results using the RMEC algorithm [27] and our proposed algorithm — ISAECC*. It can be seen that RMEC consumes much smaller energy than the given energy constraint, and its energy consumption is less than our algorithm ISAECC*. Nevertheless, the reliability values obtained by RMEC are not optimistic. It seems that it does not fully utilize the given energy consumption to maximize the reliability. This result implies that, RMEC may not actually for maximizing the reliability, although the authors attempted to maximize the reliability in their paper. Later (in Section 6), we will use more real parallel applications to investigate their performances.

Additionally, Table 8 shows the scheduling results using the algorithm MSLECC* and ISAECC*. It can be seen that the total energy consumption generated by ISAECC* is less than the given energy constraint, and also less than the value obtained by MSLECC*. On the other hand, the final reliability value $R(G)$ is 0.9599, which is higher than 0.82 obtained by MSLECC*. These results imply that both algorithms can satisfy the energy consumption constraints, while our proposed algorithm ISAECC* has a better performance in terms of reliability.

5.2 MRDECC Problem

Compared to the problem discussed in Section 5.1, the MRDECC problem involves with another additional constraint, i.e., deadline constraint. In this section, we first briefly introduce the concepts and notations related to this problem, followed by describing the MRDECC problem formally (Section 5.2.1), and then we cover our approach for the MRDECC problem (Section 5.2.2).

5.2.1 Concepts and Problem Formulation

Given a parallel application G, we use $D_{given}(G)$ to denote the given deadline constraint.

Definition 8. Given a task n_i executed on processor u_k, its latest finish time (LFT) is denoted as $LFT(n_i, u_k)$, which can be computed as

$$\begin{cases} LFT(n_{exit},u_k) = D_{given}(G) \\ LFT(n_i,u_k) = \min\left(AE[k], \min\limits_{n_j\in succ(n_i)}\{AST(n_j) - c_{\{i,j\}}\}\right), \end{cases}$$

where $AE[k]$ is the end of the available time that processor u_k can execute task n_i. It means that u_k would be assigned to another task after $AE[k]$. In addition, $AST(n_i)$ denotes the actual start time of task n_i, and $c_{\{i,j\}}$ has the same meaning with that in Definition 1.

Problem statement. The scheduling problem to be addressed in this section is to find a proper processor and frequency for each task in application G such that we can (i) maximize *the reliability of the application* G, i.e., $R(G)$ (cf., Formula 21); and (ii) ensure the following two conditions can be satisfied: (a) *the actual energy consumption of* G, i.e., $E(G)$, is no larger than its given energy consumption constraint $E_{given}(G)$, namely, $E(G) \le E_{given}(G)$ (cf., Formula 3); and (b) *the final schedule length of* G, i.e., $SL(G)$, does not exceed the given deadline constraint $D_{given}(G)$, namely,

$$SL(G) = AFT(n_{exit}) - AST(n_{entry}) \le D_{given}(G) \quad (22)$$

5.2.2 Our Approach for MRDECC Problem

At a high level, our approach mainly consists of two major phases. In the first phase, we use the ISAECC algorithm presented in Section 4.3 to get a preliminary scheduling result. In the second phase, we "reclaim" the *slack time* between the deadline constraint and the schedule length to reallocate the tasks, in order to improve the reliability. In the process of reallocation, we handle tasks from "exit task" to "entry task". The reason we handle tasks in this way is that the latest finish time of the exit task, $LFT(n_{exit})$, is available after we execute the ISAECC algorithm. We remark that, for the first phase, we cannot simply replace ISAECC with ISAECC* presented in Section 5.1, this is because the latter cannot guarantee the schedule length constraint. In what follows, we focus our attention on the second phase of our approach.

Definition 9. Given a task n_i executed on a processor u_k, its maximum slack time (MST), denoted by $MST(n_i, u_k)$, is computed as

$$MST(n_i, u_k) = LFT(n_i, u_k) - EST(n_i, u_k) \quad (23)$$

TABLE 8
MSLECC* vs. ISAECC* using the application in Fig. 1

n_i	$E_{given}(n_i)$		$u(n_i)$		$f(n_i)$		$E(n_i)$		$R(n_i)$	
	MSLECC*	ISAECC*	MSLECC*	ISAECC*	MSLECC*	ISAECC*	MSLECC*	ISAECC*	MSLECC*	ISAECC*
n_1	11.84	5.6971	u_1	u_1	1.0	0.65	11.62	5.5865	0.9979	0.9904
n_3	20.33	7.0626	u_1	u_1	1.0	0.86	9.13	6.9911	0.9984	0.9970
n_4	18.19	6.5687	u_2	u_2	1.0	1.0	5.9200	5.9200	0.9984	0.9984
n_2	19.26	7.7946	u_1	u_1	1.0	0.83	10.79	7.7692	0.9981	0.9960
n_5	10.7	5.0324	u_1	u_1	1.0	0.67	9.96	5.0228	0.9982	0.9925
n_6	5.611	5.4895	u_1	u_1	0.68	0.67	5.5716	5.4414	0.9923	0.9919
n_9	2.9958	8.4727	u_2	u_2	0.30	0.96	2.9803	8.4011	0.9298	0.9972
n_7	1.2563	4.2335	u_1	u_1	0.28	0.83	1.2486	4.1834	0.9654	0.9979
n_8	0.8940	4.4986	u_1	u_1	0.28	1.0	0.8919	4.1500	0.9751	0.9993
n_{10}	1.7266	6.3735	u_2	u_2	0.28	1.0	1.7260	5.1800	0.9527	0.9986
MSLECC*: E(G)=59.8384, R(G)=0.8200 ISAECC*: E(G)=58.6455, R(G)=0.9599										

where $EST(n_i, u_k)$ (cf., Definition 1) and $LFT(n_i, u_k)$ (cf., Definition 8) refer to the earliest start time and latest finish time of task n_i executed on processor u_k, respectively.

Definition 10. Given a task n_i running on processor u_k with frequency $f_{\{k,h\}}$, its execution time, denoted by $w_{\{i,k,h\}}$, is computed as

$$w_{\{i,k,h\}} = w_{\{i,k\}} \times \frac{f_{\{k,max\}}}{f_{\{k,h\}}} \tag{24}$$

where $w_{\{i,k\}}$ (cf., Table 1) denotes the execution time of task n_i running on processor u_k with the maximum frequency $f_{\{k,max\}}$.

Specifically, we reallocate each task n_i in order to obtain the large reliability, $R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}})$, as far as possible. Our idea is to traverse the processors and frequencies to seek the most proper "processor-frequency" combination. Formally, it can be formulated as follows.

$$R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) = \max\{R(n_i, u_k, f_{\{k,h\}})\} \tag{25}$$

where $u_k \in U$ and $f_{\{k,low\}} \leq f_{\{k,h\}} \leq f_{\{k,max\}}$. Meanwhile, in the reallocation we need to check two conditions:

• The execution time $w_{\{i,k,h\}}$ should be smaller than the maximum slack time, namely, $w_{\{i,k,h\}} \leq MST(n_i, u_k)$.

• The energy consumption $E(n_i, u_k, f_{\{k,h\}})$ should satisfy the constraint, namely,

$$\begin{aligned} E(n_i, u_k, f_{\{k,h\}}) \leq & E_{given}(n_i) \\ = & E_{given}(G) - \sum_{x=1}^{i-1} E(n_x, u_{pr(x)}, f_{\{pr(x),hz(x)\}}) \\ & - \sum_{y=i+1}^{|N|} E(n_y, u_{pr(y)}, f_{\{pr(y),hz(y)\}}) \end{aligned} \tag{26}$$

where $\sum_{x=1}^{i-1} E(n_x, u_{pr(x)}, f_{\{pr(x),hz(x)\}})$ represents the energy consumption of tasks that have not been reassigned, $\sum_{y=i+1}^{|N|} E(n_y, u_{pr(y)}, f_{\{pr(y),hz(y)\}})$ represents the energy consumption of tasks that have been reassigned.

Algorithm 3 shows the pseudo-codes of our approach for solving the MRDECC Problem. Firstly, it uses the ISAECC algorithm to obtain a preliminary scheduling result (Line 1). Then, it reallocates tasks in order to achieve a large reliability (Lines 2-16). Note that, different from that in Algorithms 1 and 2, we here employ a list $dl2$ in which the tasks are ranked by ascending order of $rank_u$ (Line 2), since we need to handle tasks from exit task to entry task. In addition, Line 5 is used to obtain the new energy consumption constraint. Lines 6-16 are to select processor and frequency with maximum reliability for each task based on the maximum slack time. Then, it computes the final reliability, energy consumption and schedule length (Line 17), and finally returns the results.

Algorithm 3 The MRDECC Algorithm

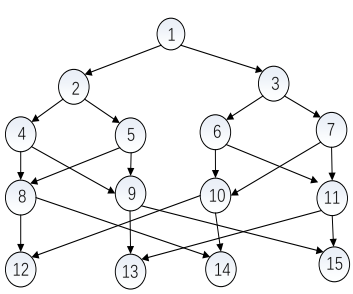
Input: G=(N,M,C,W),U,$E_{given}(G)$, $D_{given}(G)$;
Output: R(G),SL(G),E(G)
1: Execute the ISAECC algorithm;
2: Sort tasks in a list $dl2$ by ascending order of $rank_u$;
3: **while** ($dl2 \neq \emptyset$) **do**
4: $n_i = dl2$.out();
5: Compute $E_{given}(n_i)$; // Eq. 26
6: **for** each $u_k \in U$ **do**
7: compute $MST(n_i, u_k)$; // Eq. 23
8: **for** each $f_{\{k,h\}} \in [f_{\{k,low\}}, f_{\{k,max\}}]$ **do**
9: Compute $E(n_i, u_k, f_{\{k,h\}})$ and $w_{\{i,k,h\}}$; // Eqs. 1 and 24
10: **if** $E(n_i, u_k, f_{\{k,h\}}) \leq E_{given}(n_i)$ and $w_{\{i,k,h\}} \leq MST(n_i, u_k)$ **then**
11: Compute $R(n_i, u_k, f_{\{k,h\}})$; // Eq. 20
12: **if** ($R(n_i, u_k, f_{\{k,h\}}) > R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}})$) **then**
13: Let $u_{pr(i)} = u_k$ and $f_{\{pr(i),hz(i)\}} = f_{\{k,h\}}$;
14: $E(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) = E(n_i, u_k, f_{\{k,h\}})$;
15: $R(n_i, u_{pr(i)}, f_{\{pr(i),hz(i)\}}) = R(n_i, u_k, f_{\{k,h\}})$;
16: break; //skip the lower frequencies
17: Compute $R(G), E(G)$ and $SL(G)$; //Eqs. 21, 3 and 26
18: **return** $R(G)$,$E(G)$, $SL(G)$

Theorem 4. The time complexity of the MRDECC Algorithm is $O(|N|^2 \times |U| \times |F|)$, where $|F|$ denotes the maximum number of discrete frequencies from $f_{\{k,low\}}$ to $f_{\{k,max\}}$.

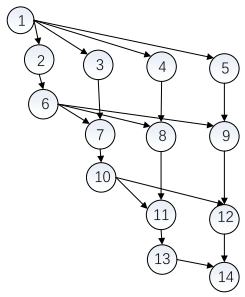
Proof. By Theorem 2, Line 1 takes $O(|N|^2 \times |U| \times |F|)$ time. For each task, computing MST and selecting the processor and frequency take $O(|N| \times |U| \times |F|)$ time. In addition, there are $O(N)$ tasks needing to be traversed. Thus, the time complexity for reallocating is $O(|N|^2 \times |U| \times |F|)$. Putting all together, the time complexity of MRDECC algorithm is $O(|N|^2 \times |U| \times |F|)$. ■

TABLE 9
task assignment of application in Fig.1 using MRDECC

n_i	$E_{given}(n_i)$	$u(n_i)$	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$	$R(n_i)$
n_1	7.68	u_3	0.84	1.52	12.23	7.68	0.9955
n_3	10.18	u_2	0.95	28.32	42.0	10.18	0.9968
n_4	7.77	u_2	1.0	42.0	50.0	6.72	0.9984
n_2	11.84	u_1	1.0	30.23	43.23	10.79	0.9981
n_5	9.00	u_1	0.94	43.23	56.0	8.92	0.9977
n_6	13.20	u_1	1.0	56.0	69.0	10.79	0.9981
n_9	15.71	u_2	1.0	81.0	93.0	10.08	0.9976
n_7	11.44	u_1	1.0	69.0	76.0	5.81	0.9990
n_8	9.78	u_1	1.0	77.0	82.0	4.15	0.9992
n_{10}	11.51	u_2	1.0	93.0	100.0	5.88	0.9986
$R(G) = 0.9791, E(G) = 80.99, SL(G) = 98.48$							



(a) FFT appl. with $\rho = 4$



(b) GE appl. with $\rho = 5$

Fig. 3. Example of real parallel applications.

► *The running example.* In this section, we still consider the parallel application described in Fig. 1 as the running example. The parameters related to processors are the same as that in Section 3.3 and Table 6. The deadline constraint $D_{given}(G)$ is set to 100, and the energy consumption constraint $E_{given}(G)$ is set to 80.995, which is the same as that in section 3.3. Table 9 shows the scheduling results.

We can see that the final schedule length is 98.48, which is smaller than the deadline constraint $D_{given}(G) = 100$. And the actual energy consumption is 80.99, which is under the energy consumption constraint $E_{given} = 80.995$. In addition, the final reliability is 0.9791, which is higher than the reliability value 0.9531 achieved by ISAECC algorithm. This is because MRDECC algorithm further enhances the reliability by using the maximum slack time between the deadline constraint and schedule length obtained by ISAECC.

6 EXPERIMENTS

In this section, we first describe the experimental settings (Section 6.1), and then cover the experimental results (Sections 6.2∼6.4).

6.1 Experimental Settings

In our experiments, we select two real parallel applications for tests: (i) fast Fourier transform (FFT), and (ii) Gaussian elimination (GE). Fig. 3 (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. 3 (a). In order to match the application model (recall Section 3.1), 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. 3 (b) shows a GE parallel application example with $\rho = 5$.

The simulated heterogeneous platform for testing the problem of minimizing the schedule length uses 64 processors. In addition, we set $10ms \leq w_{\{i,k\}} \leq 100ms$, $10ms \leq c_{\{i,j\}} \leq 100ms$. For the problem of maximizing the reliability, the simulated heterogeneous platform uses 128 processors, and we set $10h \leq w_{\{i,k\}} \leq 100h$, $10h \leq c_{\{i,j\}} \leq 100h$, $0.0000001 \leqslant \gamma_k \leqslant 0.0000128$. For the problem of maximizing reliability with two constraints (i.e., MRDECC problem), the simulated platform uses 64 processors, and we set $10ms \leq w_{\{i,k\}} \leq 100ms$, $10ms \leq c_{\{i,j\}} \leq 100ms$, $0.0000001 \leqslant \gamma_k \leqslant 0.0000064$. Other parameters related to applications and processors are: $0.03 \leq P_{\{k,ind\}} \leq 0.07$, $0.8 \leq C_{\{k,ef\}} \leq 1.2$, $2.5 \leq m_k \leq 3.0$, and $f_{\{k,max\}} = 1.0$ GHz. The frequency precision is 0.01 GHz. For ease of observing the effectiveness of our proposed algorithms, in our experiments we vary the sizes of energy consumption constraints and the scales of applications, respectively. Particularly, for the MRDECC problem, we also vary the deadline constraint D_{given}. All these main parameter settings basically follow prior works in this field.

As for the problem of minimizing the schedule length, we compare our algorithm ISAECC with HEFT [38], ECS [23] and MSLECC [26], since they have the same model. The HEFT is a precedence-constrained application scheduling algorithm for minimizing the schedule length on heterogeneous systems, and it does not consider the power consumption constraint. The ECS considers both the energy consumption and the scheduling length, while it focuses more attention on the trade-off between the schedule length and energy consumption. In contrast, the MSLECC is closest to our algorithm, since both methods solve the same problem with the same constraints. Following prior works [26], we use the actual energy consumption $E(G)$ and schedule length $SL(G)$ as the performance metrics for this problem.

On the other hand, for the reliability maximization problem, we compare our algorithm ISAECC* with RMEC [27] and MSLECC*, since all of them have the same model and solve the same problem. Here MSLECC* is a method adapted from [26], recall the "Remark" in Section 5.1.2. Similar to the first problem, we here use the actual energy consumption $E(G)$ and reliability $R(G)$ as the performance metrics. In addition, for the problem of maximizing reliability with two constraints, we compare our algorithm MRDECC with HEFT [38], ISAECC (Section 4.3), and FFSV2 [12]. For this problem, we use $E(G)$, $R(G)$, and $SL(G)$ as

TABLE 10
Scheduling results of FFT application by varying $E_{given}(G)$

$E_{given}(G)$	HEFT		ECS		MSLECC		ISAECC	
	E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)
1544	4850	736	2033	999	1544	1195	1544	856
2090	5121	772	2264	1057	2090	1127	2089	832
2704	5049	747	2057	1028	2704	1057	2699	837
3178	5170	743	2063	1034	3178	1039	3171	775
4162	4795	753	2271	1045	4162	1024	4093	764

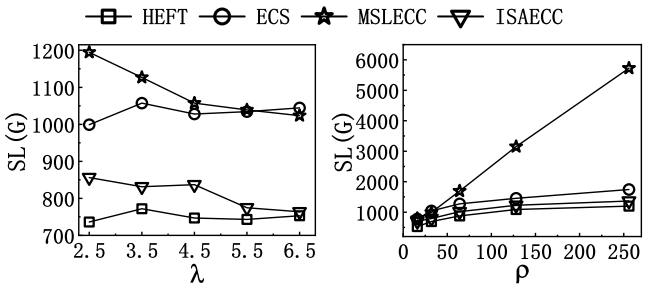


Fig. 4. The schedule lengths of FFT parallel application. (a) varying λ (essentially, the energy consumption constraint $E_{given}(G)$); (b) varying ρ (essentially, the scale of application).

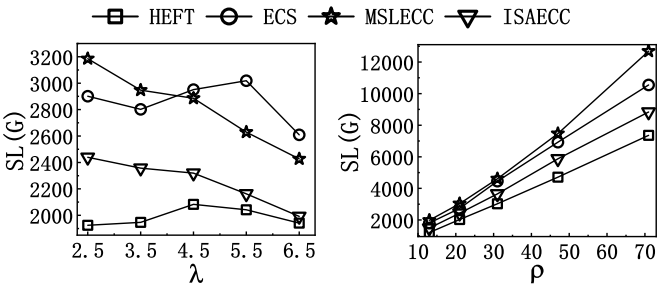


Fig. 5. The schedule lengths of GE parallel application. (a) varying λ; (b) varying ρ.

TABLE 11
Scheduling results of FFT application with different scales

ρ	$E_{given}(G)$	HEFT		ECS		MSLECC		ISAECC	
		E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)
16	899	2142	528	901	778	899	790	899	698
32	2272	4838	692	2394	1043	2272	991	2271	791
64	5056	10958	879	5519	1272	5056	1692	5056	1018
128	11053	22963	1088	12008	1457	11053	3152	11053	1226
256	25265	44492	1199	28926	1745	25265	5727	25265	1368

TABLE 13
Scheduling results of GE parallel application with different scales

ρ	$E_{given}(G)$	HEFT		ECS		MSLECC		ISAECC	
		E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)
13	903	1831	1201	997	1809	903	1978	902	1489
21	2116	4177	2028	2421	2732	2116	3035	2116	2427
31	4726	10076	3028	6534	4456	4726	4607	4726	3654
47	10416	21398	4712	14698	6925	10416	7466	10416	5870
71	24040	55984	7357	36554	10551	24040	12691	24039	8839

the performance metrics.

6.2 Experimental Results for Minimizing Schedule Length with Energy Consumption Constraint

Exp-1. In this experiment, we compare $E(G)$ and $SL(G)$ of the *FFT application* under *different* $E_{given}(G)$. The application size is set to $\rho = 32$ (i.e., $|N| = 223$). $E_{given}(G)$ is set to $E_{min}(G) \times \lambda$. We vary λ from 2.5 to 6.5. Table 10 shows the scheduling results. We can see that, although HEFT obtains the smaller schedule length, it exceeds the energy consumption constraint in each case. In addition, although the energy consumption generated by ECS is less than HEFT, it can not satisfy all constraints (e.g., when $E_{given}(G) = 1544$) and its schedule length is larger than that of HEFT. In contrast, MSLECC and ISAECC can always satisfy the given energy consumption constraint even if $E_{given}(G)$ is small. For the sake of intuition, Fig. 4(a) shows the schedule length when 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 ECS and MSLECC, our algorithm has the obvious advantage on the schedule length $SL(G)$; it outperforms MSLECC by about 20.8%∼28.3%, for example. Further, considering $SL(G)$ of MSLECC and that of ISAECC, we can find that their gap increases when $E_{given}(G)$ decreases. This is because the preassignment policy of MSLECC, together with the small $E_{given}(G)$, makes 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 compare $E(G)$ and $SL(G)$ of the *FFT application* with *different scales.* We fix λ to 3.5 (i.e., $E_{given}(G) = E_{min}(G) \times 3.5$), 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 11 shows the scheduling results. Similar to the results in Exp-1, although HEFT can obtain the smaller schedule length while it exceeds the energy consumption constraint in each case. As for ECS, it exceeds the energy consumption constraint for each case in this experiment. Also, both MSLECC and ISAECC can always satisfy the energy consumption constraint. On the other hand, Fig. 4(b) depicts the variation tendency of $SL(G)$. It can be seen that, our algorithm can achieve better schedule lengths than ECS and MSLECC. And for the MSLECC algorithm, its $SL(G)$ dramatically increases when the scale grows, while $SL(G)$ obtained by our algorithm only increases slightly. This essentially illustrates that our algorithm has the better scalability.

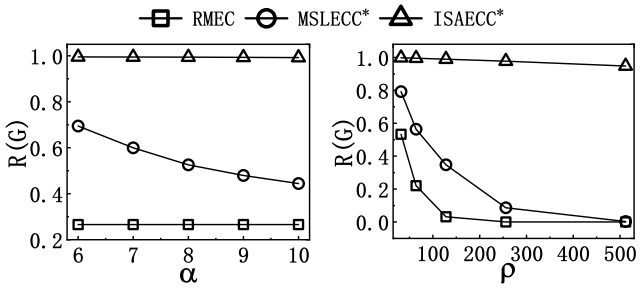
Exp-3. This experiment compares $E(G)$ and $SL(G)$ of *the GE parallel application* under *different* $E_{given}(G)$. The application size is set to $\rho = 21$ (i.e.,$|N|$=230), which is roughly equal to that in Exp-1, where we tested *the FFT parallel application*. We vary $E_{given}(G)$ from $E_{min}(G) \times 2.5$ to $E_{min}(G) \times 6.5$.

TABLE 12
Scheduling results of GE application by varying $E_{given}(G)$

$E_{given}(G)$	HEFT		ECS		MSLECC		ISAECC	
	E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)	E(G)	SL(G)
1579	4601	1925	2335	2902	1579	3185	1579	2439
2222	4343	1947	2547	2802	2222	2947	2221	2357
2971	4422	2083	2531	2951	2971	2886	2967	2320
3374	4648	2042	2562	3019	3374	2630	3371	2163
4004	4366	1941	2308	2610	4004	2427	3927	1992

TABLE 14
Scheduling results of FFT application by varying $E_{given}(G)$

$E_{given}(G)$	RMEC		MSLECC*		ISAECC*	
	E(G)	R(G)	E(G)	R(G)	E(G)	R(G)
10456	1183.9	0.2659	10456	0.6946	10400	0.9954
8962	1183.9	0.2659	8962	0.5996	9841	0.9948
7842	1183.9	0.2659	7842	0.5257	7813	0.9941
6970	1183.9	0.2659	6970	0.4797	6962	0.9932
6273	1183.9	0.2659	6273	0.4443	6269	0.9921



(a) (b)

Fig. 6. The reliability of FFT parallel application. (a)varying α; (b) varying ρ.

TABLE 15
Scheduling results of FFT application with different scales

ρ	$E_{given}(G)$	RMEC		MSLECC*		ISAECC*	
		E(G)	R(G)	E(G)	R(G)	E(G)	R(G)
32	4577	535	0.5331	4577	0.7923	4572	0.9979
64	10426	1195	0.2205	10426	0.5639	10397	0.9949
128	23651	2807	0.0317	23651	0.3471	23633	0.9893
256	51997	5957	3.341E-4	51997	0.0864	51983	0.9769
512	114811	13397	4.273E-8	114811	0.0028	114803	0.9479

Fig. 5(a) plots $SL(G)$ for different $E_{given}(G)$, and Table 12 shows the detailed scheduling results. Both of them indicate that ISAECC achieves better schedule lengths than ECS and MSLECC. Similar to that in Exp-1, MSLECC and ISAECC can always satisfy energy consumption constraints while HEFT and ECS cannot. On the other hand, combining Exp-3 and Exp-1, it shows that our solution is feasible for different types of applications.

Exp-4. This experiment uses *the GE parallel application* under *different scales*. We fix $E_{given}(G)$ to $E_{min}(G) \times 3.5$, 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, where we tested *the FFT parallel application*. Table 13 shows detailed scheduling results, and Fig. 5(b) plots the variation tendency of $SL(G)$ when varying ρ. Similar to that in Exp-2, the actual energy consumptions generated by ISAECC and MSLECC are still within the given constraints, and our algorithm can generate shorter schedule lengths, compared against the ECS and MSLECC algorithms. Additionally, by comparing Figs. 5(b) and 4(b), we find an interesting phenomenon. That is, when ρ increases, the schedule lengths obtained by ISAECC, ECS and HEFT increase *slightly* in Fig. 4(b), while they increase *dramatically* in Fig. 5(b). This phenomenon could be due to that the FFT parallel application has better parallelism than the GE parallel application.

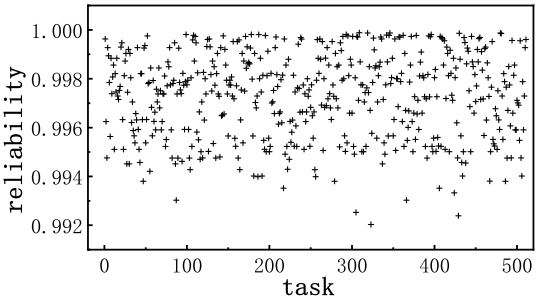
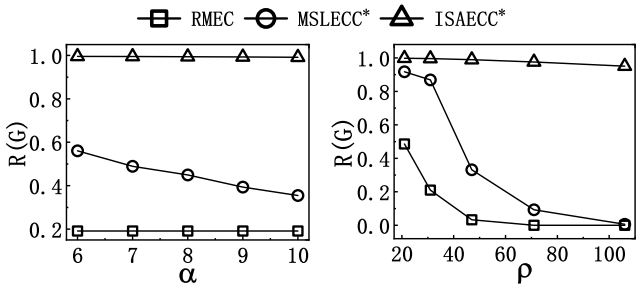


Fig. 7. Distribution of reliability values of all task sequences.



(a) varying α (b) varying ρ

Fig. 8. The reliability of GE parallel application.

TABLE 16
Scheduling results of GE application with $\rho = 31$.

$E_{given}(G)$	RMEC		MSLECC*		ISAECC*	
	E(G)	R(G)	E(G)	R(G)	E(G)	R(G)
10157	1188	0.1912	10157	0.5607	10101	0.9955
8706	1188	0.1912	8706	0.4894	8681	0.9948
7618	1188	0.1912	7618	0.4497	7605	0.9939
6771	1188	0.1912	6771	0.3936	6761	0.9929
6094	1188	0.1912	6094	0.3545	6089	0.9915

6.3 Experimental Results for for Maximizing Reliability with Energy Consumption Constraint

Exp-5. This experiment is to compare $E(G)$ and $R(G)$ of the *FFT application* using *different* $E_{given}(G)$. The application size is set to $\rho = 64$ (i.e.,$|N| = 511$); $E_{given}(G)$ is set to $(E_{min}(G) + E_{max}(G))/\alpha$; we vary α from 6 to 10. Table 14 shows the scheduling results while Fig. 6(a) plots the variation tendency of $R(G)$. We can see from Table 14 that, for all these three methods, the actual energy consumption $E(G)$ can always satisfy the given energy constraint $E_{given}(G)$. In particular, for each case $E(G)$ and $R(G)$ produced by RMEC are constant. These results indicate that RMEC is insensitive to the energy constraint, and RMEC essentially does not really maximize the reliability. As for its low reliability, it can be understood from Fig. 7 and Eq. 21. Notice that, Fig. 7 shows the distribution of reliability values of RMEC for all 511 tasks, and in Eq. 21 the notation $\prod$ means the multiplication of reliability values of $|N|$ tasks. On the other hand, we observe that the energy consumptions generated by MSLECC* and ISAECC* are both close to the given energy constraint $E_{given}(G)$, and the reliability values decrease when the given energy consumption constraints turn smaller. Nevertheless, our proposed method, ISAECC*, achieves much higher reliability than RMEC and MSLECC*. For example, it outperforms MSLECC* by about

TABLE 17
Scheduling results of GE application with $E_{given}(G) = (E_{min}(G) + E_{max}(G))/6$.

ρ	$E_{given}(G)$	RMEC		MSLECC*		ISAECC*	
		E(G)	R(G)	E(G)	R(G)	E(G)	R(G)
21	4707	544	0.4859	4707	0.9175	4699	0.9981
31	10114	1170	0.2104	10114	0.8688	10067	0.9961
47	22832	2675	0.0327	22831	0.3315	22828	0.9895
71	52405	6493	3.944E-4	52405	0.0925	52398	0.9759
106	115763	13324	1.455E-8	115763	0.0063	115762	0.9506

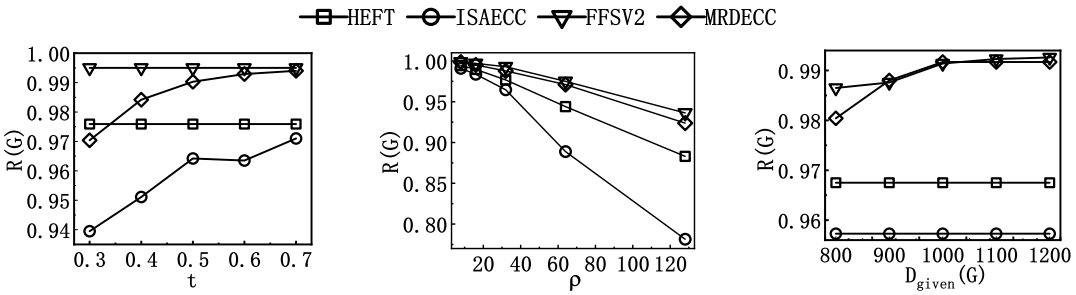


Fig. 9. The reliability of FFT parallel application. (a) varying t; (b) varying ρ; (c)varying $D_{given}(G)$

TABLE 18
Scheduling results of FFT application by varying $E_{given}(G)$

$E_{given}(G)$	HEFT			ISAECC			FFSV2			MRDECC		
	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)
1485.53	776	4951.77	0.9759	945.50	1485.49	0.9395	1147	4117.16	0.9950	1119.79	1485.52	0.9704
1980.71	776	4951.77	0.9759	867.35	1980.68	0.9511	1147	4117.16	0.9950	994.41	1980.69	0.9842
2475.88	776	4951.77	0.9759	843.25	2475.02	0.9642	1147	4117.16	0.9950	985.98	2475.84	0.9903
2971.06	776	4951.77	0.9759	864.14	2963.95	0.9635	1147	4117.16	0.9950	995.76	2963.41	0.9929
3466.24	776	4951.77	0.9759	848.98	3445.85	0.9710	1147	4117.16	0.9950	1087.13	3460.65	0.9940

43.3%∼123.2%. Also, we observe that the reliability are relatively stable when $E_{given}(G)$ varies, as shown in Fig. 6(a). These results essentially demonstrate the superiority of our approach.

Exp-6. In this experiment, we compare $E(G)$ and $R(G)$ of the *FFT application* using *different scales*. We fix α to 6 (i.e., $E_{given}(G) = (E_{min}(G) + E_{max}(G))/6$), then vary ρ from 32 (i.e., $|N| = 223$, small scale) to 512 (i.e.,$|N| = 5631$, large scale). Table 15 shows the scheduling results. We can see that all these methods can satisfy the energy consumption constraints, while our proposed method, ISAECC*, obtains the highest reliability, compared against the competitors. Fig. 6(b) plots the variation tendency of the reliability values when we vary ρ. The results show also that ISAECC* outperforms RMEC and MSLECC* in terms of the reliability. Meanwhile, we can see that, when the scale increases, the reliability values obtained by RMEC and MSLECC* turn smaller drastically. This is mainly because (i) $|N|$ (in Eq. 21) increases, and (ii) the reliability values of many tasks are not very close to 1. As a result, the final reliability value is relatively small. On the contrary, our proposed method, ISAECC*, performs stable and can consistently maintain a relatively high reliability, further demonstrating its superiorities.

Exp-7. This experiment compares $E(G)$ and $R(G)$ of the *GE application* under *different* $E_{given}(G)$. The application size is limited to $\rho = 31$ (i.e., $|N|$=495), which is roughly equal to that in Exp-5. We vary $E_{given}(G)$ from $(E_{min}(G) + E_{max}(G))/6$ to $(E_{min}(G) + E_{max}(G))/10$. Table 16 shows the scheduling results. From the table, it can be seen that all these three methods can satisfy energy consumption constraints. Nevertheless, our method achieves the best performance and it significantly outperforms the competitors. In addition, one can see from from Fig. 8(a) that, our method still performs pretty stable and obtains a high reliability, when we vary $E_{given}(G)$. Combining the results in Exp-5, this indicates that our algorithm, ISAECC*, can also work effectively for different parallel applications.

Exp-8. This experiment compares $E(G)$ and $R(G)$ of the *GE application* under *different scales*. The energy consumption constraint is fixed to $(E_{min}(G) + E_{max}(G))/6$, and we vary ρ from 21 (i.e., $|N|$=230) to 106 (i.e., $|N|$=5670), which is roughly equal to that in Exp-6. Table 17 shows the schedule results, and Fig. 8(b) plots the variation tendency of reliability values when varying ρ. Generally, these results are similar to that in Exp-6. That is, our algorithm can satisfy energy consumption constraints for all these cases, and can produce the higher reliability even if the scale is large. On the other hand, the reliability values obtained by the competitors turn smaller drastically, when the scale increases. This further demonstrates the competitiveness of our approach.

6.4 Experimental Results for Maximizing Reliability with Deadline and Energy Consumption Constraints

Exp-9.This experiment is to compare $R(G)$, $SL(G)$, $E(G)$ of the FFT application under different $E_{given}(G)$. The application size is set to $\rho = 32$ (i.e., $|N| = 223$), the deadline constraint $D_{given}(G)$ is set to 1200, $E_{given}(G)$ is set to $E_{heft}(G) \times t$, where t varies from 0.3 to 0.7, and $E_{heft}(G)$ is the energy consumption generated by the HEFT algorithm [38]. Table 18 shows the schedule length, energy consumption and reliability results. Meanwhile, Fig. 9(a) plots the variation tendency of reliability values when changing $E_{given}(G)$.

One can see from Fig. 9(a) and Table 18 that, the reliability values generated by FFSV2 and HEFT are fixed, and FFSV2 achieves the best reliability, which is even better than our proposed method. Nevertheless, FFSV2 cannot satisfy the energy consumption constraint, which can be understood from Table 18. In contrast, our suggested method MRDECC can achieve the higher reliability, while both $SL(G)$ and $E(G)$ can satisfy the constraints, as shown in Table 18 . On the other hand, one can observe that

TABLE 19
Scheduling results of FFT application with different scales

ρ	$D_{given}(G)$	$E_{given}(G)$	HEFT			ISAECC			FFSV2			MRDECC		
			SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)
8	739.5	517.96	435	1035.92	0.9942	493.57	497.62	0.9909	548	589.29	0.9987	546.51	517.88	0.9985
16	912.9	1040.76	537	2081.55	0.9896	608.39	1039.54	0.9837	692	1535.06	0.9969	668.73	1040.75	0.9951
32	1125.4	2416.10	662	4832.19	0.9767	749.82	2413.77	0.9646	1096	3550.20	0.9929	887.91	2415.93	0.9877
64	1523.2	5597.35	896	11194.69	0.9440	1008.66	5597.16	0.8889	1512.99	9558.59	0.9749	1242.19	5597.32	0.9710
128	1844.5	11688.97	1085	23377.94	0.8830	1229.41	11688.85	0.7814	1844.12	20739.06	0.9362	1761.41	11688.90	0.9237

TABLE 20
Scheduling results of FFT application by varying $D_{given}(G)$

$D_{given}(G)$	HEFT			ISAECC			FFSV2			MRDECC		
	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)	SL(G)	E(G)	R(G)
800	795	5175.67	0.9675	795	3104.60	0.9573	800	4443.26	0.9865	799.95	3105.17	0.9804
900	795	5175.67	0.9675	795	3104.60	0.9573	900	4432.42	0.9876	896.64	3105.37	0.9880
1000	795	5175.67	0.9675	795	3104.60	0.9573	1000	4274.50	0.9914	963.74	3100.19	0.9917
1100	795	5175.67	0.9675	795	3104.60	0.9573	1095	4312.91	0.9923	963.74	3100.19	0.9917
1200	795	5175.67	0.9675	795	3104.60	0.9573	1192	4257.67	0.9926	963.74	3100.19	0.9917

both MRDECC and ISAECC can satisfy two constraints, while ISAECC is dominated by the suggested method MRDECC. Particularly, when E_{given} increases, the reliability of MRDECC increases and it gradually approaches that of FFSV2, while it can still satisfy the constraints.

Exp-10. This experiment is to compare $R(G)$, $SL(G)$ and $E(G)$ of FFT application by varying the application size ρ. The energy consumption E_{given} is set to $E_{heft}(G) \times 0.5$, and the deadline constraint $D_{given}(G)$ is set to $SL_{heft}(G) \times 1.7$, where $E_{heft}(G)$ and $SL_{heft}(G)$ are generated by HEFT algorithm. We vary ρ from 8 (i.e., $|N| = 39$) to 128 (i.e., $|N| = 1151$). Table 19 shows the schedule length, energy consumption and reliability results. Meanwhile, Fig. 9(b) plots the variation tendency of reliability values when varying ρ.

One can see from Fig. 9(b) and Table 19 that, the reliability values decrease for all these methods when ρ increases. This is mainly because N in Eq. 21 increases. In addition, FFSV2 still achieves the best reliability (cf., Fig. 9(b)), yet it does not satisfy energy constraint (cf., Table 19). In contrast, our suggested method MRDECC can satisfy both deadline and energy consumption constraints (cf., Table 19). In this regard, it is the same to ISAECC. Yet, MRDECC is significantly better than ISAECC, especially when ρ is large (e.g., when p=128, our suggested method outperforms ISAECC by about 18.2%).

Exp-11. This experiment is to compare $R(G)$, $SL(G)$ and $E(G)$ of FFT application by varying $D_{given}(G)$. The application size ρ is set to 32 (i.e., $|N| = 223$), and the energy consumption constraint $E_{given}(G)$ is fixed to $E_{heft}(G) \times 0.6 = 3105.40$. We vary $D_{given}(G)$ form 800 to 1200. Table 20 shows the scheduling results. Fig. 9(c) plots the variation tendency of reliability values when varying $D_{given}(G)$.

As we expected, all these methods can satisfy the deadline constraint (cf., Table 20). In addition, one can see from Fig. 9(c) and Table 20 that, FFSV2 and our suggested method MRDECC have larger reliability values, compared against other two methods. Particularly, the curve of MRDECC is closely near to that of FFSV2, and even exceeds that of FFSV2 (when $D_{given}(G)$ ranges from 900 to 1000). Note that, when $D_{given}(G) > 1000$, the reliability is no longer increasing. This is mainly because another constraint, i.e., energy consumption constraint, also affects the performance. Additionally, we can see from Table 20 that FFSV2 cannot satisfy energy consumption constraint for all cases (recall that $E_{given}(G) = 3105.40$). On a whole, all these evidences indicate that our suggested method performs well for the MRDECC problem.

7 Conclusion

In this paper, we proposed a new algorithm to minimize the schedule length for energy consumption constrained parallel applications on heterogeneous computing systems. The basic idea of our algorithm is decompose the problem into two-subproblems and then use a weight-based mechanism to preassign the energy consumption for unassigned tasks. Furthermore, we also extended our idea to two other interesting problems, i.e., MRECC and MRDECC problems. Extensive experiments on real applications demonstrated that our algorithms are effective and competitive, compared against state-of-the-art algorithms. In the future, we would like to leverage other factors (e.g., latency) to develop more efficient solutions. Another interesting topic is to adapt our techniques to solve online scheduling problems in data center network.

Acknowledgement

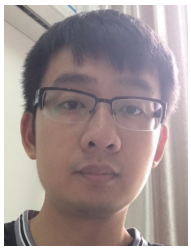
We thank the editors and anonymous reviewers very much for their constructive comments. This work was supported in part by the National Key R&D Program of China (2018YFB0204100, 2018YFB1004400), the National Natural Science Foundation of China (61472124, 61472453, 61602166, 61702320, U1401256, U1501252, U1611264, U1711261, U1711262, U61811264).

References

[1] T. Ye, Z.-J. Wang, Z. Quan, S. Guo, K. Li, and K. Li, "Isaecc: An improved scheduling approach for energy consumption constrained parallel applications on heterogeneous

distributed systems," in *the 24th IEEE Intl. Conf. on Parallel and Distributed Processing*, 2018, pp. 1–8.
[2] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *OSDI*, 1994, pp. 13–23.
[3] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Computers*, vol. 61, no. 12, pp. 1668–1681, 2012.
[4] G. Xie, G. Zeng, R. Li, and K. Li, "Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing," *IEEE Trans. Sust. Comput.*, vol. 2, no. 2, pp. 62–75, 2017.
[5] M. A. Islam, S. Ren, A. H. Mahmud, and G. Quan, "Online energy budgeting for cost minimization in virtualized data center," *IEEE Trans. Serv. Comput.*, vol. 9, no. 3, pp. 421–432, 2016.
[6] J. Zhang, Z. Wang, Z. Quan, J. Yin, Y. Chen, and M. Guo, "Optimizing power consumption of mobile devices for video streaming over 4g LTE networks," *Peer-to-Peer Networking and Applications*, vol. 11, no. 5, pp. 1101–1114, 2018.
[7] R. Ranjan, L. Wang, A. Y. Zomaya, and D. Georgakopoulos, "Recent advances in autonomic provisioning of big data applications on clouds," *IEEE Trans. Cloud Computing*, vol. 3, no. 2, pp. 101–104, 2015.
[8] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 870–882, 2019.
[9] J. Zhang, Z. Wang, S. Guo, D. Yang, G. Fang, C. Peng, and M. Guo, "Power consumption analysis of video streaming in 4g LTE networks," *Wireless Networks*, vol. 24, no. 8, pp. 3083–3098, 2018.
[10] C. Xu, K. Wang, and M. Guo, "Intelligent resource management in blockchain-based cloud datacenters," *IEEE Cloud Computing*, vol. 4, no. 6, pp. 50–59, 2017.
[11] L. Zhao, Y. Ren, and K. Sakurai, "A resource minimizing scheduling algorithm with ensuring the deadline and reliability in heterogeneous systems," in *25th IEEE International Conference on Advanced Information Networking and Applications, AINA 2011, Biopolis, Singapore, March 22-25, 2011*, 2011, pp. 275–282.
[12] G. Xie, Z. Gang, L. Yan, Z. Jia, R. Li, and K. Li, "Fast functional safety verification for distributed automotive applications during early design phase," *IEEE Transactions on Industrial Electronics*, vol. PP, no. 99, pp. 1–1, 2017.
[13] M. R. Stan and K. Skadron, "Guest editors' introduction: Power-aware computing," *IEEE Computer*, vol. 36, no. 12, pp. 35–38, 2003.
[14] K. Gharehbaghi, F. Kocer, and H. Kulah, "Optimization of power conversion efficiency in threshold self-compensated UHF rectifiers with charge conservation principle," *IEEE Trans. Circuits and Systems*, vol. 64-I, no. 9, pp. 2380–2387, 2017.
[15] 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.
[16] F. F. Yao, A. J. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *FOCS*, 1995, pp. 374–382.
[17] 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.
[18] 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.
[19] 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.
[20] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 175–186, 2015.
[21] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, and K. Li, "Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems," *IEEE Trans. Indust. Inform.*, vol. PP, no. 99, pp. 1–1, 2016.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] X. Xiao, G. Xie, R. Li, and K. Li, "Minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems," in *ISPA (best paper award)*, 2016, pp. 1471–1476.
[27] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Information Sciences*, vol. 319, no. C, pp. 113–131, 2015.
[28] M. Lin, Y. Pan, L. T. Yang, M. Guo, and N. Zheng, "Scheduling co-design for reliability and energy in cyber-physical systems," *IEEE Trans. Emerging Topics in Computing*, vol. 1, no. 2, pp. 353–365, 2017.
[29] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Indust. Inform.*, vol. 6, no. 3, pp. 316–328, 2010.
[30] K. Li, "Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1484–1497, 2008.
[31] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2867–2876, 2014.
[32] C. Rusu, R. Melhem, and D. Moss, "Maximizing rewards for real-time applications," *ACM Trans. Embedded Comput. Syst.*, vol. 2, no. 4, pp. 537–559, 2003.
[33] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 3560–3580, 2018.
[34] J. J. Chen and T. W. Kuo, "Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics," in *ICPP*, 2005, pp. 13–20.
[35] 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.
[36] 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.
[37] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Computing*, vol. 38, no. 4-5, pp. 175–193, 2012.
[38] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heteroge-

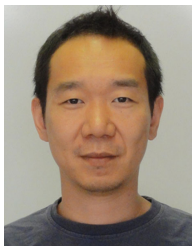
neous computing," *IEEE Trans. Parallel and Distrib. Systems*, vol. 13, no. 3, pp. 260–274, 2002.
[39] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 16–27, 1989.
[40] S. Aminzadeh and A. Ejlali, "A comparative study of system-level energy management methods for fault-tolerant hard real-time systems," *IEEE Trans. Computers*, vol. 60, no. 9, pp. 1288–1299, 2011.
[41] A. Benoit, F. Dufoss, A. Girault, and Y. Robert, "Reliability and performance optimization of pipelined real-time systems ," *J. Parallel Distributed Computing*, vol. 73, no. 6, pp. 851–865, 2013.
[42] G. Xie, H. Peng, Z. Li, J. Song, Y. Xie, R. Li, and K. Li, "Reliability enhancement towards functional safety goal assurance in energy-aware automotive cyber-physical systems," *IEEE Trans. Indust. Inform.*, vol. PP, pp. 1–14, Jul. 2018.
[43] C.-Y. Chen, "Task scheduling for maximizing performance and reliability considering fault recovery in heterogeneous distributed systems," *IEEE Trans. Parallel Distrib. Syst*, vol. 27, no. 2, pp. 521–532, 2016.
[44] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. Chang, M. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Serv. Comput.*, vol. PP, no. 99, pp. 1–1, 2017.
[45] A. Girault, rik Saule, and D. Trystram, "Reliability versus performance for critical applications," *J. Parallel Distrib. Comput.*, vol. 69, no. 3, pp. 326–336, 2009.
[46] X. Tang, K. Li, M. Qiu, and H. M. Sha, "A hierarchical reliability-driven scheduling algorithm in grid systems," *J. Parallel and Distrib. Comput.*, vol. 72, no. 4, pp. 525–535, 2012.
[47] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Trans. Computers*, vol. 58, no. 10, pp. 1382–1397, 2009.
[48] L. Zhang, K. Li, C. Li, and K. Li, "Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems," *Information Sciences*, vol. 379, 2016.
[49] A. Dogan and F. Özgüner, "Matching and scheduling algorithms for minimizing execution timeand failure probability of applications in heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst*, vol. 13, no. 13, pp. 308–323, 2002.
[50] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Indust. Inform.*, vol. 6, no. 3, pp. 316–328, 2010.
[51] G. Xie, J. Jiang, Y. Liu, R. Li, and K. Li, "Minimization energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems," *IEEE Trans. Indust. Inform.*, vol. PP, no. 99, pp. 1–1, 2017.
[52] 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.
[53] A. Benoit, L. C. Canon, E. Jeannot, and Y. Robert, "Reliability of task graph schedules with transient and fail-stop failures: complexity and algorithms," *J. Scheduling*, vol. 15, no. 5, pp. 615–627, 2012.



Zhe Quan received the PhD degree in computer science from the University de Picardie Jules Verne, France. He is currently an associate professor at the College of Computer Science and Technology, Hunan University (HNU), Changsha, China. Before joining HNU, he worked at the National University of Defense Technology, Changsha, China. His main research interests include parallel and high-performance computing, machine learning, etc.

Zhi-Jie Wang received the PhD degree in computer science from the Shanghai Jiao Tong University, Shanghai, China. He is currently a research associate professor at the Sun Yat-Sen University (SYSU), Guangzhou, China. Before joining SYSU, he was a postdoctoral research fellow at the Hong Kong Polytechnic University, Kowloon, Hong Kong. His current research interests include but not limited to algorithm design & analysis, distributed computing, databases, etc. He is a member of CCF, IEEE and ACM.

Ting Ye received the Master degree in computer science from the Hunan University, Changsha, China. She is currently a software engineer at the Citibank, Shanghai, China. Her research interests include parallel and distributed systems, natural language processing, algorithm design and analysis, etc.



Song Guo received the Ph.D. degree in computer science from the University of Ottawa, Canada. He is currently a full professor at The Hong Kong Polytechnic University (PolyU). Prior to joining PolyU, he was a full professor with the University of Aizu, Japan. His research interests are mainly in the areas of cloud and green computing, big data, wireless networks, and cyber-physical systems. He has published over 300 conference and journal papers in these areas and received multiple best paper awards from IEEE/ACM conferences. His research has been sponsored by JSPS, JST, MIC, NSF, NSFC, and industrial companies. Dr. Guo has served as an editor of several journals, including IEEE TPDS, IEEE TCC, IEEE TETC, IEEE TGCN, IEEE Communications Magazine, and Wireless Networks. He has been actively participating in conference organizations serving as general chair and TPC chair. He is a senior member of IEEE, a senior member of ACM, and an IEEE Communications Society Distinguished Lecturer.