Reliability/Performance-Aware Scheduling for Parallel Applications With Energy Constraints on Heterogeneous Computing Systems

Jiwu Peng, Kenli Li, *Senior Member, IEEE*, Jianguo Chen, and Keqin Li, *Fellow, IEEE*

Abstract—Heterogeneous Computing Systems (HCSs) have developed rapidly due to their high performance and low cost, and have been adopted by more and more applications. Energy consumption, reliability, and schedule length are the core issues of HCSs. Due to the negative correlation between frequency and reliability, DVFS-supported HCSs requires high energy consumption and a long schedule length to obtain high reliability, which resulting in performance degradation. In this paper, we focus on the reliability and performance-aware scheduling for energy-constrained parallel applications on HCSs. First, we design an energy pre-allocation mechanism based on Energy Demand Rate (EDR) to pre-allocate energy reasonably. Second, we propose an EDR-aware Maximizing Reliability of Energy-Constrained parallel applications (EMREC) scheduling algorithm. Third, considering that maximize reliability will cause the schedule length to be too long and unacceptable, we further highlight the concept of Reliability Performance Ratio (RPR). Finally, we propose a Maximizing RPR with Energy-Constrained parallel applications (MRPEC) scheduling algorithm, which enables parallel applications have a smaller schedule length while with high reliability. Extensive experimental results in real-world and randomly generated applications show the effectiveness of the proposed algorithms under different conditions.

Index Terms—DVFS, energy consumption constrained, energy demand rate, parallel application scheduling, performance and reliability, reliability performance ratio

1 Introduction

1.1 Motivation

With the promotion of new technologies such as artificial intelligence, big data, and the Internet of Things (IoT), the types of applications are diversified. Meanwhile, the calculation of application requirements is also different, and the trend of diversification of calculation is unstoppable. Heterogeneous Computing Systems (HCSs) can better meet individual computing requirements, play an increasingly important role in carrying applications, and has a broad space for development [1], [2], [3], [4]. Environmental constraints are a prominent contradiction in the current economic and social development. Promoting energy conservation is of great significance to the realization of sustainable economic development. In addition, energy saving can prolong the service life of the device, thereby improving the usability of the device, which is especially important in embedded devices of IoTs [5], [6], [7]. In recent years, parallel application scheduling based on energy consumption constraints has attracted widespread attention [8], [9], [10].

As an effective energy-saving technology, dynamic voltage and frequency scaling (DVFS) has been widely used in academic research and industrial practice [11], [12], [13]. Currently, most mainstream chip manufacturers support DVFS technology, such as Intel SpeedStep, AMD PowerNow, and ARM IEM (Intelligent Energy Manager) and AVS (Adaptive Voltage Scaling)[10], [14]. However, dynamically reducing the chip's voltage may cause the processor's transient failure to rise sharply thereby affecting the reliability of the system [15], [16]. Therefore, it is very important to maximize the reliability of parallel applications of energy constraints on the DVFS-supported HCSs. Although the above issues have been studied in [17], [18], due to the pessimistic energy pre-allocation characteristics, there is still room for improvement. At the same time, performance is also one of the core indicators of HCSs. However, most existing algorithms for maximizing reliability and energy consumption optimization do not consider application performance, and exhibit poor scheduling lengths for large-scale tasks. In this case, the schedule length of the maximum reliability algorithm with energy consumption constraints is too pessimistic. Therefore, it is critical to propose an effective algorithm that balances reliability and performance.

- *Jiwu Peng and Kenli Li are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China. E-mail: {jiwu_peng, lkl}@hnu.edu.cn.*
- *Jianguo Chen is with the Institute for Infocomm Research, Agency for Science Technology and Research, Singapore 117684. E-mail: chen_jianguo@i2r.a-star.edu.sg.*
- *Keqin Li is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 19 June 2021; revised 18 December 2021; accepted 20 January 2022. Date of publication 27 January 2022; date of current version 8 September 2022.
This work was supported in part by the National Key R&D Program of China under Grant 2020YFB2104000 and in part by the National Natural Science Foundation of China under Grants 61860206011, 62172151, and 61876061. (Corresponding author: Kenli Li.)
Recommended for acceptance by G. Min.
Digital Object Identifier no. 10.1109/TSUSC.2022.3146138

1.2 Contributions

In this paper, we first introduce the concept of Energy Demand Rate (EDR) and propose a novel EDR-aware Maximize Reliability for Energy-Constrained parallel applications (EMREC) scheduling algorithm in HCSs. Then, to solve the problem of excessively long schedule length in the process of maximizing application reliability, we propose the concept of Reliability Performance Ratio (RPR) to balance reliability and performance. Further, we propose a Maximize RPR for Energy-Constrained parallel applications (MRPEC) scheduling algorithm in HCSs. The main contributions of this paper can be summarized as follows:

- We introduce the concept of EDR to reasonably transfer the energy consumption constraints of the application to the constraints of each task. Then, we propose EMREC, an EDR-based scheduling algorithm, to maximize parallel applications reliability.
- We introduce the concept of RPR to balance applications reliability and performance. Then, we further propose MRPEC, a RPR-based scheduling algorithm to maximize parallel applications RPR. RPR is not only used as the optimization objective of algorithms, but also as an evaluation index.
- We conduct extensive experiments on real-world and randomly generated parallel applications to evaluate the effectiveness of the proposed algorithms by comparing with the state-of-the-art algorithms. Experimental results proved that the proposed EMREC and MRPEC algorithms have superiority in balancing reliability and schedule length under different energy consumption constraints and application scales.

The rest of this article is organized as follows. Section 2 introduces related work. Section 3 builds related models and introduces the preliminaries. Section 4 proves the feasibility of EDR-based scheduling and introduces the proposed EMREC algorithm. Section 5 describes the scheduling strategy based on RPR and introduces the proposed MRPEC algorithm. Section 6 verifies the performance of the proposed algorithms. Section 7 summarizes this work.

2 Related Work

We mainly review the latest studies on the energy, reliability, schedule length, and their relationship of parallel application scheduling in HCSs.

Energy consumption optimization of DAG-based parallel applications is a hot research topic in recent years. There are a lot of excellent research in energy optimization. In [19], Lee *et al.* studied energy-saving-conscious scheduling that simultaneously minimizes the energy consumption and schedule length of parallel applications. In [20], Salami *et al.* proposed an energy-saving framework for heterogeneous perception fairness, which uses DVFS to satisfy fairness constraints and provides energy-saving scheduling in heterogeneous multicore processors. In [21], Tang *et al.* investigated the energy consumption optimization problem of computing nodes on the fat-tree interconnection network. They proposed an energy-saving scheduling algorithm based on the heuristic list method to achieve low communication and calculation ratio. In [22], Song *et al.* investigated the minimization of the schedule length for energy-constrained applications in HCSs. The core of their work is the energy pre-allocation mechanism. However, the above studies do not consider the reliability of the applications.

Reliability-based scheduling of parallel applications has also been extensively studied. In [23], Tang *et al.* investigated the reliable scheduling of DAG-based parallel applications on large heterogeneous grid systems, and proposed a hierarchy-based reliability-driven scheduling algorithm. In [24], Naithani *et al.* conducted a large number of investigations and showed that applications showed different reliability characteristics on large high-performance cores, and showed huge opportunities on small energy-saving cores. They monitor reliability characteristics and dynamically schedule applications to the different heterogeneous core to maximize system reliability. In [25], Wang *et al.* designed a replication-based scheduling algorithm to maximize system reliability, and incorporated task communication into system reliability. In [17], Zhang *et al.* proposed a competitive-aware reliability method for parallel application in HCSs.

Various work has been conducted to maximize reliability under energy consumption constraints. In [26], Zhang *et al.* focused on the reliability maximization energy conservation problem in HCSs, and proposed the reliability maximization energy conservation (RMEC) algorithm. In [18], Xiao *et al.* studied the reliability maximization problem under energy consumption constraints, and proposed a method of pre-allocating each task according to the minimum energy demand. However, this method is too pessimistic and there is still the possibility of further improvement, and they did not study the problem of reliability performance balance under energy consumption constraints. In [27] Kumar *et al.* investigated the energy consumption optimization of heterogeneous multi-processor environments by restricting the timing and reliability of non-preemptive periodic real-time tasks. However, they did not consider the scheduling problem based on the precedence-constrained tasks, and the optimization goals they consider are not the same as ours. In [28], Zhou *et al.* focused on the joint optimization of soft-error reliability (SER) and lifetime reliability (LTR) of real-time homogeneous MPSoC systems. However, they did not consider heterogeneous systems and the issue of reliability and performance balance under energy constraints.

However, based on our investigation, there is currently no research work on the balance between reliability and schedule length under given energy constraints for DAG-based parallel applications in HCSs.

3 Models and Preliminaries

The main concepts and their definitions in this paper are listed in Table 1.

3.1 Application Model

$\mathcal{G} = (\mathcal{T}, \mathcal{E}, \mathcal{C}, \mathcal{W})$ is used to define the DAG application model in this paper, where $\mathcal{T}$ is the set of computing tasks, $\mathcal{E}$ is the set of communication edges, $\mathcal{C}$ represents the set of communication time, and $\mathcal{W}$ is a computing matrix. $\tau_i \in \mathcal{T}$ represents a computing task, $c_{i,j} \in \mathcal{C}$ represents the communication time between tasks τ_i and τ_j, and $w_{i,k} \in \mathcal{W}$ is the

TABLE 1
Important Notations in this Study

Symbol	Meaning
$\mathcal{P}$	The set of processors in HCS.
$\mathcal{G}$	The DAG-basd application model.
$\varphi_{k,\text{ind}}$	Dynamic frequency independent of frequency in processor μ_k.
$\epsilon_{k,ef}$	Effective switched capacitance in processor μ_k.
τ_{entry}	Entry task of application $\mathcal{G}$.
τ_{exit}	Exit task of application $\mathcal{G}$.
$\text{EST}(\tau_i)$	Earliest start time of task τ_i executed on processor p_k.
$\text{EFT}(\tau_i)$	Earliest finish time of task τ_i executed on processor p_k.
$\text{AST}(\tau_i)$	Actual execute time of task p_i.
$\text{AFT}(\tau_i)$	Actual finish time of task τ_i.
$SL(\mathcal{G})$	Final schedule length of application $\mathcal{G}$.
$E(\tau_i)$	Energy consumption of task τ_i.
$E(\mathcal{G})$	Total energy consumption of application $\mathcal{G}$.
$E_{\text{ae}}(\mathcal{G})$	Allocable energy of application $\mathcal{G}$.
$\text{EDR}(\tau_i)$	Energy demand rate of each task in the application.
$E_{\text{tae}}(\tau_i)$	Task allocation energy of task τ_i.
$E_{\text{pre}}(\tau_i)$	Pre-allocated energy constraints of task τ_i.
$E_{\text{cons}}(\tau_i)$	Energy constraints of task τ_i.
$E_{\text{cons}}(\mathcal{G})$	Given energy constraints of application $\mathcal{G}$.

execution time of task τ_i running on processor p_k with maximum frequency. $\mathcal{P} = \{p_1, p_2, \ldots, p_{|\mathcal{P}|}\}$ represents the set of processors in a HCS. $pred(\tau_i)$ and $succ(\tau_i)$ respectively represent the direct predecessor and successor set of task τ_i. τ_{entry} and τ_{exit} represents the entry task and the exit task of the application. Fig. 1 shows an example of a DAG-based application with 10 tasks , where the integer between nodes represents their communication time and arrows indicate dependencies between tasks [15], [22], [29].

3.2 Energy Model

In this study, we adopt the system-level power model originally proposed in [30], which has been widely used in recent works such as [15], [31]. The system power model[1] of the frequency ϕ is given by

$$\varphi(\phi) = \varphi_{\text{s}} + h(\varphi_{\text{ind}} + \varphi_{\text{d}}) = \varphi_{\text{s}} + h(\varphi_{\text{ind}} + \xi_{\text{ef}}\phi^{\epsilon}), \tag{1}$$

where φ_s is static power, φ_{ind} is frequency-independent power, frequency-dependent dynamic power is expressed as φ_d , h is the system state, ξ_{ef} represents the effective capacitance, and ϵ represents the dynamic power exponent. For a HCS, the parameter sets are defined as follows:

$$\begin{cases} \varphi_{\text{ind}} = \left\{\varphi_{\text{ind},1}, \varphi_{\text{ind},2}, \ldots, \varphi_{\text{ind},|\mathcal{P}|}\right\} \\ \varphi_d = \left\{\varphi_{1,\text{d}}, \varphi_{2,\text{d}}, \ldots, \varphi_{|\mathcal{P}|,\text{d}}\right\} \\ \xi_{\text{ef}} = \left\{\xi_{\text{ef},1}, \xi_{\text{ef},2}, \ldots, \xi_{\text{ef},|\mathcal{P}|}\right\} \\ \epsilon = \left\{\epsilon_1, \epsilon_2, \ldots, \epsilon_{|\mathcal{P}|}\right\} \\ \phi_{\text{low}} = \{\phi_{1,\text{low}}, \phi_{2,\text{low}}, \ldots, \phi_{|\mathcal{P}|,\text{low}}\}. \end{cases} \tag{2}$$

1. The power consumption model is based on [30], and more advanced power consumption models will be studied in the future.

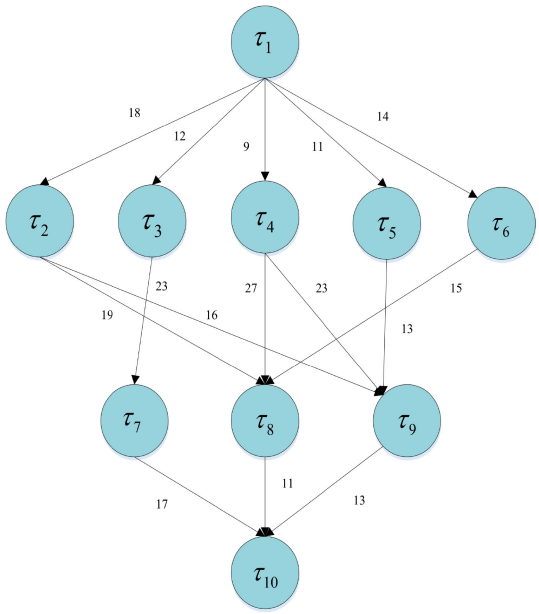


Fig. 1. Example of a DAG-based parallel application.

The energy-efficient frequency [30] is denoted as ϕ_{ee}, which is computed by

$$\phi_{ee} = \sqrt[\epsilon]{\frac{\varphi_{\text{ind}}}{(\epsilon - 1)\xi_{\text{ef}}}}.$$

The actual frequency ϕ is in the interval $[\phi_{\text{low}}, \phi_{\max}]$, where $\phi_{\text{low}} = \max(\phi_{\min}, \phi_{\text{ee}})$. The actual effective frequency set is defined as

$$\begin{gathered} \{\phi_{1,\text{ low}}, \phi_{1,\alpha}, \phi_{1,\beta}, \ldots, \phi_{1,\max}\} \\ \{\phi_{2,\text{ low}}, \phi_{2,\alpha}, \phi_{2,\beta}, \ldots, \phi_{2,\max}\} \\ \cdots, \\ \{\phi_{|\mathcal{P}|,\text{ low}}, \phi_{|\mathcal{P}|,\alpha}, \phi_{|\mathcal{P}|,\beta}, \ldots, \phi_{|\mathcal{P}|,\max}\}. \end{gathered}$$

Then, the energy consumption of task τ_i executed on the processor p_k with frequency $\phi_{k,h}$ is calculated by

$$E(\tau_i, p_k, \phi_{k,h}) = \left(\rho_{\text{ind}} + \xi_{k,\text{ef}} \times \phi_{k,h}^{\epsilon_k}\right) \times w_{i,k} \times \frac{\phi_{k,\max}}{\phi_{k,h}}, \tag{3}$$

where $\xi_{k,\text{ef}}$ represents the effective capacitance and ϵ_k represents the dynamic power exponent of the processor p_k. Therefore, the energy consumption of the application $\mathcal{G}$ is obtained by

$$E(\mathcal{G}) = \sum_{i=1}^{|\mathcal{T}|} E(\tau_i, p_k, \phi_{k,h}). \tag{4}$$

Then, the minimum and maximum energy consumption of task τ_i can be calculated as

$$\begin{cases} E_{\min}(\tau_i) = \min_{p_k \in \mathcal{P}} E(\tau_i, p_k, \phi_{k,\text{low}}), \\ E_{\max}(\tau_i) = \max_{p_k \in \mathcal{P}} E(\tau_i, p_k, \phi_{k,\max}). \end{cases} \tag{5}$$

Similarly, the minimum and maximum energy consumption of application $\mathcal{G}$ can be calculated as

$$\begin{cases} E_{\min}(\mathcal{G}) = \sum_{i=1}^{|\mathcal{T}|} E_{\min}(\tau_i), \\ E_{\max}(\mathcal{G}) = \sum_{i=1}^{|\mathcal{T}|} E_{\max}(\tau_i). \end{cases} \tag{6}$$

We define $E_{\text{cons}}(\mathcal{G})$ as the energy constraint of application $\mathcal{G}$. $E_{\text{cons}}(\mathcal{G})$ is set between $E_{\min}(\mathcal{G})$ and $E_{\max}(\mathcal{G})$, which is lower than $E_{\min}(\mathcal{G})$ the application cannot be scheduled, and greater than $E_{\max}(\mathcal{G})$ energy cannot be saved. The relationship between $E_{\text{cons}}(\mathcal{G})$, $E_{\min}(\mathcal{G})$, and $E_{\max}(\mathcal{G})$ is $E_{\min}(\mathcal{G}) \leq E_{\text{cons}}(\mathcal{G}) \leq E_{\max}(\mathcal{G})$.

3.3 Reliability Model

In actual HCS, transition errors in the task execution stage cannot be predicted and avoided, and usually follow a probability distribution [32]. The commonly used model of processor transient failure is the Poisson distribution with the parameter λ [15]. In a non-DVFS system, let λ_k be the failure rate of each time unit of processor p_k, the reliability of task τ_i is calculated by

$$R(\tau_i, p_k) = e^{-\lambda_k w_{i,k}}. \tag{7}$$

For a DVFS-capable system, the failure rate varies with frequency due to the effect of DVFS on transient failures [15]. Let $\lambda_{k,\max}$ be the failure rate of each per time unit in maximum frequency of the processor p_k. Then, the failure rate $\lambda_{k,h}$ of each time unit at frequency $\phi_{k,h}$ is

$$\lambda_{k,h} = \lambda_{k,\max} 10^{\frac{\delta\left(\phi_{k,\max} - \phi_{k,h}\right)}{\phi_{k,\max} - \phi_{k,\min}}}, \tag{8}$$

where δ is the sensitivity of failure rates to voltage scaling. From Eqs. (7) and (8), the reliability of task τ_i executed on the processor p_k at frequency $\phi_{k,h}$ can be calculated as

$$R\left(\tau_i, p_k, \phi_{k,h}\right) = e^{-\lambda_{k,h} \times \frac{w_{i,k} \times \phi_{k,\max}}{\phi_{k,h}}}. \tag{9}$$

Correspondingly, for the application $\mathcal{G}$, its reliability denoted by $R(\mathcal{G})$, which is computed as

$$R(\mathcal{G}) = \prod_{\tau_i \in \mathcal{T}} R(\tau_i) = \prod_{\tau_i \in \mathcal{T}} R\left(\tau_i, p_k, \phi_{k,h}\right). \tag{10}$$

3.4 Preliminaries

In a DVFS-capable HCS, a DAG-based parallel application has its own unique execution model. We introduce the preliminaries of the application execution model here.

1) Earliest Start Time (EST): The EST of task τ_i executed on processor p_k with frequency $\phi_{k,h}$ is denoted as $EST(\tau_i, p_k, \phi_{k,h})$

$$\begin{cases} \text{EST}(\tau_{\text{entry}}, p_k, \phi_{k,h}) = 0 \\ \text{EST}(\tau_i, p_k, \phi_{k,h}) = \max_{\tau_j \in pred(\tau_i)} \left\{ avail[k], \max(\text{AFT}(\tau_j) + c'_{j,i}) \right\}, \end{cases} \tag{11}$$

where $\text{avail}[k]$ is the available timeline of the processor u_k.

2) Earliest Finish Time (EFT): The EFT of task τ_i executed on processor p_k with frequency $\phi_{k,h}$ is denoted as $\text{EFT}(\tau_i, p_k, \phi_{k,h})$

$$\text{EFT}\left(\tau_i, p_k, \phi_{k,h}\right) = \text{EST}\left(\tau_i, p_k, \phi_{k,h}\right) + w_{i,k} \times \frac{\phi_{k,\max}}{\phi_{k,h}}. \tag{12}$$

3) Task Priority Scoring (TPS): TPS is an important issue of DAG list scheduling in HCSs. Considering that the focus of this work is energy and reliability efficiency scheduling, we introduce the task priority scoring method in [29] in our work. The task priority scoring method is widely used in energy-saving scheduling and reliability-aware scheduling [15], [18], [22]. TPS is obtained by:

$$\underset{u}{\text{rank}}(\tau_i) = \overline{w_i} + \max_{\tau_j \in \text{succ}(\tau_i)} \left\{ c_{i,j} + \underset{u}{\text{rank}}(\tau_i) \right\}, \tag{13}$$

where $\overline{w_i} = \left(\sum_{k=1}^{|\mathcal{P}|} w_{i,k} \right) / |\mathcal{P}|$.

4) Schedule length (SL): The schedule length $\text{SL}(\mathcal{G})$ of an application is defined as the execution time of the application from the entry task to the exit task, as calculated as

$$SL(\mathcal{G}) = \max_{\tau_i \in \text{exit task}} \text{AFT}(\tau_i). \tag{14}$$

4 Maximizing Reliability With Energy Constraints

In this section, we first describe the problem of maximizing reliability with energy constraints. Then, we introduce the energy consumption satisfaction mechanism based on the minimum energy requirement and the EDR. Finally, we propose a maximum reliability energy constraints algorithm based on the EDR.

4.1 Problem Description

The target platform of this research is a heterogeneous multi-processor system, and each processor supports different frequency levels. Correspondingly, the problem to be solved in this section is to allocate available processors p_k with appropriate frequency $\phi_{k,h}$ for each task τ_i of parallel applications, while maximizing reliability and ensuring energy consumption does not exceed the given energy limit. The formal description is given as follows:

$$\begin{aligned} &\text{Maximize} : R(\mathcal{G}) = \sum_{i=0}^{|\mathcal{T}|} R(\tau_i) \\ &\text{Subject to} : E(\mathcal{G}) = \sum_{i=1}^{|\mathcal{T}|} E\left(\tau_i, p_k, \phi_{k,h}\right) \leq E_{\text{cons}}(\mathcal{G}) \\ &E_{\min}(\mathcal{G}) \leq E_{\text{cons}}(\mathcal{G}) \leq E_{\max}(\mathcal{G}). \end{aligned} \tag{15}$$

4.2 Satisfying Energy Constraints

For the current allocated task $\tau_{s(j)}$, $\{\tau_{s(1)}, \tau_{s(2)}, \ldots, \tau_{s(j-1)}\}$ represents the allocated task set, meanwhile, the unallocated task set is $\{\tau_{s(j+1)}, \tau_{s(j+2)}, \ldots, \tau_{s(|N|)}\}$. In [18], the minimum energy consumption is pre-allocated to $\{\tau_{s(j+1)}, \tau_{s(j+2)}, \ldots, \tau_{s(|N|)}\}$ to maximize application reliability. However, due to its pessimistic allocation method, the allocatable energy is allocated by high-priority tasks and the low-priority tasks will have fewer and fewer feasible processor and frequency combinations, resulting in reliability and schedule length not optimistic. Therefore, we propose a more effective pre-allocation mechanism based on the proposed energy demand rate. For convenience of description, the concept of energy demand rate and allocatable energy are defined as follows.

Definition 1. *The Energy Demand Rate (EDR) is defined as the minimum energy requirement of each task divided by the minimum energy requirement of the application, which is calculated by*

$$\mathrm{EDR}(\tau_i) = \frac{E_{min}(\tau_i)}{E_{min}(\mathcal{G})}. \tag{16}$$

Definition 2. *The Allocatable Energy (AE) is defined as the given energy constraint minus minimum energy consumption requirement of the application, which is calculated by*

$$\Delta E_{\mathrm{ae}}(\mathcal{G}) = E_{\mathrm{cons}}(\mathcal{G}) - E_{\mathrm{min}}(\mathcal{G}). \tag{17}$$

In order to assign the allocatable energy more reasonably, we propose a distributable energy consumption allocation mechanism for each task based on the EDR:

$$E_{\mathrm{tae}}(\tau_i) = E_{\mathrm{min}}(\tau_i) + \Delta E_{\mathrm{ae}}(\mathcal{G}) \times \mathrm{EDR}(\tau_i). \tag{18}$$

Since the energy required for the task does not need to be greater than its maximum energy requirement, we have

$$E_{\mathrm{pre}}(\tau_i) = \min\{E_{\mathrm{tae}}(\tau_i), E_{\mathrm{max}}(\tau_i)\}. \tag{19}$$

Thus, the task $n_{s(j)}$, the energy consumption of the application can be expressed as:

$$\begin{aligned} E_{s(j)}(\mathcal{G}) = &\sum_{x=1}^{j-1} E\Big(\tau_{s(x)}, p_{k(s(x))}, \phi_{k(s(x)),h(s(x))}\Big) \\ &+ E\big(\tau_{s(j)}, p_k, \phi_{k,h}\big) + \sum_{x=1}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{s(x)}\big). \end{aligned} \tag{20}$$

$\sum_{x=1}^{j-1} E(\tau_{s(x)}, p_{k(s(x))}, \phi_{k(s(x)),h(s(x))})$ represents the energy consumed by the assigned tasks. $E(\tau_{s(j)}, p_k, \phi_{k,h})$ represents the energy consumed by the current task. The last part $\sum_{x=1}^{|N|} E_{\mathrm{pre}}(\tau_{s(x)})$ is the pre-allocated energy of unallocated tasks. From the definition of the problem, $E_{s(j)}(\mathcal{G}) \leq E_{\mathrm{cons}}(\mathcal{G})$, so we have

$$\begin{aligned} E_{s(j)}(\mathcal{G}) = &\sum_{x=1}^{j-1} E\Big(\tau_{s(x)}, p_{k(s(x))}, \phi_{k(s(x)),p(s(x))}\Big) \\ &+ E\big(\tau_{s(j)}, p_k, \phi_{k,h}\big) + \sum_{x=1}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{s(x)}\big) \\ &\leq E_{\mathrm{cons}}(\mathcal{G}). \end{aligned} \tag{21}$$

For task $\tau_{s(j)}$, according to Eq. (21), we have

$$\begin{aligned} E\big(\tau_{\mathrm{s}(j)}, p_k, \phi_{k,h}\big) \leq\ & E_{\mathrm{cons}}(\mathcal{G}) \\ &- \sum_{x=1}^{j-1} E\Big(\tau_{\mathrm{s}(x)}, p_{k(s(x))}, \phi_{k(s(x)),h(\mathrm{s}(x))}\Big) \\ &- \sum_{x=j+1}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{\mathrm{s}(x)}\big). \end{aligned} \tag{22}$$

As each task satisfies the given energy consumption constraint, that is, $E\big(\tau_{s(j)}, p_k, \phi_{k,h}\big) \leqslant E_{\mathrm{cons}}\big(\tau_{s(j)}\big)$, then $E(\mathcal{G}) \leq E_{\mathrm{cons}}(\mathcal{G})$. Thus, the energy consumption constraints of parallel applications $\mathcal{G}$ are decomposed into each task. Theorem 1 illustrate the feasibility of the EDR-based mechanism.

Theorem 1. *Given energy constraints of the application $E_{\mathrm{min}}(\mathcal{G}) \leq E_{\mathrm{cons}}(\mathcal{G}) \leq E_{\mathrm{max}}(\mathcal{G})$, using pre-allocation mechanism based on the EDR, each task $\tau_{s(j)}$ can always find a processor frequency combination that satisfying:*

$$\begin{aligned} E_{\mathrm{s}(j)}(\mathcal{G}) = &\sum_{x=1}^{j-1} E\Big(\tau_{\mathrm{s}(x)}, p_{k(\mathrm{s}(x))}, \phi_{k(\mathrm{s}(x)),h(\mathrm{s}(x))}\Big) \\ &+ E\big(\tau_{\mathrm{s}(j)}, p_k, \phi_{k,h}\big) + \sum_{x=j+1}^{|\mathcal{T}|} E_{\mathrm{pre}}(\tau_{\mathrm{s}(x)}) \leq E_{\mathrm{cons}}(\mathcal{G}). \end{aligned}$$

Proof. We use mathematical induction to prove Theorem 1. First, for task $\tau_{s(1)}$, all other tasks are not assigned, and $\mathcal{G}$ should satisfy its energy constraints:

$$E_{s(1)}(\mathcal{G}) = E\big(\tau_{s(1)}, p_k, \phi_{k,h}\big) + \sum_{x=2}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{s(x)}\big) \leq E_{\mathrm{cons}}(\mathcal{G}). \tag{23}$$

According to Eq. (19), there is $E_{\mathrm{pre}}(\tau_i) \leq E_{\mathrm{tae}}(\tau_i)$,

$$\begin{aligned} &E\big(\tau_{s(1)}, p_k, \phi_{k,h}\big) + \sum_{x=2}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{s(x)}\big) \\ &\qquad \leq E\big(\tau_{s(1)}, p_k, \phi_{k,h}\big) + \sum_{x=2}^{|\mathcal{T}|} E_{\mathrm{tae}}\big(\tau_{s(x)}\big). \end{aligned} \tag{24}$$

Correspondingly, by Eqs. (18), (19), and (20), we have,

$$\begin{aligned} &E\big(\tau_{\mathrm{s}(1)}, p_k, \phi_{k,h}\big) + \sum_{x=2}^{|\mathcal{T}|} E_{\mathrm{pre}}(\tau_{\mathrm{s}(x)}) \\ &= E\big(\tau_{\mathrm{s}(1)}, p_k, \phi_{k,h}\big) + \sum_{x=1}^{|\mathcal{T}|} E_{\mathrm{pre}}(\tau_{\mathrm{s}(x)}) - E_{\mathrm{pre}}(\tau_{\mathrm{s}(1)}) \\ &\leq E\big(\tau_{\mathrm{s}(1)}, p_k, \phi_{k,h}\big) + E_{\mathrm{cons}}(\mathcal{G}) - E_{\mathrm{tae}}(\tau_{\mathrm{s}(1)}). \end{aligned} \tag{25}$$

Evidently, according to Eq. (25) $E_{\mathrm{tae}}(\tau_{\mathrm{s}(1)})$ is greater than or equal to $E_{\mathrm{min}}(\tau_{s(1)})$. Thus, the available processor and frequency combinations can be found by $\tau_{s(1)}$ to satisfy:

$$\begin{aligned} E_{\mathrm{s}(1)}(\mathcal{G}) &= E\big(\tau_{\mathrm{s}(1)}, p_k, \phi_{k,h}\big) + \sum_{x=2}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{s(x)}\big) \\ &\leq E_{\mathrm{cons}}(\mathcal{G}). \end{aligned} \tag{26}$$

Second, assuming that the jth task $\tau_{s(j)}$ can find a processor frequency combination that satisfy the energy constraints, then have

$$\begin{aligned} E_{\mathrm{S}(j)}(\mathcal{G}) = &\sum_{x=1}^{j-1} E\Big(\tau_{s(x)}, p_{k(\mathrm{s}(x))}, \phi_{k(\mathrm{s}(x)),h(\mathrm{s}(x))}\Big) \\ &+ E\big(\tau_{\mathrm{s}(j)}, p_k, \phi_{k,h}\big) \\ &+ \sum_{x=j+1}^{|\mathcal{T}|} E_{\mathrm{pre}}(\tau_{\mathrm{s}(x)}) \leq E_{\mathrm{cons}}(\mathcal{G}). \end{aligned} \tag{27}$$

Therefore, the $(j+1)$th task $\tau_{s(j+1)}$ energy consumption of the parallel application $\mathcal{G}$ is

$$\begin{aligned} E_{\mathrm{s}(j+1)}(\mathcal{G}) = &\sum_{x=1}^{j} E\Big(\tau_{\mathrm{s}(x)}, p_{k(\mathrm{S}(x))}, \phi_{k(\mathrm{s}(x)),h(\mathrm{s}(x))}\Big) \\ &+ E\big(\tau_{\mathrm{s}(j+1)}, p_k, \phi_{k,h}\big) + \sum_{x=j+2}^{|\mathcal{T}|} E_{\mathrm{pre}}(\tau_{\mathrm{s}(x)}). \end{aligned} \tag{28}$$

Substitute Eq. (27) into Eq. (28), we have

$$\begin{aligned} E_{\mathrm{S}(j+1)}(\mathcal{G}) = & \sum_{x=1}^{j-1} E\Big(\tau_{\mathrm{S}(x)}, p_{k(\mathrm{s}(x))}, \phi_{k(\mathrm{s}(x)),h(\mathrm{s}(x))}\Big) \\ & + E\Big(\tau_{\mathrm{S}(j)}, p_{k(\mathrm{s}(j))}, \phi_{k(\mathrm{S}(j)),h(\mathrm{S}(j))}\Big) \\ & + E\big(\tau_{\mathrm{S}(j+1)}, p_k, \phi_{k,h}\big) + \sum_{x=j+1}^{|\mathcal{T}|} E_{\mathrm{pre}}(\tau_{\mathrm{S}(x)}) \\ & - E_{\mathrm{pre}}(\tau_{\mathrm{S}(j+1)}) \le E_{\mathrm{cons}}(\mathcal{G}) \\ & + E\big(\tau_{\mathrm{S}(j+1)}, p_k, \phi_{k,h}\big) - E_{\mathrm{pre}}(\tau_{\mathrm{S}(j+1)}). \end{aligned} \tag{29}$$

Since $E_{\mathrm{pre}}(\tau_{(s)(j+1)}) \ge E_{\min}(\tau_{s(j+1)})$, similar to $j = 1$, there is $E_{s(j+1)} \le E_{\mathrm{cons}}(\mathcal{G})$. Therefore, $\tau_{s(j+1)}$ can also have a suitable processor frequency combination that satisfies the energy constraints. Thus, the correctness of Theorem 1 is proved. □

4.3 EDR-Based Reliability Maximization

According to Theorem 1, when the task $\tau_s(j)$ is allocated, the energy constraint of $\tau_s(j)$ can be expressed as

$$\begin{aligned} E_{\mathrm{cons}}\big(\tau_{s(j)}\big) = & E_{\mathrm{cons}}(\mathcal{G}) \\ & - \sum_{x=1}^{j-1} E\Big(\tau_{s(x)}, p_{k(s(x))}, \phi_{k(s(x)),h(s(x))}\Big) \\ & - \sum_{x=j+1}^{|\mathcal{T}|} E_{\mathrm{pre}}\big(\tau_{s(x)}\big). \end{aligned} \tag{30}$$

The energy required by the task does not need to be greater than the maximum energy consumption of the task, so there is

$$E_{\mathrm{cons}}\big(\tau_{s(j)}\big) = \min\big\{E_{\mathrm{cons}}\big(\tau_{(j)}\big), E_{\max}\big(\tau_{(j)}\big)\big\}. \tag{31}$$

Therefore, after determining the given energy for each task, an algorithm for maximizing reliability under energy constraints based on the EDR is given in Algorithm 1. In EMREC, for each task, under a given energy constraint, it traverses the maximum reliability search frequency of each processor, repeats the above steps, and finds the maximum reliability value of the application under the energy constraint. We describe the detailed steps of each stage as follows:

(1) Prioritize tasks. In Line 1, EMREC sorts each task in the application into S_{dsort} according to $rank_u(\tau_i)$.
(2) Obtain minimum and maximum energy consumption. In Lines 2-5, EMREC uses Eqs. (5) and (6) to calculate the minimum and maximum energy consumption for each task and the application, respectively.
(3) Acquire energy demand rate and pre-allocated energy. In lines 6-9, EMREC calculates EDR and pre-allocates energy constraints for each task.
(4) Determine the given energy constraint for each task. In Lines 11-12, EMREC calculates application allocated energy and application unallocated energy of scheduling task sequence $\tau_{s(j)}$ and calculate the energy constraints for each task.
(5) Satisfy energy constraints. In Lines 13-30, EMREC traverses each processor frequency combination, and finds the combination with the greatest reliability value that meets the task energy constraint at the same time. Lines 16-18 skip frequencies that are greater than the energy constraint.
(6) Maximize application reliability. In Lines 22-28, EMREC finds the processor frequency combination with the maximum reliability and updates the relevant values.
(7) Calculate $E(\mathcal{G}), \mathrm{SL}(\mathcal{G}), R(\mathcal{G})$. In Lines 32-34, EMREC calculate the energy consumption of the application $E(\mathcal{G})$, the actual reliability $R(\mathcal{G})$, and the schedule length $\mathrm{SL}(\mathcal{G})$.

The time complexity of EMREC is mainly in lines 10-31. The time complexity of traversing $|\mathcal{T}|$ tasks is $O(|\mathcal{T}|)$. For each task, the complexity of using EFT to select the processor and frequency is $O(|\mathcal{T}| \times |\mathcal{P}| \times |\Phi|)$, where $|\Phi|$ is the discrete frequency number from $[\phi_{k,\mathrm{low}}, \phi_{k,\max}]$. Therefore, the total time complexity of EMREC is $O(|\mathcal{T}|^2 \times |\mathcal{P}| \times |\Phi|)$.

Algorithm 1. The EMREC Algorithm

Input: $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{C}, \mathcal{W}), \mathcal{P}, E_{\mathrm{cons}}(\mathcal{G})$.
Output: $E(\mathcal{G})$, $\mathrm{SL}(\mathcal{G})$.
1: Sort tasks in application by descending order of $rank_u(\tau_i)$ as S_{dsort};
2: **for** $(\forall i, \tau_i \in \mathcal{T})$ **do**
3: Calculate $E_{\min}(\tau_i)$, $E_{\max}(\tau_i)$ using Eq. (5);
4: **end for**
5: Calculate $E_{\min}(\mathcal{G})$, $E_{\max}(\mathcal{G})$ using Eq. (6);
6: **for** $(\forall i, \tau_i \in N)$ **do**
7: Calculate $\mathrm{EDR}(\tau_i)$ using Eq. (16);
8: Calculate $E_{\mathrm{pre}}(\tau_i)$ using Eqs. (17) and (18);
9: **end for**
10: **while** tasks in *deslist* **do**
11: $\tau_i \leftarrow deslist$.out();
12: Calculate $E_{\mathrm{cons}}(\tau_i)$ using Eqs.(25) and (26);
13: **for** $(\forall k, p_k \in (\mathcal{P}))$ **do**
14: **for** $(\phi_{k,h} \in [\phi_{k,\mathrm{low}}, \phi_{k,\max}]$ **do**
15: Calculate $E(\tau_i, p_k, \phi_{k,h})$ using Eq. (3);
16: **if** $E(\tau_i, u_k, \phi_{k,h}) > E_{\mathrm{cons}}(\tau_i)$ **then**
17: Continue;
18: **end if**
19: Calculate $\mathrm{EST}(\tau_i, p_k, \phi_{k,h})$ using Eq. (12);
20: Calculate $\mathrm{EFT}(\tau_i, p_k, \phi_{k,h})$ using Eq. (13);
21: Calculate $R(\tau_i, p_k, \phi_{k,h})$ using Eq. (9);
22: **if** $(R(\tau_i, u_k, \phi_{k,h}) < R(\tau_i))$ **then**
23: $E(\tau_i) \leftarrow E(\tau_i, u_k, \phi_{k,h})$;
24: $\mathrm{AST}(\tau_i) \leftarrow \mathrm{EST}(\tau_i, p_k, \phi_{k,h})$;
25: $\mathrm{AFT}(\tau_i) \leftarrow \mathrm{EFT}(\tau_i, p_k, \phi_{k,h})$;
26: $R(\tau_i) \leftarrow R(\tau_i, p_k, \phi_{k,h})$;
27: break; // Skip lower frequencies;
28: **end if**
29: **end for**
30: **end for**
31: **end while**
32: Calculate the $E(\mathcal{G})$ using Eq. (4);
33: Calculate the $\mathrm{R}(\mathcal{G})$ using Eq. (10);
34: Calculate the $\mathrm{SL}(\mathcal{G})$ using Eq. (14);
35: **return** $E(\mathcal{G})$, $\mathrm{SL}(\mathcal{G})$, $\mathrm{R}(\mathcal{G})$

5 Maximizeing RPR With Energy Constraints

After analyzing the energy-constrained maximum reliability algorithm, we found that as the number of tasks increases, the schedule length of the application program will increase sharply, which is unacceptable, resulting in extremely poor user QoS. Therefore, how to strike a balance between application reliability and performance is an problem to be solved. In this section, we will address this issue.

Algorithm 2. The MRPEC Algorithm

Input: $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{C}, \mathcal{W}), \mathcal{P}, E_{\text{cons}}(\mathcal{G})$.
Output: $E(\mathcal{G})$, $\text{SL}(\mathcal{G})$, $R(\mathcal{G})$, $\text{RPR}(\mathcal{G})$.

```
Sort tasks in application by descending order of rank_u(τ_i) as S_dsort;
for (∀i, τ_i ∈ N) do
    Calculate E_min(τ_i), E_max(τ_i) using Eq. (5);
    Calculate w̄(τ_i) using Eq. (5);
end for
Calculate E_min(G), E_max(G) using Eq.(6);
Calculate W̄(G) using Eq. (14);
for (∀i, τ_i ∈ T) do
    Calculate EDR(τ_i) using Eq. (16);
    Calculate E_pre(τ_i) using Eqs. (17) and (18);
end for
while tasks in deslist do
    τ_i ← deslist.out();
    Calculate E_cons(τ_i) using Eqs.(25) and (26);
    for (∀k, p_k ∈ P) do
        for (φ_{k,h} ∈ [φ_{k,low}, φ_{k,max}] do
            Calculate E(τ_i, p_k, φ_{k,h}) using Eq. (4);
            if E(τ_i, p_k, φ_{k,h}) > E_cons(τ_i) then
                Continue;
            end if
            Calculate R(τ_i, p_k, φ_{k,h}) using Eq. (10);
            Calculate EST(τ_i, p_k, φ_{k,h}) using Eq. (12);
            Calculate EFT(τ_i, p_k, φ_{k,h}) using Eq. (13);
            Calculate RPR(τ_i, p_k, φ_{k,h}) using Eq. (30);
            if (RPR(τ_i, p_k, φ_{k,h}) < RPR(τ_i)) then
                E(τ_i) ← E(τ_i, p_k, φ_{k,h});
                AST(τ_i) ← EST(τ_i, p_k, φ_{k,h});
                AFT(τ_i) ← EFT(τ_i, p_k, φ_{k,h});
                R(τ_i) ← R(τ_i, p_k, φ_{k,h});
                RPR(τ_i) ← RPR(τ_i, p_k, φ_{k,h});
                break; // Skip lower frequencies;
            end if
        end for
    end for
end while
Calculate the E(G) using Eq. (4);
Calculate the SL(G) using Eq. (14);
Calculate the R(G) using Eq. (10);
Calculate the RPR(G) using Eq. (32);
return E(G), SL(G), R(G), RPR(G)
```

5.1 Problem Description

As in Section 4.1, we consider a heterogeneous multi-processor platform that supports DVFS. In this section, the problem to be solved is to allocate an available processor and appropriate frequency for each task in DAG-based parallel applications, while maximizing the application reliability performance rate, and ensuring that the energy consumption does not exceed a given energy constraints.

We formalize this problem as follows:

$$\begin{aligned} &\text{Maximize}: \text{RPR}(\mathcal{G}) = \sum_{i=0}^{|\mathcal{T}|} \text{RPR}(\tau_i) \\ &\text{Subject to}: E(\mathcal{G}) = \sum_{i=1}^{|\mathcal{T}|} E\Big(\tau_i, p_k, \phi_{\{k,h\}}\Big) \le E_{\text{cons}}(\mathcal{G}) \\ &\qquad E_{\min}(\mathcal{G}) \le E_{\text{cons}}(\mathcal{G}) \le E_{\max}(\mathcal{G}). \end{aligned} \tag{32}$$

5.2 Reliability/Performance-Aware Scheduling Strategy

As seen in the experimental part, the algorithm based on maximum reliability increases the schedule length sharply when the task volume increases, reaching a very high value, which greatly extends the response time. Meanwhile, when the performance-based algorithm increases with the amount of tasks, the reliability value decreases sharply. For this reason, maintaining high reliability of tasks and making parallel applications run at a reasonable schedule length is the focus of our research. For the first time, we define the concepts of task reliability performance ratio and application reliability performance ratio to address this problem. For the convenience of description, we make the following definitions:

Definition 3. *The Task Execution Time (TET) of the task τ_i is denoted as $\text{TET}(\tau_i)$, which is calculated by*

$$\text{TET}(\tau_i) = \text{AFT}(\tau_i) - \text{AST}(\tau_i). \tag{33}$$

Definition 4. *The Task Execution Time (TET) of the application $\mathcal{G}$ is denoted as $\text{TET}(\mathcal{G})$, which is calculated by*

$$\text{TET}(\mathcal{G}) = \sum_{i=0}^{|\mathcal{T}|} (\text{AFT}(\tau_i) - \text{AST}(\tau_i)). \tag{34}$$

Note that in order to obtain the reliability performance ratio, in this study $\text{TET}(\mathcal{G})$ refers to the sum of the execution time of each task on the processors, that is $\sum_{i=0}^{|\mathcal{T}|} \text{TET}(\tau_i)/\text{TET}(\mathcal{G}) = 1$.

Definition 5. *The Reliability Performance Ratio (PRP) of the task τ_i is denoted as $\text{RPR}(\tau_i)$, which is calculated by*

$$\text{RPR}(\tau_i) = \frac{R(\tau_i)}{1} \Big/ \frac{\text{TET}(\tau_i)}{\overline{w}(\tau_i)} = \frac{R(\tau_i)}{1} \times \frac{\overline{w}(\tau_i)}{\text{TET}(\tau_i)}. \tag{35}$$

Definition 6. *The Reliability Performance Ratio (PRP) of the application $\mathcal{G}$ is denoted as $\text{RPR}(\mathcal{G})$, which is calculated by*

$$\text{RPR}(\mathcal{G}) = \frac{R(\mathcal{G})}{1} \Big/ \frac{\text{TET}(\mathcal{G})}{\overline{\mathcal{W}}(\mathcal{G})} = \frac{R(\mathcal{G})}{1} \times \frac{\overline{\mathcal{W}}(\mathcal{G})}{\text{TET}(\mathcal{G})}. \tag{36}$$

The first part of Eq. (35) (i.e., $\frac{R(\tau_i)}{1}$) focuses on reliability, where 1 means completely reliable. When the actual reliability value $R(\tau_i)$ is larger, the overall $\text{RPR}(\tau_i)$ is also larger. The second part of Eq. (35) (i.e., $\frac{\overline{w}(\tau_i)}{\text{TET}(\tau_i)}$) focuses on performance, when the task's $\text{TET}(\tau_i)$ is smaller, the overall $\text{RPR}(\tau_i)$ is also larger. Namely, $\text{RPR}(\tau_i)$ is a trade-off between actual reliability and execution time. By using the strategy of finding maximum task performance reliability

TABLE 2
Example of Task Parameters

τ_i	p_1	p_2	p_3	$rank_u(\tau_i)$
τ_1	14	16	9	108.000
τ_2	13	19	18	77.000
τ_3	11	13	19	80.000
τ_4	13	8	17	80.000
τ_5	12	13	10	69.000
τ_6	13	16	9	63.333
τ_7	7	15	11	42.667
τ_8	5	11	14	35.667
τ_9	18	12	20	44.333
τ_{10}	21	7	16	14.667

ratio in the task scheduling process, the mutual exclusion contradiction between reliability and performance can be alleviated. The task and application can be maintained at a higher ISO26262[33] exposure level. We carried out the algorithm design of this strategy in the next section.

5.3 The Proposed MRPEC Algorithm

The MRPEC algorithm uses the energy demand rate mechanism, which is the same as EMREC in terms of energy for each task. However, we change the optimization goal to maximize the reliability performance rate of each task, thereby balancing application reliability and performance. Namely, it has a smaller schedule length while maintaining high reliability. We describe the detailed process of difference from the EMREC algorithm as follows:

(1) Get the average expected execution time. MRPEC calculates the average expected execution time of task τ_i in Line 4, and calculates the serial average expected execution time of the application in Line 7.

(3) Obtain the RPR value of each task. In Line 25, MRPEC calculates the reliability performance ratio of task τ_i under the combination of processor u_k and frequency $\phi_{k,h}$.

(2) Maximize the PRP. In Lines 26-34, MRPEC traversing all processors and frequencies, first skip frequencies greater than the given energy constraint of the task. When the energy constraint is reached, each task will select the processor with the highest reliability performance ratio and update the relevant value.

(3) Calculate $E(\mathcal{G})$, $\mathrm{S}L(\mathcal{G})$, $R(\mathcal{G})$, $\mathrm{R}PR(\mathcal{G})$. In Lines 38-41, MRPEC calculate the energy consumption $E(\mathcal{G})$, the schedule length $\mathrm{S}L(\mathcal{G})$, the actual reliability $R(\mathcal{G})$, and the reliability performance ratio $\mathrm{R}PR(\mathcal{G})$ of the application.

TABLE 3
Example of Processor Parameters

p_k	$\varphi_{k,\mathrm{ind}}$	$\epsilon_{k,\mathrm{ef}}$	ϵ_k	$\phi_{k,\mathrm{low}}$	$\phi_{k,\mathrm{max}}$	$\lambda_{k,\mathrm{max}}$
p_1	0.03	0.8	2.9	0.26	1.0	0.00015
p_2	0.04	0.8	2.5	0.26	1.0	0.00020
p_3	0.07	1.0	2.5	0.29	1.0	0.00025

The time complexity of MRPEC is mainly in lines 12-35. The time complexity of traversing $|\mathcal{T}|$ tasks is $O(|\mathcal{T}|)$. For each task, the complexity of using EFT to select the processor and frequency is $O(|\mathcal{T}| \times |\mathcal{P}| \times |\Phi|)$, where $|\Phi|$ is the discrete frequency number from $[\phi_{k,\mathrm{low}}, \phi_{k,\mathrm{max}}]$. Therefore, the total time complexity of MRPEC is $O(|\mathcal{T}|^2 \times |\mathcal{P}| \times |\Phi|)$, which is the same as HEFT.

5.4 Example of the Proposed Algorithms

We use the standard example in Fig. 1 to illustrate the motivation of our algorithms, the same as [18]. The parameters of each task in the parallel application in Fig. 1 are listed in Table 2. The processor parameters are listed in Table 3, such as dynamic power and frequency $\varphi_{k,\mathrm{ind}}$, effective switching capacitance $\xi_{k,e}$ and dynamic power index ϵ_k, and related frequency parameters. The frequency accuracy of each processor is set to 0.01, and the maximum frequency $\phi_{k,\mathrm{max}}$ is 1.0 [18]. Therefore, according to Eq. (5), $E_{\min}(\mathcal{G}) = 20.31$ and $E_{\max}(\mathcal{G}) = 161.99$, respectively. The energy constraint of the standard example application is set to $E_{\min}(\mathcal{G}) \leq E_{\mathrm{cons}}(\mathcal{G}) \leq E_{\max}(\mathcal{G})$, and set $E_{\mathrm{cons}}(\mathcal{G})$ to $E_{\mathrm{HEFT}}(\mathcal{G}) \times 0.5 = 80.995$. Table 4 shows the task allocation of EMREC algorithm for the standard example application. The results of the standard example application using the MRPEC algorithm are shown in Table 5.

The actual energy consumption of the EMREC algorithm is 75.98, and MRPEC is 75.99, both of which are less than the given energy constraint. The schedule length of EMREC and MRPEC is 100.48 and 81.46, respectively. Compared with the EMREC algorithm, the schedule length of MRPEC is reduced by 18.93%. The actual reliability of EMREC is 0.9470, and MRPEC is 0.9226. Compared with EMREC, the reliability of MRPEC is only reduced by 2.24%. The PRP value of EMREC and MRPEC algorithm is 1.0263 and 1.2758, respectively. For the sake of intuition, Figs. 2 and 3

TABLE 4
Scheduling Results Generated by EMREC Algorithm of Parallel Application in Fig. 1

Task	$E_{\mathrm{cons}}(\tau_i)$	$p(\tau_i)$	$f(\tau_i)$	$\mathrm{AST}(\tau_i)$	$\mathrm{AFT}(\tau_i)$	$E(\tau_i)$	$R(\tau_i)$	$\mathrm{RPR}(\tau_i)$
τ_1	8.3982	p_2	0.69	0.0	23.1884	8.26	0.9879	0.5538
τ_2	11.1309	p_1	1.0	41.1884	54.1884	10.79	0.9935	1.2737
τ_3	8.6076	p_2	0.83	23.1884	38.8511	8.49	0.9947	0.9103
τ_4	7.9611	p_2	1.0	38.8511	46.8511	6.72	0.9984	1.5808
τ_5	7.7778	p_2	0.77	46.8511	63.7342	7.70	0.9931	0.6863
τ_6	8.1447	p_1	0.85	54.1884	69.4825	8.10	0.9879	0.8182
τ_7	6.9202	p_1	1.0	69.4825	76.4825	5.81	0.9965	1.5659
τ_8	6.5477	p_1	1.0	76.4825	81.4825	4.15	0.9975	1.9950
τ_9	10.7530	p_2	1.0	70.1884	82.1884	10.08	0.9976	1.3856
τ_{10}	10.8922	p_2	1.0	93.4825	100.4825	5.88	0.9986	2.0923

$E(\mathcal{G}) = 75.98$, $SL(\mathcal{G}) = 100.48$, $R(\mathcal{G}) = 0.9470$, $RPR(\mathcal{G}) = 1.2566$

TABLE 5
Scheduling Results Generated by MRPREC Algorithm of Parallel Application in Fig. 1

Task	$E_{\text{cons}}(\tau_i)$	$p(\tau_i)$	$f(\tau_i)$	$\text{AST}(\tau_i)$	$\text{AFT}(\tau_i)$	$E(\tau_i)$	$R(\tau_i)$	$\text{RPR}(\tau_i)$
τ_1	8.3982	p_3	0.90	0.0	10.0	8.38	0.9876	1.2839
τ_2	11.0141	p_1	1.0	33.4583	46.4583	10.79	0.9935	1.2737
τ_3	8.4872	p_1	0.96	22.0	33.4583	8.49	0.9935	1.2428
τ_4	7.8443	p_2	1.0	19.0	27.0	6.72	0.9984	1.5808
τ_5	7.6610	p_3	0.76	10.0	23.1579	7.55	0.9745	0.8641
τ_6	8.1837	p_3	0.88	23.1579	33.3851	8.16	0.9865	1.2218
τ_7	6.9095	p_1	1.0	46.4583	53.4583	5.81	0.9965	1.5659
τ_8	6.5371	p_1	1.0	54.0	59.0	4.15	0.9975	1.9950
τ_9	10.7424	p_2	1.0	62.4583	74.4583	10.08	0.9976	1.3856
τ_{10}	10.8815	p_2	1.0	74.4583	81.4583	5.88	0.9986	2.0923

$E(\mathcal{G}) = 75.99$, $SL(\mathcal{G}) = 81.4583$, $R(\mathcal{G}) = 0.9266$, $RPR(\mathcal{G}) = 1.5167$

describe the scheduling Gantt chart of EMREC and MRPEC, where the arrows are communication messages. Correspondingly, the energy consumption of the RMEC algorithm is 20.31, the reliability value is 0.192, the schedule length is 80, and the RPR value is 0.0653. The energy consumption generated by the MREC algorithm is 80.99, the reliability value is 0.78, the schedule length is 154.78, and the RPR value is 0.6152. It can be seen from this example that the proposed EMREC and MRPEC algorithms have higher reliability and RPR values than previously algorithms, which indicates the effectiveness of our algorithms.

6 Experiments

To better illustrate the superiority of the proposed algorithm, further comparative experiments are carried out. RMEC[26], MREC[18], EMREC and MRPEC algorithms are used to participate in the comparison in our experiments. RMEC [26] traverses all processor and frequency combinations, and achieved by selecting the appropriate combination according to the reliability maximum energy conservative function. In [18], MREC pre-allocated the minimum energy consumption for each task. Due to the extreme pre-allocation method of RMEC, the allocatable energy is divided by high-priority tasks, resulting in a relatively pessimistic actual reliability.

6.1 Experimental Metrics

The performance indicators selected for comparison are the actual reliability $R(\mathcal{G})$ (Eq. (10)), the final schedule length $\text{SL}(\mathcal{G})$ (Eq. (14)), and the reliability performance rate $\text{RPR}(\mathcal{G})$ value (Eq. (30)). The processor parameters in our experiments are the same as [18]. The detailed processor and application parameters ranges are set as follows: $10\text{ ms} \le w_{i,k} \le 100\text{ ms}, 10\text{ ms} \le c_{i,j} \le 100\text{ ms}, 0.03 \le \varphi_{k,\text{ind}} \le 0.07, 0.8 \le \xi_{k,\text{ef}} \le 1.2, 2.5 \le \epsilon_k \le 3.0$. The frequency of the processor is discrete and the precision is 0.01GHz. A simulated heterogeneous multi-processor platform with 64 processors is adopted to execute all parallel applications.

We use two typical real-world parallel applications, Fast Fourier Transform (FFT) and Gaussian Elimination (GE), to verify the effectiveness of our algorithms. FFT and GE are high parallelism and low parallelism respectively, which are widely used in high-performance computing systems [29]. In addition, we use the parallel applications generated by the random generator provided by [34] to verify our results under different application scales, different energy constraints, different heterogeneity, and different processors.

6.2 FFT Parallel Applications

In our experiments, the parameter ρ is used to indicate the size of the applications. For FFT parallel applications, the total number of tasks is $|\mathcal{T}| = (2 \times \rho - 1) + \rho \times \log_2 \rho$, where $\rho = 2^y$ [29]. Fig. 4a shows an example of the FFT parallel application with $\rho = 4$.

6.2.1 Varying Number of Tasks

Experiment 1. We fix $\lambda = 0.5$ (ie $E_{\text{cons}}(\mathcal{G}) = E_{\text{HEFT}}(\mathcal{G}) \times 0.5$), and change the scale of the parallel applications from $\rho = 8$ ($|\mathcal{T}| = 39$, small scale) to $\rho = 256$ ($|\mathcal{T}| = 2559$, large scale). The RMEC, MREC, EMREC and MRPEC algorithms can all

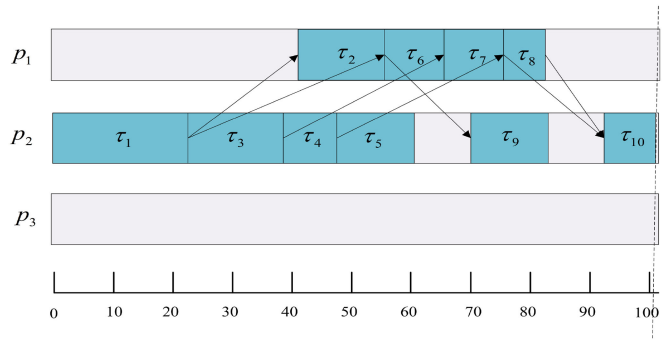


Fig. 2. Scheduling Gantt chart generated by EMREC of parallel application in Fig. 1 .

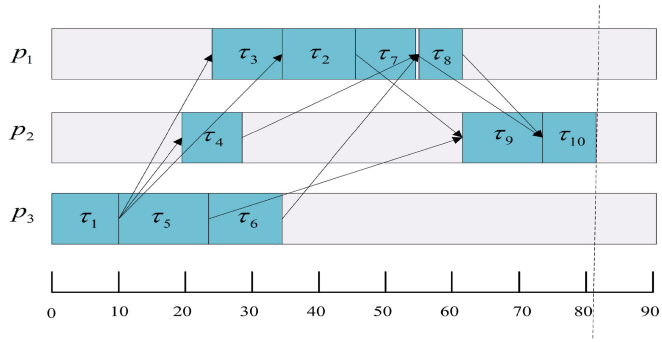
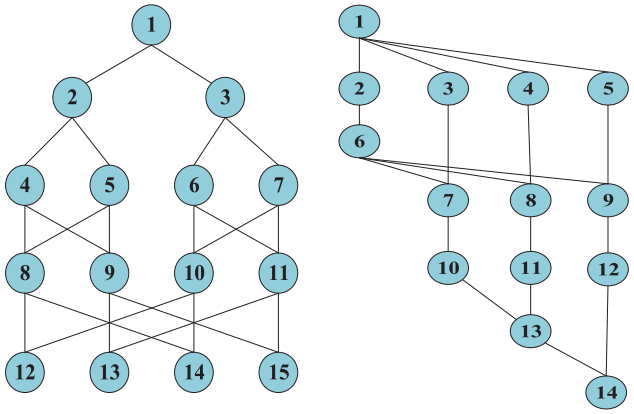


Fig. 3. Scheduling Gantt chart generated by MRPEC of parallel application in Fig. 1 .



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

Fig. 4. Example of real parallel applications.

satisfy the given energy limit, so we have not listed the results. The experimental results are listed in Table 6.

As the number of tasks increases, the RMEC algorithm does not significantly increase the schedule length, but its actual reliability is greatly reduced, resulting in a substantial decrease in the RPR value. At the same time, RMEC generally maintains a low RPP value. As the number of tasks increases, the schedule length of the MREC algorithm will increase dramatically. This is due to MREC's pessimistic allocation strategy. The allocatable energy consumption is divided by the higher-priority tasks, which causes the lower-priority tasks to find tasks that can meet the energy constraints. There are fewer and fewer available processors, which leads to an extension of the schedule length. The actual reliability of the MREC algorithm also decreases as the number of tasks increases, but its downward trend is slower than that of the RMEC algorithm. The Table 6 reflects that the RPR value of MREC is higher than that of RMEC.

The schedule length of the proposed EMREC algorithm also increases as the number of tasks increases. However, due to the reasonable energy pre-allocation method based on the EDR and the MREC algorithm, the EMREC algorithm is significantly more effective in terms of schedule length, which can almost save time. In terms of reliability, the EMREC algorithm is always better than MREC, but the actual reliability declines slowly, which shows the superiority of the proposed EMREC algorithm. Compared with the MREC and EMREC algorithms, the MRPEC algorithm has the shortest schedule length, and as the number of tasks increases, the growth trend becomes slower. Compared with the MREC algorithm, the maximum saving is 3.49 times, and compared with the EMREC algorithm, the maximum saving is 1.97 times. In terms of actual reliability, the MRPEC algorithm is slightly lower than the EMREC algorithm, and is always better than the RMEC and MREC algorithms, and maintains high reliability. In terms of reliability performance, compared with RMEC, MREC and EMREC algorithms, MRPEC maintains the highest RPR value. From Table 6 and the above analysis, we can get the superiority and effectiveness of the proposed EMREC algorithm and MRPEC algorithm.

6.2.2 *Varying Energy Constraints*

Experiment 2. In this experiment, we use the FFT parallel application to conduct experiments under different energy constraints on the same size FFT graph. The task scale of the FFT application is set to $\rho = 64$ ($|\mathcal{T}| = 511$). We compare different scheduling algorithms in terms of schedule length, actual reliability, and PRP values. For comparison, we take $E_{\text{HEFT}}(\mathcal{G})$ as the standard, and set the energy constraints range from $E_{\text{HEFT}}(\mathcal{G}) \times 0.3$ to $E_{\text{HEFT}}(\mathcal{G}) \times 0.8$, namely, the energy constraint changes from strict to loose. Table 7 shows the relevant experimental results.

TABLE 6
The Fast Fourier Transform (FFT) Application Results by Varying Different ρ

ρ	Task	$E_{\text{cons}}(\mathcal{G})$	RMEC [26]			MREC[18]			EMREC			MRPEC		
			SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)
8	39	536.17	476.00	0.9135	1.1519	627.00	0.9769	2.7173	619.00	0.9992	4.0719	602.00	0.9985	**5.3002**
16	95	1100.01	595.00	0.8258	1.0019	946.82	0.9366	2.0333	832.73	0.9974	4.4164	794.65	0.9949	**5.2189**
32	223	2724.31	782.00	0.6244	0.7775	1227.14	0.8387	1.9078	961.36	0.9941	4.2312	952.41	0.9899	**5.2566**
64	511	6058.00	1033.00	0.3278	0.4101	2013.00	0.7311	1.7501	1220.46	0.9877	4.2844	1189.31	0.9788	**5.2871**
128	1151	11143.88	1106.00	0.0902	0.1136	3158.38	0.3885	0.8325	1862.47	0.9657	4.0666	1350.22	0.9419	**4.9023**
256	2559	21213.69	1316.00	0.0042	0.0053	6092.00	0.0628	0.1167	3449.66	0.8891	3.4354	1747.88	0.8101	**3.7959**

TABLE 7
The Fast Fourier Transform (FFT) Application Results by Fix $\rho = 64$

λ	Task	$E_{\text{cons}}(\mathcal{G})$	RMEC [26]			MREC[18]			EMREC			MRPEC		
			SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)
0.3	511	3446.02	904.00	0.3830	0.4567	2206.92	0.4973	0.7249	1581.53	0.9476	2.8932	1258.50	0.9112	**3.4819**
0.4	511	4594.69	904.00	0.3830	0.4567	2138.64	0.5676	0.9287	1368.80	0.9721	3.6016	1225.02	0.9536	**4.4178**
0.5	511	5743.37	904.00	0.3830	0.4567	1965.46	0.6658	1.2718	1305.45	0.9832	4.1368	1171.76	0.9735	**5.1733**
0.6	511	6892.04	904.00	0.3830	0.4567	1811.62	0.7750	1.8000	1367.63	0.9879	4.0767	1196.00	0.9768	**5.3067**
0.7	511	8040.71	904.00	0.3830	0.4567	1748.00	0.8890	2.5408	1506.17	0.9901	3.8027	1196.00	0.9768	**5.3067**
0.8	511	9189.38	904.00	0.3830	0.4567	1543.00	0.9909	3.5354	1576.87	0.9910	3.5657	1196.00	0.9768	**5.3067**

TABLE 8
The Gaussian Elimination Application Results by Varying Different ρ

λ	Task	$E_{\text{cons}}(\mathcal{G})$	RMEC [26]			MREC [18]			EMREC			MRPEC		
			SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)
9	44	495.49	968.00	0.8804	1.1106	1772.19	0.8997	1.0881	1168.22	0.9983	4.0222	1242.00	0.9973	**5.3334**
13	90	957.88	1275.00	0.7441	1.0139	2739.85	0.8377	1.0523	1792.69	0.9965	3.0518	1743.21	0.9940	**5.1102**
21	230	1889.34	2194.00	0.5122	0.6457	4864.35	0.5559	0.6824	3165.33	0.9823	3.0721	2799.74	0.9680	**4.3391**
31	495	4898.41	3145.00	0.2516	0.3174	9254.72	0.3174	0.3930	4765.65	0.9760	3.3665	4173.47	0.9611	**4.8318**
47	1127	11937.60	5282.00	0.0411	0.0516	18690.13	0.0682	0.0823	8298.38	0.9555	3.3250	6867.24	0.9245	**4.8100**
71	2555	28840.57	7410.00	6.0E-4	7.0E-4	39122.22	0.0021	0.0025	20256.11	0.9138	2.9971	10270.79	0.8588	**4.5882**

The RMEC algorithm does not pay attention to the maximum reliability algorithm, but finds the best reliability energy function, which shows the same schedule length, reliability and RPR value under different energy constraints. The schedule length of the MREC algorithm decreases with the increase of energy consumption, and the reliability increases with the increase of energy consumption. In other words, the schedule length of the MREC algorithm is inversely proportional to the given energy consumption, and its reliability is directly proportional to the given energy consumption. The RPR value of the MREC algorithm increases with the increase of energy. This is because as the given energy increases, the processor space and frequency space that MREC can search will also increase. Because reliability is proportional to task execution time, the shorter the scheduling time, the higher the reliability.

As the energy of the EMREC algorithm increases, the schedule length remains relatively stable, but in general it is better than the MREC algorithm. The reliability of EMREC increases with the increase in energy consumption, which shows an overwhelming advantage over MREC. In terms of RPR value, EMREC is the highest, which is 3.25 times that of MREC. The MRPEC schedule length is significantly better than the MREC and EMREC algorithms. Compared with MREC, it is reduced by up to 43%, and compared with EMREC by up to 24%. In terms of reliability, MRPEC is generally better than the MREC algorithm, slightly lower than the EMRC algorithm, and maintains a high reliability value above 0.9. When the energy constraint is 0.6, it shows convergence characteristics. Compared with other algorithms, MRPEC always shows the highest PRP value. This experiment illustrates the advantages of the proposed EMREC and MRPEC algorithms in maximizing reliability and balancing reliability performance in parallel applications with different energy constraints.

6.3 GE Applications

In order to further verify the performance of the proposed algorithm, we take another important practical parallel application (i.e., the GE application) as the experimental object. For GE applications, the number of tasks with the scale parameter ρ can be calculated by $|\mathcal{T}| = \frac{\rho^2+\rho-2}{2}$. Fig. 4b shows an example of a GE parallel application with $\rho = 5$.

6.3.1 Varying Number of Tasks

Experiment 3. In this section, we fix $E_{\text{cons}}(\mathcal{G})$ to $E_{\text{HEFT}}(\mathcal{G}) \times 0.5$, and execute the algorithms in applications from small scale ($\rho = 9$, $|\mathcal{T}| = 44$,) to large scale 71 ($\rho = 71$, $|\mathcal{T}| = 2555$). The ratio corresponds to approximately the same scale of FFT parallel applications in Experiment 1. Correspondingly, the actual energy consumption of RMEC, MREC, EMREC and MRPEC can satisfy the energy consumption limit. As the number of tasks increases, the length of RMEC scheduling increases rapidly, while reliability and RPR decrease rapidly. When used in large-scale applications, the reliability of RMEC is very poor and unacceptable. This is because the parallelism of GE applications is low, and RMEC is not the most reliable method. The schedule length of the MREC algorithm increases sharply with the increase of tasks, and the schedule length is very poor. The actual reliability of the MREC algorithm also decreases as the number of tasks increases, and shows a rapid trend, which also leads to a decrease in PRP. Generally, the MREC algorithm is due to the reliability of RMEC and RPR, but it is weaker than RMEC in terms of schedule length. This is due to the extreme pre-allocation strategy of the MREC algorithm. This strategy causes the allocatable energy to be allocated by high-priority tasks, while the feasible processors and frequencies for low-priority tasks are narrowed, and the parallelism of GE applications is low.

Compared with MREC, the proposed EMREC algorithm has advantages in schedule length, which can save 55.6%. At the same time, EMREC can obtain high reliability higher than 0.9 under different number of tasks, and as the number of tasks increases, it shows a slow downward trend. The RPR of the EMREC algorithm is attributed to RMEC and MREC, and the highest is nearly 4 times. The schedule length of the proposed MRPEC algorithm shows superiority, saving 73.8% compared with MREC and 49.3% compared with EMREC. At the same time, MRPEC remains always higher than the high reliability value of RMEC and MREC, and maintains a high reliability value higher than 0.85, which is only 6% lower than EMREC. The above experimental analysis shows that, compared with RMEC and MREC, the proposed EMREC and MRPEC algorithms also have good adaptability and superiority in actual low-parallel applications. The relevant experimental results are given in Table 8.

6.3.2 Varying Energy Constraints

Experiment 4. In this experiment, we use GE application and conduct various energy constraints experiments on the same scale. We compare the RMEC, MREC, EMREC, and MRPEC algorithms in terms of schedule length, the actual reliability, and the RPR values. We set $\rho = 31$ ($|\mathcal{T}|$=495) for the GE applications. This setting is to approximate the number of tasks used by the FFT applications in Experiment 3,

TABLE 9
The Gaussian Elimination Application Results by Fix $\rho = 32$

λ	Task	$E_{\text{cons}}(\mathcal{G})$	RMEC [26]			MREC[18]			EMREC			MRPEC		
			SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)	SL($\mathcal{G}$)	$R(\mathcal{G})$	RPR($\mathcal{G}$)
0.3	527	3126.92	3463.0	0.2852	0.3597	6227.17	0.3925	0.5983	5118.74	0.9366	2.6312	4912.54	0.8908	**3.1174**
0.4	527	4169.23	3463.0	0.2852	0.3597	6068.11	0.4687	0.8018	4861.39	0.9670	3.3631	4728.83	0.9420	**4.0098**
0.5	527	5211.54	3463.0	0.2852	0.3597	5879.63	0.5560	1.1038	4689.11	0.9803	3.9406	4606.99	0.9655	**4.7373**
0.6	527	6253.84	3463.0	0.2852	0.3597	5511.47	0.6886	1.6737	4797.00	0.9868	4.2843	4517.84	0.9794	**5.2355**
0.7	527	7296.15	3463.0	0.2852	0.3597	5326.76	0.8142	2.4608	4674.02	0.9904	4.3011	4517.00	0.9795	**5.2383**
0.8	527	8338.46	3463.0	0.2852	0.3597	4790.32	0.9917	4.0638	4788.00	0.9921	4.0815	4517.00	0.9795	**5.2383**

so as to roughly compare the experimental effects of two different parallel applications. For the energy constraint range, we set $E_{\text{cons}}(\mathcal{G}) = E_{\text{HEFT}}(\mathcal{G}) \times \lambda$, where λ is from 0.3 to 0.8. Table 9 lists the relevant experimental results.

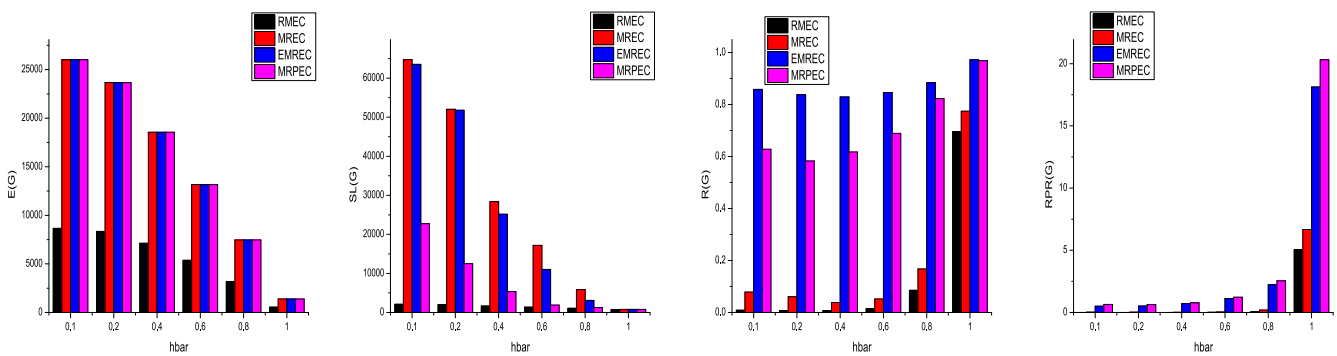
As discussed in Experiment 2, RMEC has the same schedule length, actual reliability value and reliability performance ratio under different energy constraints. The schedule length of the MREC algorithm decreases as the energy constraint increases, and the reliability increases. The reason is the same as experiment 2. The greater the energy constraint, the more feasible the processor and frequency combination. Generally, MREC is lower than RMEC in terms of schedule length, better than RMEC in terms of actual reliability, and better than RMEC in terms of RPR. The total length of EMREC is better than that of MREC, and it shows a downward trend as the energy increases. The EMREC algorithm can always find a high reliability value higher than 0.93, which is much better than the RMEC and MERC algorithms. At the same time, the EMREC algorithm has a higher RPR. Compared with MREC and MREC, the proposed MRPEC algorithm has an optimal schedule length. It shows a decreasing trend as the energy base increases, and shows a trend of convergence after $\lambda = 0.7$. Due to the existence of the RMEC and MREC algorithms, in most cases, the actual reliability of the MRPEC algorithm always maintains a high reliability value, and always maintains a high reliability value above 0.89. Under different energy constraints, the MRPEC algorithm always maintains the highest RPR value.

6.4 Randomly Generated Parallel Applications

For general purpose, randomly parallel applications generated by the task graph generator [34] are considered, which is the same as [18] and [26]. Given that the target platform is composed of heterogeneous computing systems, heterogeneity is an important factor affecting energy requirements. For randomly generated parallel applications, as long as the heterogeneity factor value is adjusted, heterogeneity can be easily achieved [26]. Meanwhile, experiments with different processor numbers can verify the resource utilization and scalability of the algorithm.

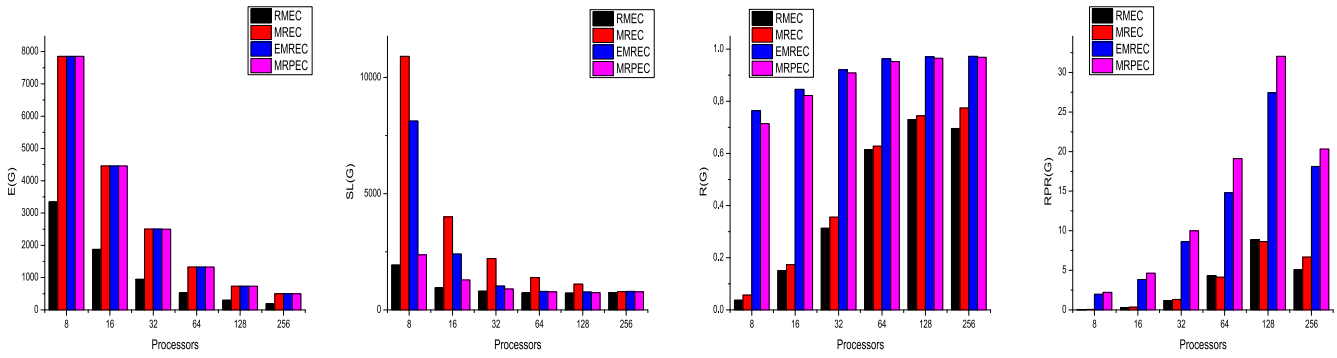
6.4.1 *Varying Heterogeneity*

Experiment 5. In this scetion, we fix the number of tasks and energy constraints, and observed the experimental results of different heterogeneous applications. We set the task benchmark execution time to 100 ms, the number of tasks $|\mathcal{T}| = 551$, $E_{\text{cons}}(\mathcal{G})$ is set to $E_{\text{HEFT}}(\mathcal{G}) \times 0.5$. Then, we set the heterogeneity factor $\hbar$ in the range of $\{0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$, that is, from low heterogeneity to high heterogeneity. The detailed experimental results are given in Fig. 5. As shown in Fig. 5, the schedule length of RMEC decreases with the increase of heterogeneity, while maintaining a low schedule length. However, its reliability value is very poor, almost completely unreliable, and its RPR value is very low. This is because when the heterogeneity is low, the execution time required for each task is very long, which is close to the benchmark execution time. There is a certain inverse relationship between execution time and reliability. The schedule length of MREC decreases as the heterogeneity factor increases, and the performance under low heterogeneity is too poor to be tolerated. MREC shows extremely poor reliability values, but overall, it is slightly better than RMEC, increases with increasing heterogeneity factor, and RPR value is very poor. This is because low heterogeneity applications usually require a long execution time. In



(a) Energy (b) schedule length (c) Reliability (d) RPR

Fig. 5. Scheduling results of comparison algorithms executed in HCS with varying heterogeneity ($|\mathcal{T}| = 551$).



(a) Energy (b) schedule length (c) Reliability (d) RPR

Fig. 6. Scheduling results of comparison algorithms executed in HCS with different processor scales ($|\mathcal{T}| = 551$).

addition, due to the unreasonable energy distribution method of MREC, the allocatable energy consumption is divided by high-priority applications, resulting in low-priority tasks that can only be executed at low frequency and low energy consumption. Insufficient power, low frequency will reduce reliability.

Like MREC, the schedule length of EMREC decreases as the heterogeneity factor increases, and the schedule length is relatively poor, but EMREC has high reliability and can maintain a high level of 0.8 or more when the heterogeneity is extremely low. This is because low heterogeneity applications require longer execution time. In order to find the standard that meets the maximum reliability objective, EMREC puts tasks on high-reliability processors for execution, which leads to the need for longer scheduling times. The schedule length of MRPEC also decreases as the heterogeneity factor increases, and remains within a relatively acceptable range. Compared with RMEC, it can save up to 64.9%, and compared with EMREC, it can save up to 67.3%. In addition, regardless of low heterogeneity or high heterogeneity, MRPEC maintains a high reliability value and at the same time has the highest RPR value. Experiments results show that the MRPEC algorithm has good adaptability under different degrees of heterogeneity.

6.4.2 Varying Number of Processors

Experiment 6. In this section, we fix the number of tasks and energy constraints and observe the experimental results of comparative algorithms under different processor scales. We set the execution time of the benchmarks to 100 ms. The number of tasks is set to $|\mathcal{T}| = 551$, and $E_{\text{cons}}(\mathcal{G})$ is set to $E_{\text{HEFT}}(\mathcal{G}) \times 0.5$. Then, we set the heterogeneous factor $\hbar = 1.0$. The number of processors $\mathcal{P}$ is varies within the range of $\{8, 16, 32, 64, 128, 256\}$. The relevant experimental results are given in Fig. 6.

It can be seen from Fig. 6. that due to the increase in processing capacity, the schedule length decreases as the number of processors increases. Since the increase in optional high-reliability processors, the actual reliability of parallel applications also increases with the increase of processors. Overall, EMREC has the highest reliability, EMEC has the lowest reliability, and the reliability of MRPEC remains close to that of EMREC. RMEC has the lowest RPR, MRPEC has the highest PRP, and EMREC's PRP is better than MREC. Compared with MREC, MRPEC can save up to 78.3%. Simultaneously, compared with EMREC, MRPEC can save up to 70.8%. Experimental results show that the proposed algorithms is effective on heterogeneous systems with different processing capabilities.

6.5 Discussion

In summary, combining the above experimental results on real-world and randomly generated applications, the proposed EMREC and MRPEC algorithms are effective in maximizing reliability while satisfying the given energy constraints. We make the following observations

- In terms of energy consumption and schedule length, RMEC is superior to MREC, EMREC, and MRPEC, but it is relatively the worst in terms of actual reliability.
- MREC has higher reliability than RMEC, but the schedule length is worse. In general, the schedule length and reliability of MREC are lower than EMREC and MRPEC, and they have similar energy consumption.
- Compared with other algorithms, EMREC has the largest reliability and is better than MREC in terms of schedule length. EMREC can adapt well to different energy-constrained environments.
- Compared with RMEC, MREC, and EMREC, MRPEC has the highest PRP and saves $49\% - 78\%$ of schedule length. In terms of reliability, MRPEC has an overwhelming advantage over MREC, and is quite close to EMREC.

7 Conclusion

In this paper, we addressed the problem of maximum reliability and maximum Reliability Performance Ratio(RPR) for energy-constrained parallel applications in HCSs. We introduced the concept of Energy Demand Rate (EDR), which reasonably pre-allocated energy for each task in parallel applications. Then, we proposed the EDR-aware maximize reliability energy-constrained scheduling algorithm of parallel application to solve the defined problem. Moreover, we defined the reliability performance ratio to evaluate the balance between reliability and performance. RPR not only can be used as an optimization objective, but also as an index to evaluate the performance of different algorithms. On this basis, we further proposed the PRP-aware energy-

constrained scheduling algorithm to achieve maximum reliability performance ratio. We conducted extensive experiments on real-world applications and random generation applications, the experimental results show that our algorithms can achieve high reliability as well as high performance. In the future, we will further consider more advanced power consumption models and the resource cost optimization of DAG-based parallel applications on HCSs.

Acknowledgments

The authors wish to express their sincere appreciation to all the anonymous reviewers and the editor for their worthwhile and constructive comments.

References

[1] Q. Chen and M. Guo, *Task Scheduling for Multi-core and Parallel Architectures - Challenges, Solutions and Perspectives*, Berlin, Germany: Springer, 2017.
[2] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang, and K. Li, "Energy-efficient resource utilization for heterogeneous embedded computing systems," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1518–1531, Sep. 2017.
[3] R. E. Kavanagh, K. Djemame, J. Ejarque, R. M. Badia, and D. Garc ía-Pérez, "Energy-aware self-adaptation for application execution on heterogeneous parallel architectures," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 1, pp. 81–94, First quarter 2020.
[4] H. Djigal, J. Feng, J. Lu, and J. Ge, "IPPTS: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1057–1071, May 2021.
[5] S. Z. Sheikh and M. A. Pasha, "Energy-efficient multicore scheduling for hard real-time systems: A survey," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 6, pp. 94:1–94:26, 2019.
[6] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, and X. Xu, "A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems," *World Wide Web*, vol. 23, no. 2, pp. 1275–1297, 2020.
[7] C. Jeong and H. Son, "Cooperative transmission of energy-constrained IoT devices in wireless-powered communication networks," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3972–3982, Mar. 2021.
[8] Z. Zhou *et al.*, "An adaptive energy-aware stochastic task execution algorithm in virtualized networked datacenters," *IEEE Trans. Sustain. Comput.*, early access, Sep. 24, 2021, doi: 10.1109/TSUSC.2021.3115388.
[9] M. Adhikari, T. Amgoth, and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 68:1–68:36, 2019.
[10] G. Xie, X. Xiao, H. Peng, R. Li, and K. Li, "A survey of low-energy parallel scheduling algorithms," *IEEE Trans. Sustain. Comput.*, early access, Feb. 9, 2021, doi: 10.1109/TSUSC.2021.3057983.
[11] S. Hajiamini, B. Shirazi, A. Crandall, and H. Ghasemzadeh, "A dynamic programming framework for DVFS-based energy-efficiency in multicore systems," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 1, pp. 1–12, First quarter 2020.
[12] M. Chadha and M. Gerndt, "Modelling DVFS and UFS for region-based energy aware tuning of HPC applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2019, pp. 805–814.
[13] S. Hajiamini, B. Shirazi, A. Crandall, and H. Ghasemzadeh, "A dynamic programming framework for DVFS-based energy-efficiency in multicore systems," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 1, pp. 1–12, First quarter 2020.
[14] M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry, "Performance evaluation and energy efficiency of high-density HPC platforms based on intel, AMD and ARM processors," in *Proc. Eur. Conf. Energy Efficiency Large Scale Distrib. Syst.*, 2013, pp. 182–200.
[15] G. Xie *et al.*, "Reliability enhancement toward functional safety goal assurance in energy-aware automotive cyber-physical systems," *IEEE Trans. Ind. Inform.*, vol. 14, no. 12, pp. 5447–5462, Dec. 2018.
[16] M. Ansari, J. Saber-Latibari, M. Pasandideh, and A. Ejlali, "Simultaneous management of peak-power and reliability in heterogeneous multicore embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 623–633, Mar. 2020.
[17] L. Zhang, K. Li, W. Zheng, and K. Li, "Contention-aware reliability efficient scheduling on heterogeneous computing systems," *IEEE Trans. Sustain. Comput.*, vol. 3, no. 3, pp. 182–194, Third quarter 2018.
[18] X. Xiao, G. Xie, C. Xu, C. Fan, R. Li, and K. Li, "Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems," *J. Comput. Sci.*, vol. 26, pp. 344–353, 2018.
[19] 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, Aug. 2011.
[20] B. Salami, H. Noori, and M. Naghibzadeh, "Fairness-aware energy efficient scheduling on heterogeneous multi-core processors," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 72–82, Jan. 2021.
[21] X. Tang, W. Shi, and F. Wu, "Interconnection network energy-aware workflow scheduling algorithm on heterogeneous systems," *IEEE Trans. Ind. Inform.*, vol. 16, no. 12, pp. 7637–7645, Dec. 2020.
[22] J. Song, G. Xie, R. Li, and X. Chen, "An efficient scheduling algorithm for energy consumption constrained parallel applications on heterogeneous distributed systems," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl., IEEE Int. Conf. Ubiquitous Comput. Commun.*, 2017, pp. 32–39.
[23] X. Tang, K. Li, M. Qiu, and E. H. Sha, "A hierarchical reliability-driven scheduling algorithm in grid systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 4, pp. 525–535, 2012.
[24] A. Naithani, S. Eyerman, and L. Eeckhout, "Reliability-aware scheduling on heterogeneous multicore processors," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 397–408.
[25] S. Wang, K. Li, J. Mei, G. Xiao, and K. Li, "A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems," *J. Grid Comput.*, vol. 15, no. 1, pp. 23–39, 2017.
[26] 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," *Inf. Sci.*, vol. 319, pp. 113–131, 2015.
[27] N. Kumar, J. Mayank, and A. Mondal, "Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 871–885, Apr. 2020.
[28] J. Zhou *et al.*, "Resource management for improving soft-error and lifetime reliability of real-time MPSoCs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 12, pp. 2215–2228, Dec. 2019.
[29] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling forheterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[30] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2004, pp. 35–40.
[31] J. Li, G. Xie, K. Li, and Z. Tang, "Enhanced parallel application scheduling algorithm with energy consumption constraint in heterogeneous distributed systems," *J. Circuits Syst. Comput.*, vol. 28, no. 11, pp. 1950190:1–1950190:23, 2019.
[32] A. Naithani, S. Eyerman, and L. Eeckhout, "Optimizing soft error reliability through scheduling on heterogeneous multicore processors," *IEEE Trans. Comput.*, vol. 67, no. 6, pp. 830–846, Jun. 2018.
[33] S.-H. Jeon, J.-H. Cho, Y. Jung, S. Park, and T.-M. Han, "Automotive hardware development according to ISO 26262," in *Proc. 13th Int. Conf. Adv. Commun. Technol.*, 2011, pp. 588–592.
[34] Task graph generator, 2015. [Online]. Available: https://sourceforge.net/projects/taskgraphgen/

Jiwu Peng is currently working toward the PhD degree in computer science and technology with the College of Information Science and Engineering, Hunan University, Changsha, China. His research interests include heterogeneous distributed computing systems, embedded systems and cyber-physical systems, cloud computing, mobile edge computing, and machine learning.

Kenli Li (Senior Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar with the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a Cheung Kong professor of computer science and technology with Hunan University, the dean of the College of Information Science and Engineering, Hunan University. His major research interests include high-performance computing, parallel and distributed processing, big data management, and cloud computing. He has published more than 260 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Cloud Computing*, ICPP, ICDCS, etc. He has served on the editorial board of the *IEEE Transactions on Computers*. He is an outstanding member of the CCF.

Jianguo Chen received the PhD degree in computer science and technology from Hunan University, China, in 2018. He is currently a research scientistic with Institute for Infocomm Research, Agency for Science Technology and Research, Singapore. He was a visiting PhD student with the University of Illinois at Chicago from 2017 to 2018. He was a postdoctoral fellow with the University of Toronto, Canada, and Hunan University, China from 2018 to 2020. His major research interests include distributed computing, machine learning, deep learning, and intelligence transportation systems.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a national distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 820 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than 60 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the worlds top 10 most influential scientists in parallel and distributed computing based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.