Journal of Systems Architecture 152 (2024) 103173

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Energy-efficient scheduling for parallel applications with reliability and time constraints on heterogeneous distributed systems

Hongzhi Xu [a,*], Binlian Zhang [a], Chen Pan [b], Keqin Li [c]

[a] College of Computer Science and Engineering, Jishou University, Zhangjiajie 427000, China
[b] Department of Electrical & Computer Engineering, The University of Texas at San Antonio (UTSA), San Antonio, TX 78249, USA
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Dynamic voltage and frequency scaling
Energy consumption
Execution time constraint
Parallel application
Reliability requirement

ABSTRACT

Reliability is a crucial index of the system, and many safety-critical applications have reliability requirements and deadline constraints. In addition, in order to protect the environment and reduce system operating costs, it is necessary to minimize energy consumption as much as possible. This paper considers parallel applications on heterogeneous distributed systems and proposes two algorithms to minimize energy consumption for meeting the deadline and satisfying the reliability requirement of the applications. The first algorithm is called minimizing scheduling length while satisfying the reliability requirement (MSLSRR). It first transforms the reliability requirement of the application into the reliability requirement of the task and then assigns the task to the processor with the earliest finish time. Since the reliability generated by MSLSRR is often higher than the reliability requirement of the application, and the scheduling length is also less than the deadline, an algorithm called improving energy efficiency (IEE) is designed, which redefined the minimum reliability requirement for the task and applied dynamic voltage and frequency scaling (DVFS) technique for energy conservation. The proposed algorithms are compared with existing algorithms by using real parallel applications. Experimental results demonstrate that the proposed algorithms consume the least energy.

1. Introduction

Heterogeneous distributed platforms are widely used in many fields such as automotive electronics, medical health monitoring, and warehouse logistics, among others. The execution efficiency of parallel applications on such platforms is closely related to task scheduling methods [1]. Therefore, in order to make the resources of heterogeneous platforms more effectively utilized, scheduling algorithms that meet special requirements should be developed. For many safety-critical systems, the reliability and deadline requirements must be satisfied, otherwise, it may lead to disastrous consequences [2]. The reliability of an application (task) is defined as the probability that the application (task) can be executed correctly [3]. There are many safety standards for reliability, such as ISO 26262 and DO-178C [3]. For example, ISO 26262 introduces the concept of Exposure (E) to quantify the likelihood that drivers or other road users may encounter hazardous events under normal driving conditions and further categorizes exposure levels into four distinct classifications: very low (E1), low (E2), medium (E3), and high (E4). The reliability requirements corresponding to exposure levels E1, E2, E3, and E4 are respectively at least 0.99, 0.99, 0.9, and lower than or equal to 0.9 [3]. In general, enhancing the reliability of a system will also increase its energy consumption. For energy conservation and environmental concerns, many systems use dynamic voltage and frequency scaling (DVFS) techniques to reduce processor execution frequency and thus reduce energy consumption [4–7]. However, previous studies have demonstrated that decreasing processor execution frequency can negatively affect system reliability [2,8,9]. In addition, reducing the processor execution frequency will increase the scheduling length of the application, which may result in deadlines no longer being met. Therefore, a tradeoff has to be made between the degree of reliability, scheduling length, and energy consumption. Specifically, in the system design phase, energy consumption should be minimized while ensuring that both reliability and scheduling length requirements are met.

From the perspective of system design, parallel applications are composed of precedence-constrained tasks, which are typically modeled as directed acyclic graphs (DAGs) [5,33,34]. Many energy-efficient algorithms with DVFS techniques have been proposed for DAG-based parallel applications on heterogeneous systems [5,35–37]. However, these algorithms do not consider the application's reliability requirements. While ensuring that the application satisfies the reliability requirement, many researchers have designed algorithms to optimize

* Corresponding author.
E-mail addresses: xuhongzhi9@163.com (H. Xu), zhangbinlian@163.com (B. Zhang), chen.pan@tamucc.edu (C. Pan), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.sysarc.2024.103173
Received 25 November 2023; Received in revised form 4 April 2024; Accepted 1 May 2024
Available online 6 May 2024

Table 1
Review of relevant researches.

System Platform	Reference	Application Model	Optimization Goal	Constraint	Fault-tolerant Technique
homogeneous	Haque et al. [9]	periodic tasks	energy	reliability	replication
	Salehi et al. [10]	DAG	energy	deadline and reliability	N-modular redundancy
	Wu et al. [11], Cui et al. [12]	DAG	energy	deadline and reliability	replication
heterogeneous	Huang et al. [13]	DAG	energy	deadline and original reliability	re-execution
	Kumar et al. [14]	periodic tasks	energy	reliability	replication
	Xie et al. [3]	DAG	resource	reliability	
	Xu et al. [15], Ye et al. [16]	DAG	energy	reliability	
	Xie et al. [17], Han et al. [18]	DAG	energy	reliability	replication
	Xie et al. [19], Wang et al. [20]	DAG	redundancy	reliability	replication
	Liu et al. [21], Mao et al. [22]	DAG	reliability	deadline	replication
	Zhang et al. [23]	DAG	energy and reliability	deadline	
	Huang et al. [24]	DAG	schedule length and energy	reliability	
	Chen et al. [25]	DAG	schedule length	energy	
	Zhang et al. [26]	DAG	reliability	energy	
	Peng et al. [27]	DAG	schedule length and reliability	energy	
	Tang et al. [28]	DAG	deadline miss ratio	reliability and ECU cost	replication
	Han et al. [29], Kumar et al. [30]	periodic tasks	energy	deadline and reliability	replication
	Zhao et al. [31]	DAG	resource	deadline and reliability	replication
	Hu et al. [32]	DAG	energy	deadline and reliability	
	Ours	DAG	energy	deadline and reliability	

system resource (energy) consumption [3,15,17]. However, these algorithms do not consider the application's deadline constraints.

At present, several approaches address both reliability requirements and deadline constraints. For instance, there are methods to optimize energy consumption for parallel applications on homogeneous platforms [10–12], strategies to optimize energy consumption for periodic tasks on heterogeneous platforms [29,30], as well as techniques to optimize energy (resource) consumption for parallel applications on heterogeneous platforms [31,32].

The problem addressed in this paper is akin to those studied in [31,32], which reduces system energy consumption while meeting deadlines and satisfying the reliability requirements of the applications. In [31], the author did not discuss using DVFS techniques to reduce energy consumption. In [32], the reliability requirements of the application are transformed into the reliability requirements of the task. However, the obtained reliability requirements of tasks are not balanced, which does not effectively reduce the scheduling length and save energy consumption. Therefore, this paper proposes several algorithms to overcome these issues, which can further improve energy efficiency and meet relatively tight deadlines or relatively higher reliability requirements compared to [31,32].

This paper focuses on task assignment during the system design phase, and its main contributions are as follows:

(1) This paper transforms the reliability requirement of the application into the reliability requirement of each task and proposes an algorithm that minimizes the scheduling length while satisfying the reliability requirement (MSLSRR) of the application.

(2) When the scheduling length obtained by the MSLSRR algorithm is less than the application's deadline, this paper designs an algorithm to reassign processor and execution frequency for each task, which can further reduce energy consumption while meeting the deadline and reliability requirement.

(3) Experiments with real parallel applications are conducted in different scenarios. Experimental results confirm that the proposed algorithms consume less energy than other approaches.

2. Related work

In recent years, many researchers have studied scheduling problems related to system energy consumption, scheduling length, and reliability. They have explored different optimization problems based on different system platforms, application models, and constraints, and have achieved many excellent results. Table 1 reviews some research related to this paper.

Haque et al. [9] introduced methods to obtain the degree of replication and the corresponding execution frequency for a set of periodic tasks on multi-core platforms. Salehi et al. [10] proposed a two-phase scheme for reducing system energy consumption under the n-modular redundancy mechanism. In [11,12], the authors investigated methods with task replication to improve energy efficiency while meeting the application's deadline and reliability requirement. However, the above studies are all based on homogeneous platforms.

For heterogeneous platforms, many algorithms that optimize energy consumption (resource, or redundancy) while satisfying system reliability requirements have also been investigated [3,15–20]. In [3], the reliability requirement of the application is first transformed into the reliability requirement of each task, and then the system resources are minimized. In [15], the authors proposed two energy-efficient scheduling approaches under reliability constraints. In [16], the authors first decomposed the workflow reliability constraint into the task sub-reliability constraint and then minimized the system energy consumption. In [17,18], the authors adopted an active replication method to meet the reliability requirement of the task. In [19], the authors defined the concept of the reliability increment ratio and then designed a redundancy minimization algorithm. In [20], the authors introduced a fast task assignment approach to find the minimum redundancy of parallel applications with reliability requirements. However, these studies did not take into account the application's deadline. There are also some studies that optimize the reliability or energy consumption of applications while considering deadline constraints [21–23]. Some authors have also designed algorithms to minimize energy consumption while meeting deadlines constraints and reliability requirements [29–32]. Among them, [29,30] are based on periodic tasks, while [31,32] are based on parallel applications. Furthermore, some temperature-aware scheduling techniques have also been proposed, such as FATS-2TC [38], RT-SEAT [39], and TMDS [40].

In recent years, the energy-efficient scheduling technique in standby-sparing systems has attracted the attention of some researchers. The standby-sparing is a special case of a heterogeneous system composed of primary processors and spare processors. Niu et al. [41] proposed

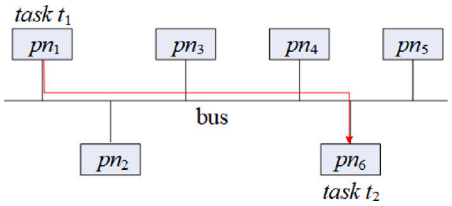


Fig. 1. Distributed system platform.

an energy-aware scheduling algorithm for periodic task sets on a dual processor standby-sparing system. Zhang et al. [42] designed a partitioning scheduling algorithm in which tasks requiring access to shared resources are assigned to the primary processor, while tasks with no resource requirements are assigned to the spare processor. Safari et al. [43] introduced the low energy standby-sparing technique in mixed-criticality systems. Zhang et al. [44] designed an energy-aware scheduling algorithm for periodic real-time task sets with shared resources on a standby-sparing system. These task allocation methods based on standby-sparing systems have lower time complexity and can achieve good results. However, these studies mainly focus on independent tasks or mixed-criticality tasks, which is different from the research in this paper.

The research in this paper is similar to [31,32], which minimize energy consumption while meeting the deadline and reliability requirements of applications with precedence constraints. Compared to [31,32], our proposed method consumes less energy while meeting the relatively tighter deadline or higher reliability requirement.

3. System model and problem formulation

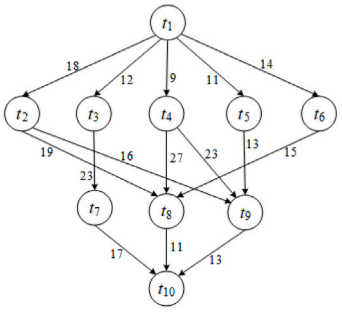
3.1. Heterogeneous system model

The heterogeneous distributed system studied in this work is composed of m DVFS-enabled processor nodes, which are represented by a set as $PN = \{pn_1, pn_2, \ldots, pn_m\}$. The system architecture is shown in Fig. 1, where the processors are connected via a bus [17]. These processor nodes can jointly execute parallel applications, and the execution results generated by any node can be transmitted to other nodes as input. For example, when task t_1 on processor pn_1 is completed, it will send data (message) to its successor task t_2 located in processor pn_6.

3.2. Application model

Parallel applications are composed of tasks constrained by predecessors, which are typically modeled as directed acyclic graphs (DAGs). Similar to [3,15,17,27,33,36], the parallel application is represented as DAG $A = (T, C)$, where T and C are explicated as follows:

$T = \{t_1, t_2, \ldots, t_n\}$ is a vertex set of DAG, which indicates that there are n tasks in parallel application A. C is an edge set of DAG, which indicates the data communication relationship between different tasks. The value of $c_{i,j}$ represents the time required for data transmission from t_i to t_j. If t_i and t_j are assigned to the different processors, then the data transmission time is $c_{i,j}$. For conveniently describing the relationship between tasks in a parallel application, the direct predecessor tasks of t_i are denoted as $pred(t_i)$, and the direct successor tasks of t_i are denoted as $succ(t_i)$. Tasks without precursor tasks and successor tasks are called t_{entry} and t_{exit}, respectively. An example of a parallel application represented by DAG is shown in Fig. 2(a) [3,15,17,27,33,36], which consists of ten tasks and fifteen data transmission edges. The edge $c_{1,2} = 18$ indicates that if t_1 and t_2 are assigned to the different processors, then the data transmission time is 18; otherwise, the data transmission time is 0.



(a) DAG

Task	pn_1	pn_2	pn_3	Rank
t_1	14	16	9	108.0
t_2	13	19	18	77.0
t_3	11	13	19	80.0
t_4	13	8	17	80.0
t_5	12	13	10	69.0
t_6	13	16	9	63.3
t_7	7	15	11	42.7
t_8	5	11	14	35.7
t_9	18	12	20	44.3
t_{10}	21	7	16	14.7

(b) WCET

Fig. 2. An example of a parallel application.

Table 2
Symbol definitions.

Symbol	Definition
$c_{i,j}$	time required for data transmission from t_i to t_j
$w_{i,j}$	worst case execution time of t_i on pn_k
$E_{\text{d}}(t_i, pn_k, f_{k,l})$	dynamic energy consumption of t_i on pn_k with execution frequency $f_{k,l}$
$E_{\text{t}}(t_i, pn_k)$	data transmission energy consumption of t_i on pn_k
$E_{\text{dt}}(t_i)$	sum of dynamic energy consumption and data transmission energy consumption consumed by task t_i
$E_{\text{dt}}(A)$	sum of dynamic energy consumption and data transmission energy consumption consumed by application A
$E_{\text{s}}(A)$	static energy consumption of application A
$E(A)$	total energy consumption of application A
$R(t_i)$	reliability of task t_i
$R_{\text{req}}(t_i)$	reliability requirement of task t_i
$R(A)$	reliability of application A
$R_{\text{req}}(A)$	reliability requirement of application A
$EST(t_i, pn_k)$	earliest start time of task t_i on pn_k
$EFT(t_i, pn_k)$	earliest finish time of task t_i on pn_k
$AET(t_i, pn_k)$	available execution time of t_i on pn_k
$ST(t_i, pn_k)$	actual start time of task t_i on pn_k
$FT(t_i, pn_k)$	actual finish time of task t_i on pn_k
$SL(A)$	scheduling length (execution time) of application A

Because the processors in the system are heterogeneous, the required execution time of the same task on different processors may be different. An $n \times m$ matrix is used to represent the worst-case execution time (WCET) of each task execution on different processors with maximum execution frequency [3,15,17,27,33,36]. $w_{i,k}$ represents the WCET of the task t_i executes on processor pn_k. The WCET of ten tasks on three processors is shown in Fig. 2(b), where the WCET of t_1 on pn_1, pn_2, and pn_3 is 14, 16, and 9, respectively.

Table 2 shows the symbol definitions used in this paper.

3.3. Energy consumption model

Based on [2,5,26,36,45], the power dissipation of the processor operating at frequency f is calculated as

$$P(f) = P_{\text{st}} + h(P_{\text{in}} + P_{\text{dy}}) = P_{\text{st}} + h(P_{\text{in}} + C_{\text{sw}} f^a). \tag{1}$$

In (1), P_{st} and P_{in} represent the static power and the leakage power dissipation, respectively. P_{dy} is the dynamic power dissipation, which is related to the execution frequency of the processor. C_{sw} is the switching capacitance and a indicates the dynamic energy exponent. If the processor is running, $h = 1$; otherwise, $h = 0$.

As P_{st} is always consumed [17], similar to [5,15,17,36], this work focuses on optimizing dynamic energy consumption, but we still include static energy when calculating system energy consumption. Due to the influence of P_{in}, reducing the execution frequency will extend the task execution time, thereby increasing static power consumption [46].

Hence, the minimum energy-efficient frequency, known as the critical frequency, can be calculated as [5,15,17,36]

$$f_{\text{critical}} = \sqrt[a]{\frac{P_{\text{in}}}{(a-1)C_{\text{sw}}}}. \tag{2}$$

When the execution frequency is lower than f_{critical}, the energy consumption generated by the processor will actually increase. Because the execution frequency of the processor can be dynamically adjusted at runtime, assuming the available discrete execution frequency of processor pn_k is $\{f_{k,\min}, \cdots, f_{k,l}, f_{j,l+1}, \cdots, f_{k,\max}\}$. When task t_i executes on processor pn_k with execution frequency $f_{k,l}$, the dynamic energy consumption $E_{\text{d}}(t_i, pn_k, f_{k,l})$ can be calculated by

$$E_{\text{d}}(t_i, pn_k, f_{k,l}) = (P_{k,\text{in}} + C_{k,\text{sw}} {f_{k,l}}^{a_k}) \times w_{i,k} \times \frac{f_{k,\max}}{f_{k,l}}. \tag{3}$$

When the processor adjusts the voltage/frequency, there is a corresponding switching overhead. According to [44,47], when the execution frequency of processor pn_k is adjusted from $f_{k,ini}$ to $f_{k,goal}$, the time overhead can be represented as

$$TO(pn_k, f_{k,ini}, f_{k,goal}) = \beta_k \cdot \left| V_{k,goal} - V_{k,ini} \right|, \tag{4}$$

and the energy overhead can be represented as

$$EO(pn_k, f_{k,ini}, f_{k,goal}) = \chi_k \cdot \left| V_{k,goal}^2 - V_{k,ini}^2 \right|, \tag{5}$$

where β_k and χ_k are constants, with $V_{k,ini}$ and $V_{k,goal}$ being the initial voltage corresponding to $f_{k,ini}$ and the goal voltage corresponding to $f_{k,goal}$, respectively. Therefore, when task t_i is assigned to processor pn_k with execution frequency $f_{k,l}$, the energy overhead of voltage/frequency switching can be represented as

$$E_{\text{sw}}(t_i, pn_k, f_{k,l}) = \begin{cases} \chi_k \cdot \left| V_{k,l}^2 - V_{k,l_0}^2 \right| & V_{k,l} \neq V_{k,l_0} \\ 0 & V_{k,l} = V_{k,l_0} \end{cases}, \tag{6}$$

where V_{k,l_0} represents the voltage used by processor pn_k in its last executed task.

Considering that different tasks may be executed on different processors, the data transmission between two processors will consume energy. Assuming that the data transmission energy consumption is proportional to the transmission time, the energy consumption rate per unit time is expressed as ecr [3]. Therefore, when task t_i executes on processor pn_k, the data transmission energy consumption is given by

$$E_{\text{t}}(t_i, pn_k) = \sum_{t_j \in pred(t_i)} ecr \times c'_{j,i}. \tag{7}$$

In Eq. (7), if the task t_i and the direct predecessor task t_j are assigned to the same processor, then the value of $c'_{j,i}$ is 0; otherwise the value of $c'_{j,i}$ is $c_{j,i}$.

According to Eqs. (3), (6), and (7), when the task t_i is assigned to a processor, the sum of processor dynamic energy consumption and data transmission energy consumption can be calculated by

$$\begin{aligned} E_{\text{dt}}(t_i) = {} & E_{\text{d}}(t_i, pn_{ap(i)}, f_{ap(i),af(i)}) \\ & + E_{\text{sw}}(t_i, pn_{ap(i)}, f_{ap(i),af(i)}) \\ & + E_{\text{t}}(t_i, pn_{ap(i)}), \end{aligned} \tag{8}$$

where $pn_{ap(i)}$ and $f_{ap(i),af(i)}$ represent the assigned processor and corresponding execution frequency of task t_i, respectively.

From the above analysis, when all tasks in application A are assigned to the processor, the total dynamic energy consumption can be calculated by

$$E_{\text{dt}}(A) = \sum_{i=1}^{n} E_{\text{dt}}(t_i). \tag{9}$$

The static energy consumption of application A is denoted as $E_{\text{s}}(A)$, which can be calculated by

$$E_{\text{s}}(A) = \sum_{j=1}^{m} P_{k,\text{sta}} \times SL(A), \tag{10}$$

where $SL(A)$ is the scheduling length of application A, which is the time from the beginning of the first task execution to the completion of the last task. According to Eqs. (9) and (10), the total energy consumption of application A can be given by

$$E(A) = E_{\text{dt}}(A) + E_{\text{s}}(A). \tag{11}$$

3.4. Reliability model

The faults might occur during the execution of an application, which can be divided into transient faults and permanent faults. Since transient faults occur more commonly than permanent faults [2,8], only transient faults are considered in this study. In general, the arrival of transient faults during task execution follows a Poisson distribution [2,3,8,17]. The reliability of an event within t unit times is $e^{-\lambda t}$, where λ indicates the transient failure rate per unit time of the processor [3]. When task t_i executes on processor pn_k with the maximum frequency, the reliability of which is given by

$$R(t_i, pn_k, f_{k,\max}) = e^{-\lambda_{k,\max} \times w_{i,k}}, \tag{12}$$

where $\lambda_{k,\max}$ indicates the transient fault rate per unit time of processor pn_k execution with the maximum frequency.

According to [2,8,26,45], the transient faults rate $\lambda_{k,l}$ of the processor pn_k with execution frequency $f_{k,l}$ is given by

$$\lambda_{k,l} = \lambda_{k,\max} \times 10^{\frac{d_k \times (f_{k,\max} - f_{k,l})}{f_{k,\max} - f_{k,\min}}}, \tag{13}$$

where d_j is the sensitivity fault rate to the corresponding execution voltage and frequency scaling of the processor pn_k, which is greater than 0.

Based on Eqs. (12) and (13), when task t_i is assigned to the processor pn_k with execution frequency $f_{k,l}$, the reliability of task t_i is given by

$$\begin{aligned} R(t_i) &= R(t_i, pn_k, f_{k,l}) \\ &= e^{-\lambda_{k,\max} \times 10^{\frac{d_k \times (f_{k,\max} - f_{k,l})}{f_{k,\max} - f_{k,\min}}} \times \frac{w_{i,k} \times f_{k,\max}}{f_{k,l}}}. \end{aligned} \tag{14}$$

When all the tasks are assigned to the processor, the reliability of the application is given by

$$R(A) = \prod_{i=1}^{n} R(t_i, pn_{ap(i)}, f_{ap(i),af(i)}). \tag{15}$$

Based on Eq. (14), the obtainable maximum reliability of task t_i can be given by

$$R_{\max}(t_i) = \max_{pn_k \in PN} \{R(t_i, pn_k, f_{k,\max})\}. \tag{16}$$

If all the tasks are assigned with the obtainable maximum reliability, the reliability of application A will reach the maximum, which can be calculated by

$$R_{\max}(A) = \prod_{i=1}^{n} R_{\max}(t_i). \tag{17}$$

3.5. Problem description

Considering a parallel application A execution on a heterogeneous distributed system, the deadline and reliability requirements are $DL(A)$ and $R_{\text{req}}(A) \leq R_{\max}(A)$, respectively. The problem of this work is to determine the processor and frequency assignments for each task in A so that the total energy consumption can be minimized while the deadline constraint and the reliability requirement of the application are satisfied. We refer to this problem as the energy consumption optimization problem under deadline and reliability constraints (ECODRC), and the formal description is to minimize

$$E(A) = E_{\text{dt}}(A) + E_{\text{s}}(A) \tag{18}$$

subject to

$$SL(A) <= DL(A), \tag{19}$$

$$R(A) \geq R_{\text{req}}(A), \tag{20}$$

and

$$f_{ap(i),\min} \leq f_{ap(i),af(i)} \leq f_{ap(i),\max} \tag{21}$$

for all i: $1 \leq i \leq n$, $pn_{ap(i)} \in PN$.

4. The proposed algorithms

In this section, two algorithms are proposed to solve the ECODRC problem. The first algorithm aims to minimize the scheduling length while satisfying the reliability requirement (MSLSRR), the second algorithm mainly aims to further improve energy efficiency (IEE) with DVFS. The algorithms MSLSRR and IEE are introduced as follows.

4.1. The MSLSRR algorithm

In order to introduce the design method of the MSLSRR algorithm, we will explain three aspects: how to determine the priority of tasks, how to satisfy the reliability requirement, and how to minimize the scheduling length.

4.1.1. Priority of tasks

In order to ensure the order of the tasks during the scheduling process, it is necessary to introduce the priority of tasks. Similar to [3,17,33], the upward rank value is used to generate the task execution sequence, and the rank value is calculated as

$$Rank(t_i) = \overline{w_i} + \max_{t_j \in succ(t_i)} \{c_{i,j} + Rank(t_j)\}. \tag{22}$$

In Eq. (22), the $\overline{w_i}$ is average WCET of task t_i on each processor, which can be calculated by

$$\overline{w_i} = \left(\sum_{j=1}^{m} w_{i,j} \right) / m. \tag{23}$$

The rank values of each task in the motivational parallel application are shown in Fig. 2(b), tasks with greater rank values have higher priority. Therefore, before the application is executed, we sort the task in a non-increasing order of rank values, which can make the task execution in the correct order. In the subsequent sections of this paper, it is assumed that the tasks have been prioritized.

4.1.2. Satisfying reliability requirement

The use of the DVFS technique has a negative impact on the reliability of tasks and also prolongs execution time, for this reason, DVFS is not used in the MSLSRR algorithm. Because the reliability of the application is determined by the reliability of all the tasks, the reliability requirement of the application can be transferred into the reliability requirement of each task. The main idea of the MSLSRR algorithm is to determine a reliability requirement for each task, and then assign it to the processor with the earliest finish time while satisfying its reliability requirement.

To determine the reliability requirement of each task, we define the reliability ratio Rr as the ratio of the application's reliability requirement to the maximum reliability that can be achieved.

$$Rr = \frac{R_{\text{req}}(A)}{R_{\max}(A)}. \tag{24}$$

Obviously, $0 < Rr \leq 1$, Eq. (24) can be decomposed into the following form

$$R_{\text{req}}(A) = Rr \times R_{\max}(t_1) \times R_{\max}(t_2) \times \cdots \times R_{\max}(t_n). \tag{25}$$

Similarly, Rr can be decomposed into the following form

$$Rr = Rr^{\frac{rw_1}{S}} \times Rr^{\frac{rw_2}{S}} \times \cdots \times Rr^{\frac{rw_n}{S}}, \tag{26}$$

where $S = \sum_{i=1}^{n} rw_i$.

Based on Eqs. (25) and (26), we have

$$R_{\text{req}}(A) = \prod_{i=1}^{n} R_{\max}(t_i) Rr^{\frac{rw_i}{S}}. \tag{27}$$

Now, the reliability requirement of task t_i is given by

$$R_{\text{req}}(t_i) = R_{\max}(t_i) \times Rr^{\frac{rw_i}{S}}, \tag{28}$$

From Eq. (28), it can be seen that the reliability requirements for each task are different. Because the reliability of the task is related to execution time, we can set lower reliability requirements for tasks with long execution times. To do this, a simple idea is to make $rw_i = \overline{w_i}$ (the average WCET of task t_i on each processor). However, when assigning tasks to processors, the actual reliability of the task may be much higher than its reliability requirements because DVFS is not used. This will result in low reliability requirements for tasks assigned later, which means that the reliability requirements for tasks assigned earlier are too high. To overcome this deficiency, we introduce a compensation sequence SE as $\{se_1, se_2, \cdots, se_n\}$, where $se_i = \overline{w_i}$. Then, we sort SE in non-ascending order and make $rw_i = \overline{w_i} + se_i$. Now, that is to give a larger compensation to the early assigned tasks, while the later assigned tasks are only given a smaller compensation. This compensation method is beneficial for reducing the reliability requirements of early assigned tasks. In Eq. (28), $0 < Rr \leq 1$, $rw_i > 0$, and $S = \sum_{i=1}^{n} rw_i$, so $R_{\text{req}}(t_i) \leq R_{\max}(t_i)$. That is to say, the processor that satisfies the reliability requirement of task t_i can always be found. Therefore, the reliability requirement of the applications can always be satisfied.

When assigning tasks to processors, let $\{t_1, t_2, \cdots, t_{i-1}\}$ represent the previously assigned tasks, t_i represent the current task to be assigned, and $\{t_{i+1}, t_{i+2}, \cdots, t_n\}$ represent the tasks that have not been assigned. The reliability requirement for the current task t_i can be calculated as

$$R_{\text{req}}(t_i) = \frac{R_{\text{req}}(A)}{\prod_{a=1}^{i-1} R(t_a) \times \prod_{b=i+1}^{n} R_{\text{req}}(t_b)}. \tag{29}$$

In Eq. (29), the item $\prod_{a=1}^{i-1} R(t_a)$ represents the reliability obtained from the previously assigned $i-1$ tasks. The item $\prod_{b=i+1}^{n} R_{\text{req}}(t_b)$ represents the reliability requirement of $n-i$ tasks that have not been assigned.

4.1.3. Minimizing scheduling length

The schedule length of parallel application A is determined by the finish time of the task. When parallel application A is executed on a heterogeneous system, the earliest start time (EST) and the earliest finish time (EFT) of task t_i on processor pn_k can be given by

$$\begin{cases} EST(t_{\text{entry}}, pn_k) = 0 \\ EST(t_i, pn_k) = \max \left(\begin{array}{l} eat[k], \\ \max_{t_j \in pred(t_i)} \{EFT(t_j, pn_{ap(j)}) + c'_{j,i}\} \end{array} \right) \end{cases} \tag{30}$$

and

$$EFT(t_j, pn_{ap(j)}) = EST(t_j, pn_{ap(j)}) + w_{j,ap(j)} \tag{31}$$

In Eq. (30), $eat[k]$ represents the earliest available time of processor pn_k. When the task executed on processor pn_k is completed, the processor will become available. $c'_{j,i}$ represents data transmission time. If task t_j is not assigned to processor pn_k, then $c'_{j,i} = c_{j,i}$; otherwise, $c'_{j,i} = 0$. In order to achieve a smaller scheduling length, this study assigns the current task to the processor that can complete the task earliest while meeting its reliability requirement. Finally, the scheduling length of the application is given by

$$SL(A) = \max_{1 \leq i \leq n} EFT(t_i, pn_{ap(i)}). \tag{32}$$

It should be noted that tasks can also be assigned to processors using other parameters such as the earliest start time, reliability metrics,

or other relevant criteria. However, under common circumstances, utilizing the earliest finish time for task allocation tends to yield relatively shorter scheduling lengths. In Section 4.3, we present the results obtained by different methods in allocating the parallel application depicted in Fig. 2, where the scheduling lengths derived from applying the earliest finish time, earliest start time, and reliability metrics are respectively 80, 90, and 136. This clearly demonstrates that using the earliest finish time as a parameter leads to a smaller scheduling length.

4.1.4. Proposed MSLSRR algorithm

Based on the aforementioned, the MSLSRR algorithm is designed as shown in **Algorithm 1**.

Algorithm 1 MSLSRR

Input: $PN = \{pn_1, pn_2, ..., pn_m\}$, application A, $DL(A)$, and $R_{\text{req}}(A)$
Output: $R(A)$, $SL(A)$, and $E(A)$

1: sort the tasks to ready queue q by non-increasing order of *Rank* value
2: **while** q is not empty **do**
3: $t_i \leftarrow q.out()$
4: calculate $R_{\text{req}}(t_i)$ using Eq. (29)
5: $eft = +\infty$
6: **for** each processor $pn_k \in PN$ **do**
7: calculate $R(t_i, pn_k, f_{k,\max})$ using Eq. (12)
8: **if** $R(t_i, pn_k, f_{k,\max}) \geq R_{\text{req}}(t_i)$ **then**
9: calculate $EFT(t_i, pn_k)$ using Eq. (31)
10: **if** $eft > EFT(t_i, pn_k)$ **then**
11: $eft \leftarrow EFT(t_i, pn_k)$
12: $u \leftarrow k$
13: **end if**
14: **end if**
15: **end for**
16: **if** $eft > DL(A)$ **then**
17: return false
18: **end if**
19: assign task t_i to processor pn_u
20: **end while**
21: $SL(A) = eft$
22: calculate $R(A)$ using Eq. (15)
23: calculate $E(A)$ using Eq. (11)

The details of MSLSRR are explained as follows.

(1) Line 1 sorts the tasks into the queue according to their priority.

(2) Lines 2–20 are a loop structure that assigns tasks to the appropriate processor.

(3) For each task, MSLSRR first calculates the reliability requirement (Line 4) and then assigns it to the processor that has the earliest finish time while satisfying the reliability requirements (Lines 6–19).

4.1.5. Time complexity of MSLSRR

MSLSRR initially sorts tasks into queue q, which takes O($n$log$n$) time. When assigning a task, MSLSRR needs to traverse all processors, which can be done in O(m) time. For each processor, MSLSRR calculates the earliest finish time for the task, which requires traversing all its predecessor tasks, with a time complexity of O(n). There are n tasks that need to be assigned, hence the time complexity of MSLSRR is O($m \times n^2$), which is equal to that of the HEFT algorithm [33].

4.2. The IEE algorithm

Because the MSLSRR algorithm does not use DVFS, the actual reliability of the application may be higher than the reliability requirements. In addition, the scheduling length obtained by the MSLSRR algorithm may also be less than the deadline. At this point, the IEE algorithm attempts to adjust the processor and frequency combination for executing tasks to improve energy efficiency.

4.2.1. Minimum reliability requirement

To ensure that the application still meets the reliability requirement and deadline constraint, it is still necessary to calculate its minimum reliability requirement and available execution time when assigning a task. Assuming that after the completion of the MSLSRR algorithm, the actual reliability of the application is $R(A)$, which is higher than the reliability requirement $R_{\text{req}}(A)$, i.e. $R(A) > R_{\text{req}}(A)$. In this case, the reliability of the task can be appropriately reduced to further improve energy efficiency.

To obtain the minimum reliability requirements of the tasks, similar to Section 4.1.2, define a new reliability ratio Nr as the ratio of the application's reliability requirement to the reliability obtained by the MSLSRR algorithm.

$$Nr = \frac{R_{\text{req}}(A)}{R(A)} \tag{33}$$

In the IEE algorithm, we still use a method similar to that in Section 4.1.2, the minimum reliability requirement of task t_i is given by

$$R_{\text{minreq}}(t_i) = R(t_i) \times Nr^{\frac{rw_i}{S}}, \tag{34}$$

where $S = \sum_{i=1}^{n} rw_i$ with $rw_i = \overline{w_i} + se_i$. Unlike the MSLSRR algorithm, the compensation sequence SE in the IEE algorithm is arranged in a non-descending order.

When adjusting the processor and execution frequency of the task using the *upward* method, the minimum reliability requirement of the current task t_i is calculated as

$$R_{\text{minreq}}(t_i) = \frac{R_{\text{req}}(A)}{\prod_{a=1}^{i-1} R_{\text{new}}(t_a) \times \prod_{b=i+1}^{n} R_{\text{minreq}}(t_b)}. \tag{35}$$

In Eq. (35), the item $\prod_{a=1}^{i-1} R_{\text{new}}(t_a)$ represents the new reliability obtained by $i-1$ tasks that have been reassigned. The item $\prod_{b=i+1}^{n} R_{\text{minreq}}(t_b)$ represents the minimum reliability requirement of $n-i$ tasks that have not been reassigned.

It should be noted that the actual reliability of the task will slightly exceed the reliability requirement when using DVFS. Therefore, we still use compensation sequences in the IEE algorithm to recalculate the reliability requirement of the task. In this way, the new reliability requirement of the task will not be too high or too low, so that all tasks have the opportunity to choose processors with lower energy consumption for execution.

4.2.2. Available execution time

If the scheduling length obtained by the MSLSRR algorithm is less than the deadline, there is a slack time between consecutive tasks. The slack time can be used to reassign processor and execution frequency for tasks, thereby improving energy efficiency. Before using slack time, we first employ the *upward* (i.e., from t_{exit} to t_{entry} [36]) method to adjust the execution time of all tasks.

After tasks are assigned by MSLSRR, for any task t_i, its start time is represented as $ST(t_i, pn_{ap(i)})$, and its finish time is represented as $FT(t_i, pn_{ap(i)})$, respectively. Thus, the execution time of task t_i is given by

$$ET(t_i) = FT(t_i, pn_{ap(i)}) - ST(t_i, pn_{ap(i)}). \tag{36}$$

We adjust the finish time of task t_i to its latest finish time $LFT(t_i, pn_{ap(i)})$, which can be calculated by

$$\begin{cases} LFT(t_{\text{exit}}, pn_k) = SL_{\text{MSLSRR}}(A) \\ LFT(t_i, pn_k) = \min \begin{pmatrix} \min_{t_j \in succ(t_i)} \left\{ ST(t_j, pn_{ap(j)}) - c'_{i,j} \right\}, \\ \min_{j>i, ap(j)=k} \left\{ ST(t_j, pn_{ap(j)}) \right\} \end{pmatrix} \end{cases}, \tag{37}$$

where $SL_{\text{MSLSRR}}(A)$ represents the scheduling length obtained by the MSLSRR algorithm. Since the processor associated with the task remains unchanged, it is easy to calculate the start time of the task, which

is the finish time minus the execution time. According to Eq. (37), after adjusting the start time and finish time of tasks, the scheduling length of the application will still not exceed $SL_{\mathrm{MSLSRR}}(A)$.

The following introduces how to utilize slack time. To make good use of slack time, we first introduce the slack ratio Sr, then apply Sr to distribute slack time to each task, and finally reassign tasks to the processor while meeting time constraints and reliability requirements. The slack ratio Sr is defined as the ratio of the deadline ($DL(A)$) to the scheduling length ($SL_{\mathrm{MSLSRR}}(A)$) obtained by the MSLSRR algorithm, which is expressed as

$$Sr = \frac{DL(A)}{SL_{\mathrm{MSLSRR}}(A)}. \tag{38}$$

If the start time of all tasks are extended by Sr times, the application can still meet the deadline. The extended start time and finish time are

$$ST(t_i, pn_{ap(i)}) = Sr \times ST(t_i, pn_{ap(i)}) \tag{39}$$

and

$$FT(t_i, pn_{ap(i)}) = ST(t_i, pn_{ap(i)}) + ET(t_i), \tag{40}$$

respectively. When the IEE algorithm reassigns a task, it is assumed that all other tasks have been assigned. If task t_i is reassigned to processor pn_k, the earliest start time $EST(t_i, pn_k)$ can still be calculated by Eq. (30). Simultaneously, by replacing '$SL_{\mathrm{MSLSRR}}(A)$' with '$DL(A)$' in Eq. (37), the latest finish time $LFT(t_i, pn_k)$ can also be obtained.

When task t_i is completed before time $LFT(t_i, pn_k)$, it will not affect the execution time of subsequent tasks, which will ensure that the application's deadline is met. Therefore, the available execution time (AET) of task t_i on processor pn_k is given by

$$AET(t_i, pn_k) = LFT(t_i, pn_k) - EST(t_i, pn_k). \tag{41}$$

If $AET(t_i, pn_k) - TO(pn_k, f_{k,l_0}, f_{k,l}) \geq w_{i,k} \times \frac{f_{k,\max}}{f_{k,l}}$, then task t_i can be executed on processor pn_k with execution frequency $f_{k,l}$, and its start time and finish time are calculated as

$$ST(t_i, pn_k) = EST(t_i, pn_k) + TO(pn_k, f_{k,l_0}, f_{k,l}), \tag{42}$$

and

$$FT(t_i, pn_k) = ST(t_i, pn_k) + w_{i,k} \times \frac{f_{k,\max}}{f_{k,l}}, \tag{43}$$

respectively. Finally, the scheduling length generated by the IEE algorithm can be obtained by

$$SL'(A) = \max_{1 \leq i \leq n} FT(t_i, pn_{ap(i)}). \tag{44}$$

4.2.3. Proposed IEE algorithm

Based on the aforementioned, the IEE algorithm is proposed as shown in **Algorithm 2**.

The main ideas of the IEE algorithm are as follows.

IEE first obtains the latest finish time and start time for tasks in *upward* order, then calculates the extended start time and finish time for each task, and finally assigns tasks to processors in *downward* order while setting their respective execution frequencies. For each task t_i, IEE first calculates its minimum reliability requirement and available execution time and then assigns the task to the processor with the minimum energy consumption while satisfying the reliability requirement and meeting the deadline. The details of the main idea are explained as follows.

(1) Line 1 sorts the tasks to list q.

(2) Line 2 calculates the latest finish time and start time for tasks in *upward* order.

(3) Line 3 calculates the extended start time and finish time for each task.

(4) Lines 5–24 constitute a nested loop, in which IEE reassigns processor and execution frequency for each task.

Algorithm 2 IEE

Input: $PN = \{pn_1, pn_2, ..., pn_m\}$, application A, $DL(A)$, $R_{\mathrm{req}}(A)$, and task assignment results of the MSLSRR algorithm
Output: $R(A)$, $SL(A)$, and $E(A)$

1: sort the tasks to list q by non-increasing order of Rank value
2: from t_n to t_1, set each task's finish time to its latest finish time based on Eq. (37), and assign the corresponding start times
3: for each task t_i, calculate extended start time and finish time by Eqs. (39) and (40)
4: define an array fr to record the frequency at which processors execute their previous tasks, initializing all values within fr to the maximum frequency.
5: **for** each task t_i in q (from t_1 to t_n) **do**
6: calculate $R_{\mathrm{minreq}}(t_i)$ using Eq. (35)
7: $e = +\infty$
8: **for** each processor $pn_k \in PN$ **do**
9: **for** execution frequency $f_{k,l} \leftarrow f_{k,\min}$ to $f_{k,\max}$ **do**
10: calculate $R(t_i, pn_k, f_{k,l})$ using Eq. (14)
11: calculate $AET(t_i, pn_k)$ using Eq. (41)
12: **if** $R(t_i, pn_k, f_{k,l}) \geq R_{\mathrm{minreq}}(t_i)$ **and** $AET(t_i, pn_k) - TO(pn_k, fr[k], f_{k,l}) \geq w_{i,k} \times \frac{f_{k,\max}}{f_{k,l}}$ **then**
13: calculate $E_{\mathrm{dt}}(t_i)$ using Eq. (8)
14: **if** $e > E_{\mathrm{dt}}(t_i)$ **then**
15: $e \leftarrow E_{\mathrm{dt}}(t_i)$
16: $u \leftarrow k$ and $v \leftarrow l$
17: **end if**
18: **end if**
19: **end for**
20: **end for**
21: assign task t_i to processor pn_u with execution frequency $f_{u,v}$
22: $fr[u] \leftarrow f_{u,v}$
23: use Eqs. (42) and (43) to set the start time and finish time for task t_i, respectively.
24: **end for**
25: calculate $R(A)$ using Eq. (15)
26: calculate $E(A)$ using Eq. (11)

(5) Line 6 calculates the minimum reliability requirement of the task.

(6) Lines 8–20 traverse all processors and execution frequencies to search for the processor and frequency combination with the lowest energy consumption, which satisfies the reliability requirement and meets the deadline of the task.

(7) Line 21 assigns a task to the selected processor and sets the execution frequency.

(8) Lines 22–23 update array fr as well as the start and finish times of the tasks.

4.2.4. Time complexity of IEE

In Line 1, IEE sorts the tasks into list q, with a time complexity of O($n$log$n$). In Line 2, IEE calculates the latest finish time and start time of tasks. For each task, IEE needs to traverse all its direct successor tasks. Therefore, the time complexity of Line 2 is O(n^2). Lines 3–4 can be completed in O(n) time. IEE then needs to traverse all processors and execution frequencies when assigning a task, which can be done in O($m \times fls$) time, where fls represents the maximum number of discrete frequency levels of the processor. For each task, IEE calculates the earliest start time and latest finish time, which requires traversing all predecessor tasks and all successor tasks separately, so its time complexity is O(n). There are n tasks that need to be assigned, hence the time complexity of IEE is O($m \times n^2 \times fls$).

Table 3
Parameters for three processors.

pn_k	$P_{k,\mathrm{st}}$	$P_{k,\mathrm{in}}$	$C_{k,\mathrm{sw}}$	a_k	$\lambda_{k,\max}$	d_k
pn_1	0.01	0.03	1.2	2.8	0.0003	1.4
pn_2	0.01	0.05	1.0	2.7	0.0002	1.6
pn_3	0.01	0.07	1.1	2.6	0.0001	1.8

Table 4
The results of the MSLSRR algorithm.

t_i	$R(t_i)$	pn_k	EF	ST	FT	$E_{\mathrm{d}}(t_i)$	$E_{\mathrm{t}}(t_i)$	$E_{\mathrm{dt}}(t_i)$
t_1	0.9991004	3	1	0	9	10.53	0	10.53
t_3	0.9981018	3	1	9	28	22.23	0	22.23
t_4	0.9984013	2	1	18	26	8.4	1.8	10.2
t_2	0.9961076	1	1	27	40	15.99	3.6	19.59
t_5	0.9990005	3	1	28	38	11.7	0	11.7
t_6	0.9968051	2	1	26	42	16.8	2.8	19.6
t_9	0.9976029	2	1	56	68	12.6	5.8	18.4
t_7	0.9989006	3	1	38	49	12.87	0	12.87
t_8	0.9985011	1	1	57	62	6.15	8.4	14.55
t_{10}	0.9986010	2	1	73	80	7.35	5.6	12.95

$R(A) = 0.98127749,\ E_{\mathrm{dt}}(A) = 152.62,\ SL(A) = 80$

Table 5
The results of the IEE algorithm.

t_i	$R(t_i)$	pn_k	EF	ST	FT	$E_{\mathrm{d}}(t_i)$	$E_{\mathrm{sw}}(t_i)$	$E_{\mathrm{t}}(t_i)$	$E_{\mathrm{dt}}(t_i)$
t_1	0.9981939	3	0.9	0.07	10.07	9.06	0.03	0	9.09
t_3	0.9961909	3	0.9	10.07	31.19	19.14	0	0	19.14
t_4	0.9942861	2	0.8	19.22	29.22	5.97	0.05	1.8	7.83
t_2	0.9931557	1	0.9	28.15	42.59	13.34	0.03	3.6	16.97
t_5	0.9959233	3	0.8	31.26	43.76	8.57	0.02	0	8.6
t_6	0.9931557	1	0.9	42.59	57.04	13.34	0	2.8	16.14
t_9	0.9954964	2	0.9	58.67	72	10.7	0.02	5.8	16.52
t_7	0.9977929	3	0.9	43.83	56.06	11.08	0.02	0	11.1
t_8	0.9973620	1	0.9	57.04	62.59	5.13	0	5.4	10.53
t_{10}	0.9986010	2	1.0	73.67	80.67	7.35	0.03	5.6	12.98

$R(A) = 0.96084714,\ E_{\mathrm{dt}}(A) = 128.88,\ SL(A) = 80.67$

4.3. Case study

In the motivational example, we assume that the power dissipation and reliability parameters for all processors are known and shown in Table 3, where the maximum execution frequency $f_{k,\max}$ of each processor are normalized to 1.0. Given that the execution frequency is proportional to the voltage, and the voltages corresponding to the minimum and maximum frequencies are 1.2 V and 3.8 V, respectively. The frequency switching constants are $\beta_k = 0.2$ and $\chi_k = 0.01$, which are consistent with the processor frequency switching scenario [47]. It should be noted that the time unit and energy unit corresponding to these parameters are milliseconds and millijoules, respectively. The communication energy consumption rate ecr is 0.2. The maximum reliability of the application is 0.9860975, which can be obtained by Eq. (17). The given reliability requirement for the application is 0.96, and the deadline is 90.

Table 4 shows the results of the motivation application using the MSLSRR algorithm, where $R(t_i)$, EF, ST, and FT represent actual reliability, execution frequency, start time, and finish time, respectively. The MSLSRR algorithm has calculated the energy consumption of the processor and the data transmission energy consumption of the network. Finally, the actual reliability of the application is $R(A) = 0.98127749$, the scheduling length is 80, and the total amount of energy consumption is $E(A) = E_{\mathrm{dt}}(A) + E_{\mathrm{s}}(A) = 152.62 + (0.01 \times 3 \times 80) = 155.02$. It should be noted that when the reliability requirements of the application are not too high, the scheduling length generated by MSLSRR is the same as that of HEFT.

Table 5 shows the results of the IEE algorithm, where task t_6 is reassigned to processor pn_1, and the execution frequency of tasks t_1 to t_9 is reduced. Finally, the reliability and scheduling length constraints

Table 6
The results of the earliest start time algorithm.

t_i	$R(t_i)$	pn_k	EF	ST	FT	$E_{\mathrm{d}}(t_i)$	$E_{\mathrm{t}}(t_i)$	$E_{\mathrm{dt}}(t_i)$
t_1	0.9968051	2	1	0	16	16.8	0	16.8
t_3	0.9974034	2	1	16	29	13.65	0	13.65
t_4	0.9961076	1	1	25	38	15.99	1.8	17.79
t_2	0.9962072	2	1	29	48	19.95	0	19.95
t_5	0.9990005	3	1	27	37	11.7	2.2	13.9
t_6	0.9991004	3	1	37	46	10.53	2.8	13.33
t_9	0.9976029	2	1	61	73	12.6	7.2	19.8
t_7	0.9979022	1	1	52	59	8.61	4.6	13.21
t_8	0.9985011	1	1	67	72	6.15	6.8	12.95
t_{10}	0.9986001	2	1	83	90	7.35	5.6	12.95

$R(A) = 0.97745796,\ E_{\mathrm{dt}}(A) = 154.33,\ SL(A) = 90$

Table 7
The results of the reliability metrics algorithm.

t_i	$R(t_i)$	pn_k	EF	ST	FT	$E_{\mathrm{d}}(t_i)$	$E_{\mathrm{t}}(t_i)$	$E_{\mathrm{dt}}(t_i)$
t_1	0.9968051	2	1	0	16	16.8	0	16.8
t_3	0.9967054	1	1	28	39	13.53	2.4	15.93
t_4	0.9961076	1	1	39	52	15.99	1.8	17.79
t_2	0.9961076	1	1	52	65	15.99	3.6	19.59
t_5	0.9964065	1	1	65	77	14.76	2.2	16.96
t_6	0.9961076	1	1	77	90	15.99	2.8	18.79
t_9	0.9976029	2	1	90	102	12.6	10.4	23
t_7	0.9979022	1	1	90	97	8.61	0	8.61
t_8	0.9985011	1	1	97	102	6.15	0	6.15
t_{10}	0.9937198	1	1	115	136	25.83	2.6	28.43

$R(A) = 0.96647485,\ E_{\mathrm{dt}}(A) = 172.05,\ SL(A) = 136$

of the application are satisfied, which are $R(A) = 0.96084714$ and $SL(A) = 80.67$, respectively. The total amount of energy consumption is $E(A) = E_{\mathrm{dt}}(A) + E_{\mathrm{s}}(A) = 128.88 + (0.01 \times 3 \times 80.67) = 131.30$, which is 84.7% of the MSLSRR algorithm.

Table 6 shows the results of tasks assigned to processors with the earliest start times while satisfying their reliability requirements, where the reliability requirements for these tasks are derived from the MSLSRR algorithm.

Table 7 shows the results of assigning tasks to processors that satisfy reliability requirements, where the reliability requirement for tasks is given by $\sqrt[n]{R_{\mathrm{req}}(A)}$.

4.4. Implementation issues

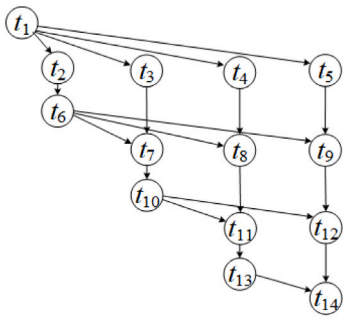
The findings from our research can be practically implemented in distributed heterogeneous system platforms, where processors are connected via the same communication bus, such as controller area networks. For optimal deployment, it is imperative that the processors in the platform support DVFS techniques, with their discrete execution frequency levels being accessible. Moreover, the transient fault rate of the processor is explicitly defined, so that our proposed algorithm can be applied and seamlessly integrated into the existing system through the following reference deployment methodology:

(1) Before the application starts, a specific processor and its corresponding frequency level should be associated with each task or application module based on the scheduling results.

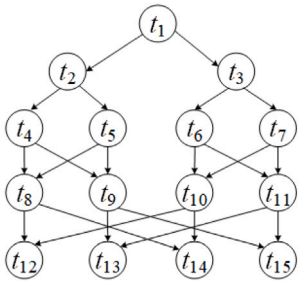
(2) A dedicated software module is needed that can communicate with the power management unit to adjust or fine-tune the voltage and frequency during the runtime.

(3) A system call will be needed during the runtime to use the software module to set the desired voltage and frequency before executing each task.

Additionally, attention is needed to the order of processor voltage and frequency adjustment. If the frequency needs to be lowered, it is important to first decrease the frequency and subsequently lower the voltage. Conversely, if the frequency needs to be raised, the voltage should be raised first, followed by the frequency adjustment.



(a) GE application



(b) FT application

Fig. 3. Examples of GE and FT.

5. Experimental performance evaluation

5.1. Experimental parameters

The proposed algorithms are intended for execution in heterogeneous computing environments, where the scheduled applications do not compete for processor cycles with other regular tasks. To simulate accurately and comprehensively the algorithms' performance and adaptability, this simulation is carefully designed to reflect the complexities of real-world computing systems as closely as possible. In particular, we utilize three shared files in the simulation: the Parallel Application file, the Processor Parameters file, and the Worst-Case Execution Time file. By leveraging these shared files, the algorithms can always access a consistent and comprehensive dataset, ensuring that the simulation results are reliable. The experimental parameters are mainly taken from [3,5,17] and listed as follows:

(1) Parallel applications: The WCET of task t_i on processor pn_k is 10 ms $\leq w_{i,k} \leq$ 100 ms, and the required data transmission time between t_i and t_j is 10 ms $\leq c_{i,j} \leq$ 100 ms.

(2) Processors: The static power, leakage power, switching capacitance, dynamic energy exponent, and the maximum frequency of processor pn_k are $P_{k,\text{sta}} = 0.01$, $0.03 \leq P_{k,\text{ind}} \leq 0.07$, $0.8 \leq C_{k,\text{swi}} \leq 1.2$, and $2.5 \leq a_k \leq 3.0$, respectively. For processor pn_k, the maximum frequency is $f_{k,\max} = 1.0$ GHz and the difference between adjacent frequency levels is 0.1 GHz. These parameter values basically simulate the characteristics of some high-performance processors, such as ARM Cortex-A9 and Intel Mobile Pentium III [5].

(3) Reliabilities: The transient fault rate of processor pn_k is $0.000001 \leq \lambda_{j,\max} \leq 0.000009$, and the sensitivity faults rate of voltage/frequency scaling is $1.0 \leq d_k \leq 3.0$.

(4) The data transmission energy consumption rate is $ecr = 0.2$ Watt.

(5) Similar to [3], a simulated heterogeneous platform with 32 processors is constructed to execute parallel applications.

The algorithm in [31] that considers Deadline constraints based on the least Resources to meet the Reliability requirement (DRR), and the DVFS technique on the Task level (DVFS-T) as described in [32], both aim to reduce energy consumption while meeting deadlines and satisfying the reliability requirements of the applications. Consequently, these two algorithms were used as benchmarks to evaluate our proposed algorithms. In the experiments, our proposed algorithm is called the MSLSRR-IEE algorithm, which first executes MSLSRR and then IEE. In addition, to ensure that the reliability requirements are not higher than the highest reliability that the application can achieve, the reliability ratio Rr is used to describe the reliability requirements (see Eq. (24), $Rr = \frac{R_{\text{req}}(A)}{R_{\max}(A)}$), and the slack ratio Sr is also used to describe the deadline constraints (see Eq. (38), $Sr = \frac{DL(A)}{SL_{\text{MSLSRR}}(A)}$).

It should be noted that, in the experiment, the reliability parameters corresponding to the ISO 26262 standard mentioned in the introduction were not directly adopted but instead replaced by Rr for the following reasons:

(1) The parameter Rr can reflect all the reliability levels that the application can achieve, and it can also prevent specifying reliability requirements that exceed the actual scope of the application. For instance, if the directly provided $R_{\text{req}}(A)$ exceeds $R_{\max}(A)$ in the experiment, it would render all algorithms unable to properly assign tasks, making the experiment meaningless. Using Rr will not result in this situation.

(2) For a given specific reliability requirement, it is straightforward to convert it into a reliability ratio Rr. Similarly, although the experiment employs the reliability ratio Rr, it is easily convertible into the corresponding explicit reliability requirement. Consequently, we display both the reliability requirements and the actual reliability achieved by all algorithms in all experiments (as shown in SubFig. (b) of Figs. 4 and 9).

(3) Although the parameter Rr is used, all reliability values in the experiment essentially fall within the range of 0.9 to 0.99, which still conforms to the reliability requirements corresponding to exposure levels in ISO 26262 mentioned in the introduction.

In addition, differing from most works that use reliability or failure probabilities (per hour) to evaluate the performance of algorithms, the simulation time in this work refers to the time required to execute the scheduled applications. The reliability achieved by each algorithm will be calculated using Eq. (15).

Gaussian elimination (GE) and Fourier transform (FT) are widely used as parallel applications for algorithm evaluation [3,17,26,36], because these two applications have the characteristics of low and high parallelism respectively. Therefore, GE and FT are used to evaluate the algorithms, and a brief introduction to these two applications is as follows.

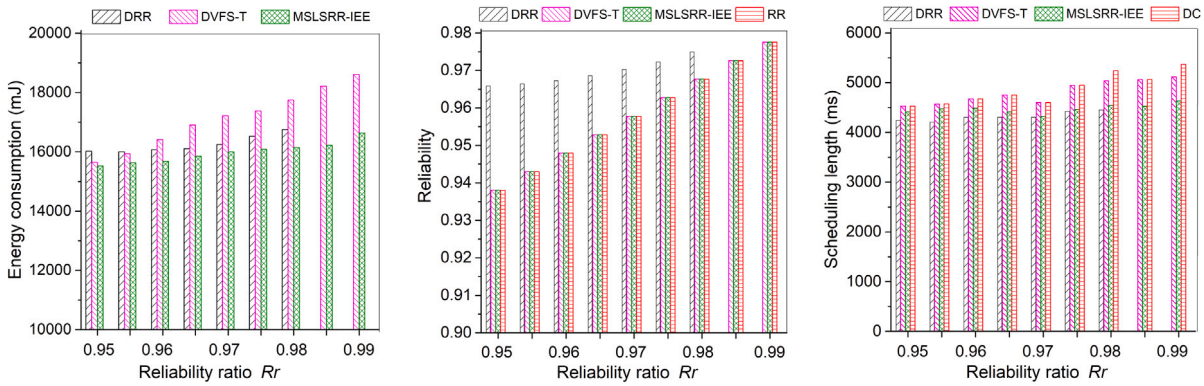
GE application: A nonnegative integer s is used to describe the scale size of the application, the total number of tasks is $n = (s^2 + s - 2)/2$. Fig. 3(a) shows an example of GE with scale size $s = 5$.

FT application: A nonnegative integer s is used to describe the scale size of the FT application, the total number of tasks is $n = (2 \times s - 1) + s \times \log_2^s$ with $s = 2^y$, where y is a nonnegative integer. Fig. 3(b) shows an example of FT with scale size $s = 4$.

5.2. Different reliability requirements

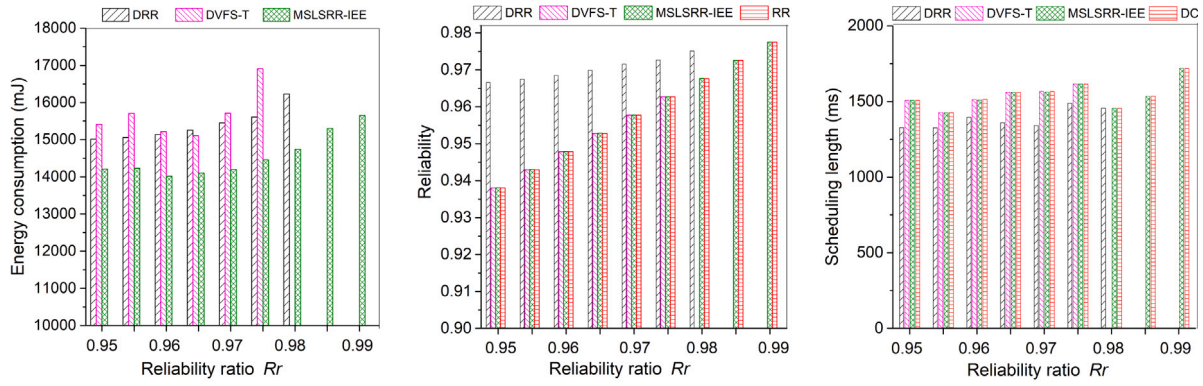
Experiment 1: This experiment uses GE applications with different reliability requirements to compare the algorithms. Scale size parameter $s = 32$ (i.e., 527 tasks), the slack ratio Sr is 1.5, and the reliability ratio Rr is changed from 0.95 to 0.99 with each increment of 0.005. The scheduling results of different algorithms are shown in Fig. 4.

As shown in Fig. 4(a), as the reliability ratio Rr increases, the energy consumption generated by all algorithms will increase. Overall, MSLSRR-IEE generates the lowest energy consumption. When Rr is less than 0.96, the energy consumption generated by DRR is greater than that of DVFS-T. However, when Rr is greater than 0.96, the energy consumption generated by DRR is less than that of DVFS-T. In this experiment, it should be noted that when Rr reaches 0.985, the DRR



(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 4. Results of the GE application for different reliability requirements. (**Experiment 1**).



(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 5. Results of the FT application for different reliability requirements. (**Experiment 2**).

algorithm will not be able to assign tasks correctly. Therefore, the schedule results at $Rr = 0.985$ and $Rr = 0.99$ using DRR are not plotted in the figure.

Fig. 4(b) shows the reliability obtained by each algorithm. In order to clarify the relationship between the reliability obtained by each algorithm and the reliability requirements, we have added the reliability requirement (RR) to the figure. When the application is scheduled with DVFS-T and MSLSRR-IEE, the obtained reliabilities are about the same and slightly higher than the reliability requirement. When the application is scheduled with DRR, the reliability of the application is obviously higher than the reliability requirement of the application.

Fig. 4(c) shows the scheduling length generated by each algorithm, in which we also added a deadline constraint (DC). When the reliability requirement of the application is increased, the changes in scheduling length generated by all algorithms are not significant. However, the scheduling length generated by DRR is the shortest, followed by MSLSRR-IEE, and the scheduling length generated by DVFS-T is the longest and very close to the deadline constraint.

Experiment 2: This experiment uses FT applications with different reliability requirements to compare the algorithms. The scale size parameter $s = 64$ (i.e., 511 tasks), the slack ratio Sr is 1.5, and the reliability ratio Rr is increased from 0.95 to 0.99 with each increment of 0.005. Fig. 5 shows the scheduling results of different algorithms.

The results of **Experiment 2** are similar to **Experiment 1**. As the reliability ratio Rr increases, the energy consumption generated by all algorithms will increase, while the energy consumption generated by MSLSRR-IEE is the lowest. It should be noted that when the reliability ratio Rr reaches 0.98, DVFS-T will not be able to assign tasks correctly, and when the reliability ratio Rr reaches 0.985, DRR will also not be able to assign tasks correctly. Based on the results of **Experiment 1** and **Experiment 2**, when the task can be properly scheduled, the energy consumption generated by MSLSRR-IEE is about 95.2% of DRR, and 92.3% of DVFS-T.

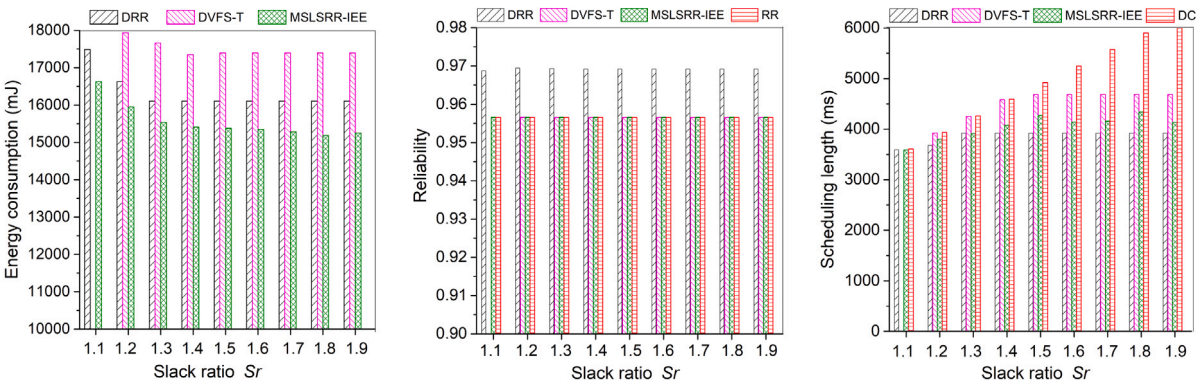
The main reasons for the above two experimental results can be explained as follows.

(1) As the reliability requirement increases, the reliability requirement for each task within the application will also increase, which will result in a decrease in the number of processors that satisfy the reliability requirement of the task. At the same time, due to the higher reliability requirement of tasks, it is more difficult to reduce the processor execution frequency. Therefore, the energy consumption generated by each algorithm will increase.

(2) DRR assumes that tasks that are not assigned are all executed with the same reliability requirement $R_{\text{req}}(t_i) = \sqrt[n]{R_{\text{req}}(A)}$. Therefore, when the reliability requirements of an application are relatively high, the reliability of some tasks cannot achieve $R_{\text{req}}(t_i)$, which will result in tasks not being assigned correctly.

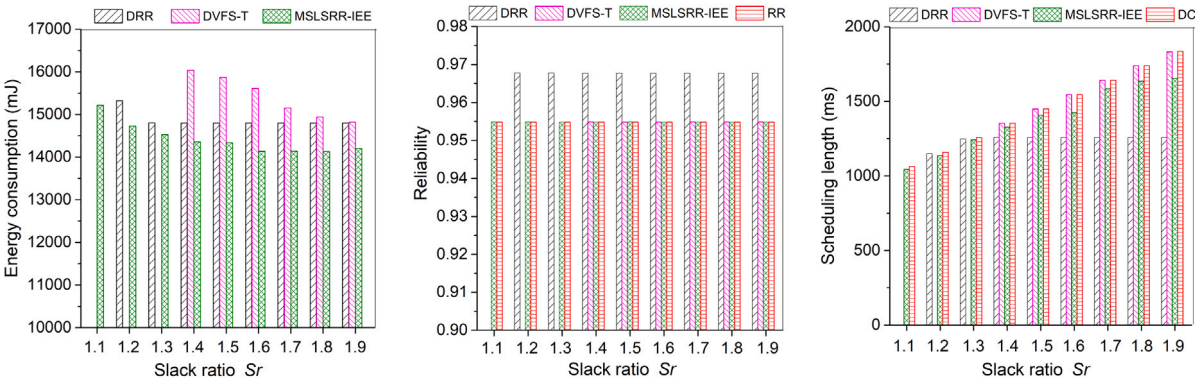
(3) DVFS-T assumes that tasks that are not assigned are executed with maximum reliability $R_{\max}(t_i)$ during initial allocation, which will result in high-reliability requirements for later assigned tasks. Therefore, for the later assigned tasks, there are too few processors available for assignment, which leads to a significant increase in scheduling length.

(4) MSLSRR-IEE uses the average execution time of tasks on each processor as a reference and designs compensation sequences to calculate the reliability requirement of tasks. Therefore, the reliability requirements of each task are neither too high nor too low, resulting in a relatively short initial scheduling length. In addition, due to the more balanced reliability requirement of tasks in MSLSRR-IEE compared



(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 6. Results of the GE application for different deadline constraints. (**Experiment 3**).



(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 7. Results of the FT application for different deadline constraints. (**Experiment 4**).

to tasks in DVFS-T, when using DVFS to improve energy efficiency, MSLSRR-IEE has a better effect than DVFS-T.

In summary, DRR may make the task unable to satisfy the reliability requirement, and DVFS-T may make the application unable to meet the deadline constraint.

5.3. Different deadline constraints

Experiment 3: This experiment uses the GE application with different deadline constraints to compare the algorithms. The scale size parameter s and the reliability ratio Rr are fixed at 32 (i.e., 527 tasks) and 0.97, respectively. The slack ratio Sr increases from 1.1 to 1.9 with increments of 0.1. Fig. 6 shows the scheduling results of different algorithms.

As shown in Fig. 6(a), when the slack ratio Sr is less than 1.5, the energy consumption generated by all algorithms tends to decrease as Sr increases. However, after Sr exceeds 1.5, there is no significant change in the energy consumption generated by all algorithms. In all cases, the energy consumption generated by MSLSRR-IEE is distinctly lower than that of the other two algorithms.

As shown in Fig. 6(b), when $Sr > 1.1$, all algorithms can satisfy the reliability requirement of the application, among which DRR achieves the highest reliability, while DVFS-T and MSLSRR-IEE generate slightly higher reliability than the reliability requirement.

As shown in Fig. 6(c), the scheduling length generated by DRR is the shortest, followed by MSLSRR-IEE, and the scheduling length generated by DVFS-T is the longest. When the slack ratio Sr reaches 1.5, there will be no significant change in the scheduling length generated by all algorithms as Sr increases.

The reasons for the experimental results are as follows. When Sr increases, DRR does not reduce the processor's execution frequency, while DVFS-T and MSLSRR-IEE are constrained by reliability constraints, thereby preventing them from arbitrarily reducing the execution frequency without limitations. Therefore, when Sr is less than 1.5, the energy consumption generated by these algorithms tends to decrease, while when Sr is greater than 1.5, the energy consumption generated by them remains almost unchanged.

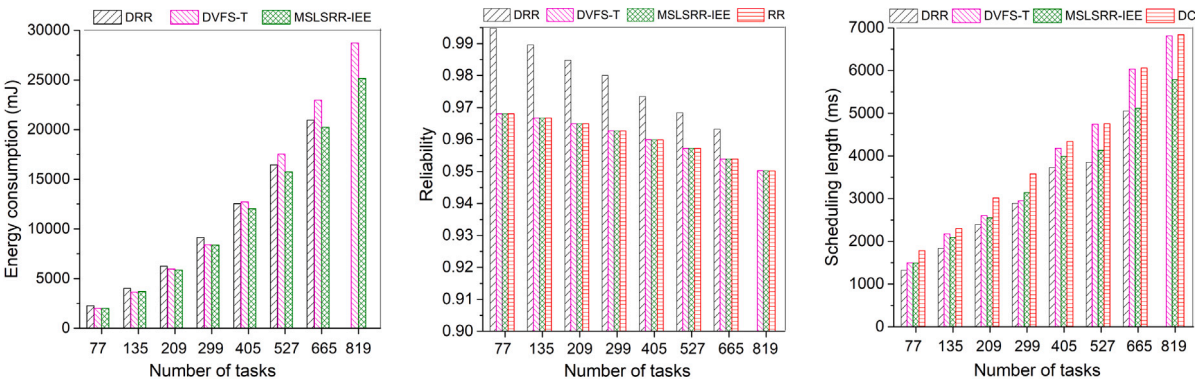
Experiment 4: This experiment uses the FT application with different deadline constraints to compare the algorithms. The scale size parameter s is fixed to 64 (i.e., 511 tasks) and other parameters are the same as in **Experiment 3**. Fig. 7 shows the scheduling results of different algorithms.

As shown in Fig. 7(a), as the slack ratio Sr increases, the energy consumption generated by all algorithms will decrease. However, when Sr reaches 1.3, the energy consumption generated by DRR no longer significantly decreases. Among the three algorithms, MSLSRR-IEE generates the lowest energy consumption. It should be noted that when Sr is less than or equal to 1.3, DVFS-T will not be able to assign tasks correctly.

As shown in Figs. 7(b) and 7(c), the reliability and scheduling length obtained by each algorithm are similar to those obtained in **Experiment 3**.

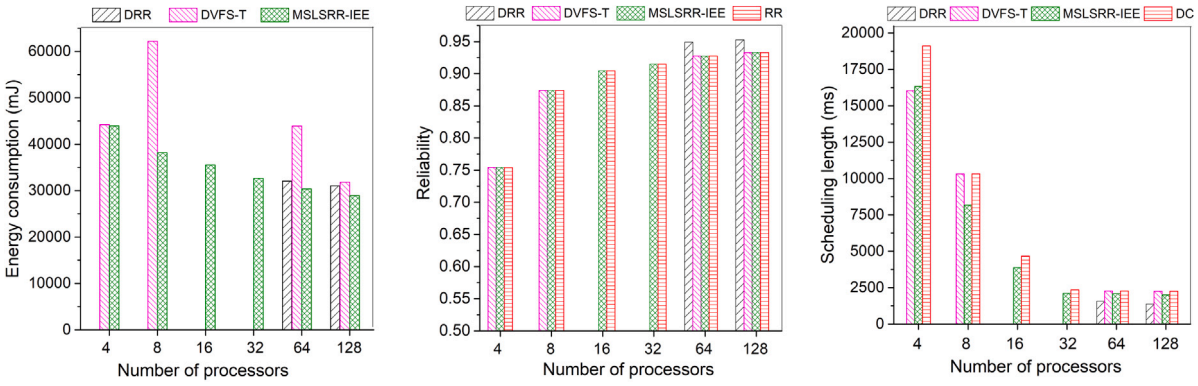
The reason for the experimental results is that the parallelism of the FT application is relatively high (compared to the GE application), and the initial scheduling length generated by MSLSRR-IEE is relatively short. Sr needs to exceed 1.3 for DVFS-T to schedule tasks correctly.

It is noteworthy that in **Experiments 3** and **4**, the parameter slack ratio Sr essentially characterizes the system workload status, where a larger Sr value denotes a lower level of system workload, while a smaller Sr indicates a correspondingly higher system workload.



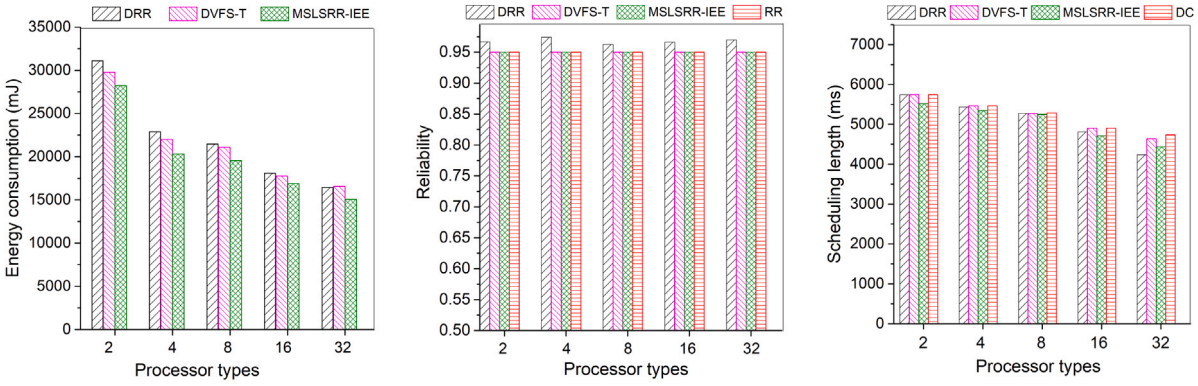
(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 8. Results of the GE applications with different number of tasks. (**Experiment 5**).



(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 9. Results of the FT application on different platforms. (**Experiment 6**).



(a) The energy consumptions (b) The reliability values (c) The scheduling lengths

Fig. 10. Results of the GE application on different platforms. (**Experiment 7**).

5.4. Different number of tasks

Experiment 5: This experiment uses the GE application with a different number of tasks to compare the algorithms. The slack ratio Sr and the reliability ratio Rr are fixed at 1.5 and 0.97, respectively. Scale size parameter s increases from 12 to 40 with increments of 4 (i.e., the number of tasks is 77, 135, 209, 299, 405, 527, 665, and 819 respectively).

Fig. 8(a) shows the energy consumption of the different algorithms. As the total number of tasks increases, the energy consumption of all three algorithms also increases. Overall, MSLSRR-IEE generates the least energy consumption.

Fig. 8(b) shows the reliability of the different algorithms. The reliability obtained by DVFS-T and MSLSRR-IEE is almost the same, which is slightly higher than the reliability requirement. However, when the number of tasks is less than or equal to 665, the reliability obtained by DRR is much higher than the reliability requirement.

Fig. 8(c) shows the scheduling lengths of the different algorithms. As the number of tasks in the application increases, the scheduling length generated by all algorithms will increase. However, when the number of tasks reaches 405, the scheduling length generated by DVFS-T is closer to the deadline constraint and significantly greater than the scheduling length generated by DRR and MSLSRR-IEE.

5.5. Different number of processors

Experiment 6: This experiment uses the FT application to compare the algorithms on the different number of processor platforms. The scale size parameter s, slack ratio Sr, and the reliability ratio Rr are fixed to 7 (1151 tasks), 2.0, and 0.95, respectively. When the number of processors is 4, 8, 16, 32, 64, and 128, the scheduling results of the three algorithms are shown in Fig. 9.

Fig. 9(a) shows the energy consumption generated by each algorithm. As the number of processors increases, the energy consumption generated by MSLSRR-IEE gradually decreases. However, when the number of processors is less than or equal to 32, DRR cannot assign tasks correctly, and when the number of processors is 16 and 32, DVFS-T also cannot assign tasks correctly. The reason for this result is that when the number of processors is small (such as less than or equal to 32), some tasks in DRR cannot meet the reliability requirement. When the initial scheduling length generated by MSLSRR-IEE is relatively short (such as with 16 and 32 processors), DVFS-T cannot meet the deadline constraint.

Fig. 9(b) shows the reliability obtained by each algorithm. As the number of processors increases, the reliability requirement also increases, indicating that the highest reliability that an application can achieve also increases. The reliability generated by DVFS-T and MSLSRR-IEE is very close to the reliability requirement, while the reliability generated by DRR is significantly higher than the reliability requirement.

Fig. 9(c) shows the scheduling length generated by each algorithm. As the number of processors increases, the scheduling length generated by each algorithm will decrease. However, when the number of processors exceeds 64, there will be no significant change in the scheduling length generated by each algorithm.

It should be noted that if we reduce the slack ratio Sr or increase the reliability ratio Rr in **Experiment 6**, there will be more cases where DVFS-T or DRR cannot assign tasks correctly.

5.6. Different number of processor types

Experiment 7: This experiment evaluates the impact of varying processor types on algorithms. GE applications with a scale size of s = 32 (i.e., 527 tasks) are used. The slack ratio Sr is 1.5, and the reliability requirement is 0.95. Assume that there are tp types of processors, with each type having m/tp processors, where m is the total number of processors in the system. In the experiment, m is set to 32. When tp is respectively 1, 2, 4, 8, 16, and 32, the scheduling results of different algorithms are shown in Fig. 10.

From Fig. 10(a), it can be seen that the energy consumption generated by all algorithms significantly decreases as the type of processor increases. This indicates that tasks have a greater opportunity to be executed on processors with lower energy consumption when the type of processor is incremented. Among all algorithms, MSLSRR-IEE generates the lowest energy consumption.

From Fig. 10(b), it can be seen that the reliability generated by all algorithms satisfies the reliability requirements.

From Fig. 10(c), it can be observed that the scheduling lengths produced by all algorithms will decrease as the type of processor increases.

The results of the above seven experiments can be summarized as follows:

(1) Under different application scenarios, MSLSRR-IEE always generates the least energy consumption among all the algorithms.

(2) When the DVFS technique is used to reduce the execution frequency of some tasks, the energy consumption may be increased. However, MSLSRR-IEE can effectively apply the DVFS technique to reduce energy consumption.

(3) Compared to DVFS-T, MSLSRR-IEE can meet relatively tight deadline constraints. Compared to DRR, MSLSRR-IEE can satisfy relatively higher reliability requirements.

5.7. Discussion on slack time

The slack time can be divided into static slack time and dynamic slack time. This paper focuses on the system design phase, primarily addressing how to efficiently utilize static slack time. Dynamic slack time is determined during task execution and is typically generated when tasks complete earlier than expected. In the parallel application studied in this paper, the task will have dynamic slack time when its direct predecessor tasks are completed earlier than expected. If the reliability constraint of the task is not considered, the dynamic slack time can be used to reduce the processor's execution frequency and thus reduce energy consumption. However, in the IEE algorithm, we have calculated the reliability requirement for each task and set the execution frequency accordingly. From the scheduling results of the IEE algorithm, it can be seen that the final reliability of the application is only slightly higher than its reliability requirement. Consequently, using dynamic slack time to further reduce the execution frequency of tasks will lead to their reliability being lower than that generated by the IEE algorithm, thereby failing to satisfy the reliability requirements of the application.

If we want to further use dynamic slack time, the task currently utilizing such dynamic slack time needs to be assigned to other processors, thus reducing energy consumption further while satisfying their reliability requirements. However, this course of action generates an additional challenge. Since each task also needs to obtain data from its predecessor task, after reassigning the task to a new processor, it also needs to transfer data from the processor where the predecessor tasks are located to the newly allocated processor, which will delay the start time of the task and result in energy consumption when retransmitting data. Therefore, this is an extremely intricate process that we will further investigate in the future.

6. Conclusions

Low-power design is crucial for many systems. This paper presents two algorithms to minimize energy consumption while satisfying the reliability requirement and deadline constraint of the applications. Experimental results demonstrate that the proposed algorithms not only consume less energy than the state-of-the-art algorithms but also adapt to relatively higher reliability requirements and relatively tighter deadline constraints. Since our proposed algorithms do not consider task replication, it may be necessary to apply task replication for some systems with particularly high reliability requirements. Therefore, in future work, we will study how to minimize energy consumption while ensuring that the application's reliability requirement and deadline are met under task replication.

CRediT authorship contribution statement

Hongzhi Xu: Writing – original draft, Project administration, Investigation. **Binlian Zhang:** Data curation. **Chen Pan:** Writing – review & editing. **Keqin Li:** Writing – review & editing, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The authors express gratitude to the anonymous reviewers for their constructive comments, which have helped improve the manuscript. The research was partially funded by the National Natural Science Foundation of China (Grant Nos. 62062036)

References

[1] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694, http://dx.doi.org/10.1109/TPDS.2013.57.

[2] D. Zhu, Reliability-aware dynamic energy management in dependable embedded real-time systems, ACM Trans. Embedd. Comput. Syst. (TECS) 10 (2) (2010) 26.

[3] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, K. Li, Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems, IEEE Trans. Ind. Inform. 13 (4) (2017) 1629–1640.

[4] K. Li, Power and performance management for parallel computations in clouds and data centers, J. Comput. System Sci. 82 (2) (2016) 174–190.

[5] G. Xie, G. Zeng, X. Xiao, R. Li, K. Li, Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Parallel Distrib. Syst. 28 (12) (2017) 3426–3442.

[6] H. Xu, R. Li, L. Zeng, K. Li, C. Pan, Energy-efficient scheduling with reliability guarantee in embedded real-time systems, Sustain. Comput.: Inform. Syst. 18 (2018) 137–148.

[7] Y. Sharma, S. Moulik, CETAS: A cluster based energy and temperature efficient real-time scheduler for heterogeneous platforms, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, SAC '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 501–509, http://dx.doi.org/10.1145/3477314.3507079.

[8] D. Zhu, H. Aydin, Reliability-aware energy management for periodic real-time tasks, Comput. IEEE Trans. on 58 (10) (2009) 1382–1397.

[9] M.A. Haque, H. Aydin, D. Zhu, On reliability management of energy-aware real-time systems through task replication, IEEE Trans. Parallel Distrib. Syst. 28 (3) (2017) 813–825, http://dx.doi.org/10.1109/TPDS.2016.2600595.

[10] M. Salehi, A. Ejlali, B.M. Al-Hashimi, Two-phase low-energy N-modular redundancy for hard real-time multi-core systems, IEEE Trans. Parallel Distrib. Syst. 27 (5) (2016) 1497–1510, http://dx.doi.org/10.1109/TPDS.2015.2444402.

[11] Z. Wu, L. Han, J. Liu, Y. Robert, F. Vivien, Energy-aware mapping and scheduling strategies for real-time workflows under reliability constraints, J. Parallel Distrib. Comput. 176 (2023) 1–16.

[12] M. Cui, A. Kritikakou, L. Mo, E. Casseau, Fault-tolerant mapping of real-time parallel applications under multiple DVFS schemes, in: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium, RTAS, IEEE, 2021, pp. 387–399.

[13] K. Huang, X. Jiang, X. Zhang, R. Yan, K. Wang, D. Xiong, X. Yan, Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems, IEEE Access 6 (2018) 57614–57630.

[14] N. Kumar, J. Mayank, A. Mondal, Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system, IEEE Trans. Parallel Distrib. Syst. 31 (4) (2019) 871–885.

[15] H. Xu, R. Li, C. Pan, K. Li, Minimizing energy consumption with reliability goal on heterogeneous embedded systems, J. Parallel Distrib. Comput. 127 (2019) 44–57.

[16] L. Ye, Y. Xia, S. Tao, C. Yan, R. Gao, Y. Zhan, Reliability-aware and energy-efficient workflow scheduling in iaas clouds, IEEE Trans. Autom. Sci. Eng. 20 (3) (2023) 2156–2169, http://dx.doi.org/10.1109/TASE.2022.3195958.

[17] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, K. Li, Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Sustain. Comput. 3 (3) (2018) 167–181.

[18] Y. Han, J. Liu, W. Hu, Y. Gan, High-reliability and energy-saving DAG scheduling in heterogeneous multi-core systems based on task replication, in: 2021 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2021, pp. 2012–2017.

[19] G. Xie, Y. Wei, Y. Le, R. Li, Redundancy minimization and cost reduction for workflows with reliability requirements in cloud-based services, IEEE Trans. Cloud Comput. 10 (01) (2022) 633–647.

[20] L. Wang, J. Zhu, S. Tian, T. Pei, H. Liu, Y. Li, Fast finding optimal redundancy to satisfy reliability requirement for safety-critical parallel applications on heterogeneous distributed automotive systems, in: 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, ISPA/BDCloud/SocialCom/SustainCom, IEEE, 2019, pp. 372–379.

[21] J. Liu, Z. Zhu, C. Deng, A novel and adaptive transient fault-tolerant algorithm considering timing constraint on heterogeneous systems, IEEE Access 8 (2020) 103047–103061.

[22] D. Mao, W. Hu, Y. Gan, J. Liu, H. Gu, A fault-tolerant scheduling algorithm based on local maximum reliability replication strategy in real-time heterogeneous systems, in: 2022 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2022, pp. 3192–3197.

[23] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, Inform. Sci. 379 (2017) 241–256.

[24] J. Huang, R. Li, X. Jiao, Y. Jiang, W. Chang, Dynamic dag scheduling on multiprocessor systems: reliability, energy, and makespan, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3336–3347.

[25] J. Chen, P. Han, Y. Zhang, T. You, P. Zheng, Scheduling energy consumption-constrained workflows in heterogeneous multi-processor embedded systems, J. Syst. Archit. 142 (2023) 102938, http://dx.doi.org/10.1016/j.sysarc.2023.102938.

[26] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inform. Sci. 319 (2015) 113–131.

[27] J. Peng, K. Li, J. Chen, K. Li, Reliability/performance-aware scheduling for parallel applications with energy constraints on heterogeneous computing systems, IEEE Trans. Sustain. Comput. 7 (3) (2022) 681–695.

[28] Y. Tang, Y. Yuan, Y. Liu, Cost-aware reliability task scheduling of automotive cyber-physical systems, Microprocess. Microsyst. 87 (2021) 103507.

[29] L. Han, Y. Gao, J. Liu, Y. Robert, F. Vivien, Energy-aware strategies for reliability-oriented real-time task allocation on heterogeneous platforms, in: ICPP '20: 49th International Conference on Parallel Processing, 2020, pp. 1–11, http://dx.doi.org/10.1145/3404397.3404419.

[30] N. Kumar, J. Mayank, A. Mondal, Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system, IEEE Trans. Parallel Distrib. Syst. 31 (4) (2020) 871–885, http://dx.doi.org/10.1109/TPDS.2019.2950251.

[31] L. Zhao, Y. Ren, K. Sakurai, Reliable workflow scheduling with less resource redundancy, Parallel Comput. 39 (10) (2013) 567–585.

[32] B. Hu, Z. Cao, M. Zhou, Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems, IEEE Trans. Serv. Comput. 15 (5) (2022) 2766–2779, http://dx.doi.org/10.1109/TSC.2021.3054754.

[33] H. Topcuoglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.

[34] Z. Tang, L. Qi, Z. Cheng, K. Li, S.U. Khan, K. Li, An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment, J. Grid Comput. 14 (1) (2016) 55–74.

[35] K. Li, Energy-efficient task scheduling on multiple heterogeneous computers: Algorithms, analysis, and performance evaluation, IEEE Trans. Sustain. Comput. 1 (1) (2016) 7–19.

[36] G. Xie, J. Jiang, Y. Liu, R. Li, K. Li, Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems, IEEE Trans. Ind. Inform. 13 (3) (2017) 1068–1078.

[37] J. Jiang, Y. Lin, G. Xie, L. Fu, J. Yang, Time and energy optimization algorithms for the static scheduling of multiple workflows in heterogeneous computing system, J. Grid Comput. 15 (4) (2017) 435–456.

[38] Y. Sharma, S. Moulik, FATS-2TC: A fault tolerant real-time scheduler for energy and temperature aware heterogeneous platforms with two types of cores, Microprocess. Microsyst. 96 (2023) 104744, http://dx.doi.org/10.1016/j.micpro.2022.104744.

[39] Y. Sharma, S. Moulik, RT-SEAT: A hybrid approach based real-time scheduler for energy and temperature efficient heterogeneous multicore platforms, Results Eng. 16 (2022) 100708, http://dx.doi.org/10.1016/j.rineng.2022.100708.

[40] D. Senapati, K. Rajesh, C. Karfa, A. Sarkar, TMDS: Temperature-aware makespan minimizing DAG scheduler for heterogeneous distributed systems, ACM Trans. Des. Autom. Electron. Syst. 28 (6) (2023) http://dx.doi.org/10.1145/3616869.

[41] L. Niu, J. Musselwhite, W. Li, Work-in-progress: Enhanced energy-aware standby-sparing techniques for fixed-priority hard real-time systems, in: 2018 IEEE Real-Time Systems Symposium, RTSS, 2018, pp. 165–168, http://dx.doi.org/10.1109/RTSS.2018.00031.

[42] Y. wen Zhang, Energy-aware mixed partitioning scheduling in standby-sparing systems, Comput. Stand. Interfaces 61 (2019) 129–136, http://dx.doi.org/10.1016/j.csi.2018.06.004.

[43] S. Safari, S. Hessabi, G. Ershadi, LESS-MICS: A low energy standby-sparing scheme for mixed-criticality systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (12) (2020) 4601–4610, http://dx.doi.org/10.1109/TCAD.2020.2977063.

[44] Y.-W. Zhang, Energy-aware fixed priority scheduling with shared resources in standby-sparing systems, Microprocess. Microsyst. 87 (2021) 104362.

[45] M. Lin, Y. Pan, L.T. Yang, M. Guo, N. Zheng, Scheduling co-design for reliability and energy in cyber-physical systems, IEEE Trans. Emerg. Top. Comput. 1 (2) (2013) 353–365.

[46] S. Moulik, A. Sarkar, H.K. Kapoor, Dpfair scheduling with slowdown and suspension, in: 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems, VLSID, 2018, pp. 43–48, http://dx.doi.org/10.1109/VLSID.2018.35.

[47] T.D. Burd, R.W. Brodersen, Design issues for dynamic voltage scaling, in: Proceedings of the 2000 International Symposium on Low Power Electronics and Design, 2000, pp. 9–14.

Hongzhi Xu received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2018. He is a professor at Jishou University, Zhangjiajie, China. His research interests include heterogeneous computing systems and energy-efficient computing.

Binlian Zhang received the M.S. degree in computer science and engineering from Hunan Normal University, Changsha, China, in 2007. She is an associate professor at Jishou University, Zhangjiajie, China. Her research interests include heterogeneous computing systems and energy-efficient computing.

Chen Pan (S'13-M'20) received M.S. degree in Electrical Engineering from Oklahoma State University in 2017 and the Ph.D. degree in Electrical and Computer Engineering from University of Pittsburgh in 2019. He is currently an assistant professor with the Department of Electrical & Computer Engineering at The University of Texas at San Antonio (UTSA). His current research interests include sustainable and intelligent IoT systems, intelligent low-power sparse sensing, tiny machine learning, transient computing and communication, and emerging non-volatile memories. We appreciate your assistance in ensuring the accuracy of our publication.

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 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact 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, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is a fellow of the Asia-Pacific Artificial Intelligence Association (AAIA). He is also a member of Academia Europaea (Academician of the Europe).