Journal of Parallel and Distributed Computing 191 (2024) 104915

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

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

Energy-efficient triple modular redundancy scheduling on heterogeneous multi-core real-time 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, 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:
Energy consumption
Heterogeneous multi-core platform
Real-time system
Reliability
Triple modular redundancy

ABSTRACT

Triple modular redundancy (TMR) fault tolerance mechanism can provide almost perfect fault-masking, which has the great potential to enhance the reliability of real-time systems. However, multiple copies of a task are executed concurrently, which will lead to a sharp increase in system energy consumption. In this work, the problem of parallel applications using TMR on heterogeneous multi-core platforms to minimize energy consumption is studied. First, the heterogeneous earliest finish time algorithm is improved, and then according to the given application's deadline constraints and reliability requirements, an algorithm to extend the execution time of the copies is designed. Secondly, based on the properties of TMR, an algorithm for minimizing the execution overhead of the third copy (MEOTC) is designed. Finally, considering the actual situation of task execution, an online energy management (OEM) method is proposed. The proposed algorithms were compared with the state-of-the-art AFTSA algorithm, and the results show significant differences in energy consumption. Specifically, for light fault detection, the energy consumption of the MEOTC and OEM algorithms was found to be 80% and 72% respectively, compared with AFTSA. For heavy fault detection, the energy consumption of MEOTC and OEM was measured at 61% and 55% respectively, compared with AFTSA.

1. Introduction

1.1. Background

Nowadays, integrated circuit technology and computer technology are developing rapidly, more than a hundred processor elements can be integrated into a single chip, which is called multi-core processors [30]. Multi-core platforms often integrate different types of cores to execute applications, which have heterogeneous characteristics. For example, OMAP1/OMAP2 integrates CPU and DSP on the same chip, the Tegra integrates CPU and GPU on the same chip [4]. Because the multi-core platform has high performance and low energy consumption it has recently received much attention [13][18][19][20][21][23][27][28][29]. In recent years, safety-critical embedded real-time systems also used multi-core platforms to execute parallel applications [2].

Due to the temporary failure of the components or by external interference such as cosmic ray radiations, electrical power drops, and electrostatic discharge, transient faults may occur at run-time, which will reduce the reliability of the system [10][22][40]. Reliability is an extremely important non-functional target in the safety-critical embedded real-time systems. For example, in ISO 26262, the corresponding reliability requirements for exposure level-E2 (low probability) and level-E3 (medium probability) are 0.99 and 0.9, respectively [35]. The reliability requirements of applications must be guaranteed, the faults should be handled when the system is in run-time, otherwise, it may lead to disastrous consequences [12].

N-Modular Redundancy (NMR) is one of the most popular techniques for enhancing the reliability of applications [2][30]. NMR extensively utilizes the M-out-of-N voter as a decision-making component. Within an array of inputs from N voters, the system can be considered error-free only when at least M of these inputs are equal to each other. In general, $M \geq \lfloor N/2+1 \rfloor$ is required, that is, at least $M \geq \lfloor N/2+1 \rfloor$ voter inputs should be equal. Because NMR uses the comparison of the execution results for fault detection and masking, it does not require any other specific fault detection mechanism. Since it is unlikely that

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

https://doi.org/10.1016/j.jpdc.2024.104915
Received 8 December 2022; Received in revised form 3 February 2024; Accepted 5 May 2024
Available online 13 May 2024

all modules in NMR will fail at the same time and generate the same error results, the NMR fault tolerance mechanism can provide almost perfect fault-masking [26]. Triple Modular Redundancy (TMR) is a special NMR, which uses the results of three copies of the application (or task) for voting comparison.

Existing research has demonstrated that a Dual Modular Redundant (DMR) system exhibits lower reliability compared to a simplex system. Both 4MR and 5MR configurations consistently demonstrate lower reliability than TMR. Furthermore, 6MR shows only slightly higher reliability than TMR [30]. Therefore, TMR has the great potential to enhance the reliability of the real-time systems [16][26]. When employing the TMR or NMR mechanism, multiple copies of a task are executed on different cores, and their execution results are subjected to a voting process. Consequently, the system must access the memory or cache corresponding to multiple cores. Excessive communication overhead can significantly delay voting time. Therefore, multi-core systems with robust parallelism and low communication overhead are suitable for executing multiple task copies.

1.2. Motivation

Many safety-critical systems necessitate the use of TMR technique due to their high reliability requirements. TMR involves executing multiple copies of a task simultaneously, which can result in increased system energy consumption and scheduling length of the application. An increase in the scheduling length of an application implies a decrease in system performance and may also lead to missed deadlines. Therefore, an effective task-scheduling scheme is essential in such systems. On the one hand, the reliability requirements and deadlines of the application must be met, and on the other hand, system energy consumption should be minimized. The Dynamic Voltage Frequency Scaling (DVFS) which reduces the execution voltage and frequency of the task and prolongs the execution time, can be used to reduce the energy consumption of the systems [9][11][31][36][37]. DVFS can also be applied to TMR fault tolerance. In [26] and [16], the authors have studied the energy-efficient TMR problem for a homogeneous multi-core system with DVFS. However, to the best of our knowledge, the problem of minimizing the energy consumption of parallel applications on heterogeneous multi-core platforms with the TMR technique is rarely reported. Therefore, this work mainly studies the energy-efficiency scheduling scheme for parallel applications on heterogeneous multi-core real-time systems with the TMR technique.

1.3. Main contributions

The main contributions of this paper are as follows.

(1) We have transformed the reliability requirement of the application into the reliability requirement of the task and then calculated the reliability requirement for each copy under the TMR mechanism.

(2) We have proposed an enhanced version of the Heterogeneous Earliest Finish Time (HEFT) algorithm called Improved HEFT (IHEFT). IHEFT assigns three copies of the task to different cores, which minimizes the scheduling length while meeting the reliability requirement of the application.

(3) We have proposed an algorithm, known as Extending Execution Time of the copies (EET). The purpose of this algorithm is to reduce the system energy consumption while meeting the deadline and satisfying the reliability requirements of the application.

(4) Based on the task assignment information from IHEFT, we have introduced an algorithm, known as Minimizing the Execution Overhead of the Third Copy (MEOTC) using TMR properties. The purpose of this algorithm is to minimize the execution overhead of the third copy, resulting in energy savings.

(5) Because the execution results of two copies of a task are voted on immediately after their execution, the task will be completed in advance. Therefore, we have proposed an Online Energy Management scheme (OEM). When a task is completed, the first two copies of subsequent tasks are executed immediately, and the third copy is still executed according to the offline scheduling.

2. Related work

2.1. Energy-efficient technique

From the perspective of system design, parallel applications are composed of tasks with precedence-constrainted, which are usually modeled as Directed Acyclic Graph (DAG) [1][3][8][9][31][34]. At present, for DAG-based parallel applications execution on heterogeneous platforms with deadline constraints, there are some energy-efficient scheduling algorithms with DVFS technique [8][9][11][31][36][37]. However, these algorithms do not consider the reliability requirements of the applications.

2.2. Reliability aware without task replication

To reduce the energy consumption of the system while satisfying the reliability requirements of the applications, Xie et al. [35] designed a resource minimization algorithm for parallel applications on heterogeneous embedded systems, which first transfers the reliability requirement of the application to that of each task and then assigns each task to the processor with the minimum resource consumption. Zhang et al. [43] studied the reliability maximization problem under energy constraints and proposed the RMEC algorithm to maximize the reliability of the system. Zhang et al. [44] proposed a bi-objective genetic scheduling algorithm to achieve high system reliability and low energy consumption for workflow on heterogeneous systems. Xu et al. [41] introduced two methods to decompose the reliability requirement of the application to each task for non-DVFS and DVFS respectively and designed two energy-efficient algorithms to satisfy the reliability requirement. Huang et al. [7] proposed a method of optimizing energy allocation with the reliability constraint. The above studies do not consider task replication, which may be difficult to satisfy the high-reliability requirement of applications.

2.3. Fault-tolerant with task replication or recovery

The application's reliability can be improved by applying the task replication technique. Haque et al. [6] investigated the techniques based on task replication to minimize energy consumption for a set of periodic real-time tasks executing on a multi-core system. Kumar et al. [10] introduced an active replication-based framework to minimize the energy consumption for a set of periodic real-time tasks with reliability requirements and timing constraints on a heterogeneous system. Wang et al. [33] proposed the task replication scheduling algorithm to maximize system reliability. Xie et al. [38] presented a fault-tolerant scheduling algorithm EFSRG to reduce energy consumption while satisfying the reliability requirement of the application based on an active replication. In [39] the authors proposed a redundancy minimization algorithm to satisfy the reliability requirement for a parallel application on heterogeneous platforms. Roy et al. [24] introduced an energy-efficient fault-tolerant framework for real-time tasks with precedence constraints on a heterogeneous dual core system. Han et al. [5] designed two algorithms that reduce energy consumption while meeting the reliability requirements of applications through task replication. Liu et al. [14] introduced a transient fault-tolerant scheduling algorithm AFTSA to improve system reliability within a given deadline. However, the above works require fault-detection mechanisms which imply that they can detect all faults during task execution.

2.4. N-modular redundancy scheduling

NMR executes multiple copies of a task in parallel and compares the execution results of these copies for fault detection and masking, which can achieve higher system reliability, although it increases energy consumption and reduces system utilization. Reliability is crucial for many safety-critical systems. Therefore, it is necessary to use NMR in many scenarios. In [25], a DRVS system is proposed to select a particular reliability mechanism (Single Execution, Dual Modular Redundancy, or TMR) and voltage-frequency level under reliability and time constraints. Salehi et al. [26] introduced a two-phase TMR technique to minimize energy consumption while guaranteeing reliability and deadline requirements on multi-core platforms. Mireshghallah et al. [16] proposed an energy-efficient reactive TMR approach to tolerate both transient and permanent faults. However, these works are implied based on a homogeneous multi-core processor, while we consider energy-efficient TMR technique on heterogeneous multi-core real-time systems.

3. Models

3.1. System model

The heterogeneous multi-core platform used in this paper is modeled as the core group architecture. SG is the set of groups, and for each $sg \in SG$ there are n_{sg} cores in this group. Therefore, the total number of cores in platform is $M = \sum_{sg=1}^{|SG|} n_{sg}$, where $|SG|$ is the size of the set SG. (In this paper, $|X|$ represents the size of set X.) In this case, the processor cores in the system can be represented as $\{core_1, core_2, ..., core_M\}$. For $core_i$, $group(i)$ represents the group in which it is located. The system architecture is shown in Fig. 1, where the core types within a group can be the same or different. This is a flexible model where M cores in the platform can be in the same group, or in M different groups (each group only has one core), or have the same type in the same group. Therefore, we assume that the communication cost between cores within the same group is much lower than that of cores in different groups. In addition, we assume that there is the same communication bandwidth between different groups and do not consider conflicts during data transmission. Similar to [42], each core in the platform is DVFS-enabled with a finite set of available execution frequencies and the frequency can be adjusted separately.

In the design of multi-core processors, there are typically two primary approaches to operating frequency management: shared execution frequency and independent core-level frequency adjustment.

The shared execution frequency approach benefits from a reduction in hardware complexity and associated costs, as it does not mandate separate dynamic frequency regulation components for each core. However, its disadvantage becomes evident when certain cores do not fully utilize the common frequency, potentially leading to energy waste. This approach also lacks the ability to dynamically adjust the performance of individual cores according to varying workloads.

On the other hand, the independent core-level frequency adjustment approach offers greater flexibility in resource allocation and enables more effective energy efficiency optimization. It is particularly advantageous in multitasking scenarios where resources can be dynamically allocated based on the real-time demands of each task, thereby minimizing energy consumption. The disadvantage of this approach is the requirement for an individual dynamic frequency control component for every core, which results in increased hardware costs.

This paper will investigate how to minimize the energy consumption of applications under the TMR mechanism. We hope to fully leverage the performance of each core, therefore, this paper adopts the approach of individually adjusting the execution frequency for every core.

Table 1 gives the definitions of notations used in this study.

3.2. Application model

The parallel application execution on the heterogeneous platform is usually modeled as DAG $G = (T, C, W)$, where T, C, and W are described as follows: $T = \{t_1, t_2, ..., t_{|T|}\}$ is a vertex set in G, which represents a task set in application. C is a edge set in G, $c_{i,j} \in C$ indicates the time required for data transmission from t_i to t_j, t_j can be executed only after data transmission is completed. Because a task may have multiple immediate predecessors or successors, let $parent(t_i)$ and $child(t_i)$ represent all immediate predecessor and successor tasks of t_i respectively. The task without a predecessor (or successor) task is called t_{entry} (or t_{exit}). In general, if there are multiple t_{entry} (or t_{exit}) in application G, we can construct a dummy task of entry (or exit) to G. A motivation example parallel application is shown in Fig. 2. Because the processor core is heterogeneous, the required execution time of the same task on different cores is different, so a matrix $W = |T| \times M$ is used to represent the Worst-Case Execution Time (WCET) of each task on the different cores with the maximum execution frequency. $w_{i,j}$ is the WCET of the task t_i executes on $core_j$. It should be noted that the same task requires the same execution time to execute on the same type of core. The WCET of the five tasks in Fig. 2 on three types of cores are shown in Table 2, the WCET of t_1 on three different types of cores is 19, 16, and 11, respectively.

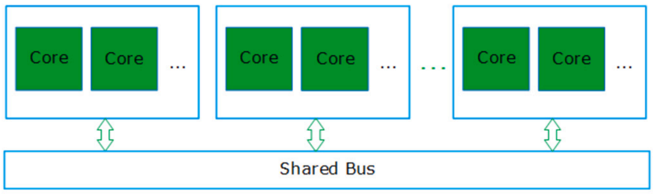


Fig. 1. System architecture.

Table 1
Definitions of notations.

Notation	Definition
$c_{i,j}$	communication time between t_i and t_j
$w_{i,j}$	WCET of the task t_i executes on $core_j$
$E_{\text{dy}}(t_{i,k}, core_j, f_{j,l})$	dynamic energy consumption of the kth copy of task t_i on $core_j$ at frequency $f_{j,l}$
$E_{\text{co}}(t_{i,k}, core_j)$	communication energy consumption of copy $t_{i,k}$ assigned to $core_j$
$E_{\text{rt}}(t_i, core_j)$	energy consumption of transmitting the execution results of all copies of task t_i to $core_j$
$E_{\text{vc}}(t_i, core_j, f_{j,l})$	energy consumption of the execution results of all copies of task t_i voting on $core_j$ with frequency $f_{j,l}$
$rtt_{i,k,j}$	the time required for the execution result of the copy $t_{i,k}$ to be transmitted to $core_j$
$vct_{i,j,l}$	the time required for the execution results of task t_i to be voted on $core_j$ with frequency $f_{j,l}$
$E_{\text{dcrv}}(t_i)$	the total amount of the dynamic energy consumption of task t_i
$E_{\text{dcrv}}(G)$	the total amount of the dynamic energy consumption of the application
$E_{\text{st}}(G)$	the static energy consumption of the application
$E(G)$	the total energy consumption of the application
$R(t_{i,k}, core_j, f_{j,l})$	the reliability of the copy $t_{i,k}$ executed on $core_j$ with frequency $f_{j,k}$
$R(t_i)$	the reliability of task t_i
$R(G)$	the reliability of the application
$EST(t_{i,k}, core_j)$	the earliest start time of the copy $t_{i,k}$ executing on $core_j$
$EFT(t_{i,k}, core_j)$	the earliest finish time of the copy $t_{i,k}$ executing on $core_j$
$ST(t_{i,k})$	the actual start time of the copy $t_{i,k}$
$FT(t_{i,k})$	the actual finish time of the copy $t_{i,k}$
$ExFT(t_{i,k})$	the extended finish time of the copy $t_{i,k}$
$STC(t_i, core_j)$	the start time for comparing the execution results of the copies of task t_i on $core_j$
$FTC(t_i, core_j)$	the finish time for comparing the execution results of the copies of task t_i on $core_j$

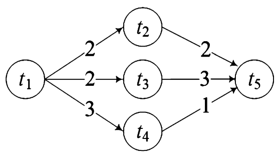


Fig. 2. Motivation example of a DAG.

Table 2
The WCET of the five tasks in motivation example on three types of cores.

task	core type 1	core type 2	core type 3
t_1	19	16	11
t_2	14	10	18
t_3	9	17	12
t_4	13	8	16
t_5	12	15	7

3.3. Energy consumption model

This work considers processor cores with adjustable voltage and frequency, according to [31][37][43], the power dissipation of CMOS chip at frequency f is given by

$$P(f) = P_{\mathrm{st}} + \hbar(P_{\mathrm{in}} + P_{\mathrm{dy}}) = P_{\mathrm{st}} + \hbar(P_{\mathrm{in}} + C_{\mathrm{sw}} f^m) \tag{1}$$

In (1), P_{st} is the static power dissipation, which is used to maintain the basic operating state of the system. P_{in} is the leakage power dissipation independent of the execution frequency, which is a constant and can be ignored when the processor is in a sleep state. P_{dy} is the dynamic power dissipation, which is related to the execution frequency. C_{sw} and m are the switching capacitance and the dynamic energy exponent, respectively.

Based on Eq. (1), the lowest energy-efficient frequency can be calculated as

$$f_{\mathrm{ee}} = \sqrt[m]{\frac{P_{\mathrm{in}}}{C_{\mathrm{sw}}(m-1)}}. \tag{2}$$

When the processor operates at a frequency lower than f_{ee}, it will generate higher energy consumption due to lower energy efficiency and longer execution time. Assume that the operating frequency range of the processor core is $f_{\min}$ to $f_{\max}$, to reduce energy consumption, the lowest execution frequency of the processor core should be $f_{\mathrm{low}} = \max(f_{\mathrm{ee}}, f_{\min})$. In practice, because the available execution frequency of the processor core is discrete, it is advisable to assume that the execution frequency of core j is $\{f_{j,\mathrm{low}}, \cdots, f_{j,l}, f_{j,l+1}, \cdots f_{j,\max}\}$. In this paper, these available frequencies are normalized with respect to the highest frequency $f_{j,\max}$, i.e. $f_{j,\max} = 1$.

In heterogeneous multi-core embedded systems, the cores with different core types have different power dissipation parameters. Let $E_{\mathrm{dy}}(t_{i,k}, core_j, f_{j,l})$ represent the dynamic energy consumption of the kth copy of task t_i on $core_j$ at frequency $f_{j,l}$, which can be calculated by

$$E_{\mathrm{dy}}(t_{i,k}, core_j, f_{j,l}) = (P_{j,\mathrm{in}} + C_{j,\mathrm{sw}} f_{j,l}^{m_j}) \times w_{i,j} \times \frac{f_{j,\max}}{f_{j,l}}. \tag{3}$$

Considering that different tasks may be executed on different cores in different groups, the communication energy consumption will be generated when there is communication between two different groups. Assuming that the communication energy consumption is proportional to the communication time, the energy consumption rate per unit time of communication is defined as cr [35]. Therefore, when kth copy of task t_i is assigned to core $core_j$, the energy consumption of communication can be calculated by

$$E_{\mathrm{co}}(t_{i,k}, core_j) = \sum\nolimits_{t_x \in parent(t_i)} cr \times c'_{x,i}. \tag{4}$$

In Eq. (4), if the copy $t_{i,k}$ and any correctly executed copy of t_x are assigned to the same group, then $c'_{x,i} = 0$; otherwise $c'_{x,i} = c_{x,i}$.

When the N-modular redundancy technique is used, we do not allow different copies of the same task to be assigned to the same core for execution. When N copies of task t_i are completed, assuming that their execution results are transmitted to $core_j$ for voting comparison, which consumes execution results transmission energy $E_{\mathrm{rt}}(t_i, core_j)$ and voting comparison energy $E_{\mathrm{vc}}(t_i, core_j, f_{j,l})$. The energy consumption for data transmission and voting comparison can be calculated as

$$E_{\mathrm{rt}}(t_i, core_j) = \sum\nolimits_{k=1}^{N} rtt_{i,k,j} \times cr \tag{5}$$

and

$$E_{\mathrm{vc}}(t_i, core_j, f_{j,l}) = (P_{j,\mathrm{in}} + C_{j,\mathrm{sw}} f_{j,l}^{m_j}) \times vct_{i,j,l} \times \frac{f_{j,\max}}{f_{j,l}} \tag{6}$$

respectively, where $rtt_{i,k,j}$ represents the time required for the execution result of $t_{i,k}$ to be transmitted to $core_j$, and $vct_{i,j,l}$ indicates the time required for the execution result to be voted on $core_j$ with frequency $f_{j,l}$.

For task t_i, let $core_{apc(i,k)}$ represent the assigned core of the kth copy, $f_{apc(i,k),aef(i,k)}$ represent the assigned execution frequency of the kth copy, $core_{cer(i)}$ represent the core used to compare the execution results, $f_{cer(i),cef(i,l)}$ represent the execution frequency used to compare the execution results.

According to the above description, for task t_i, the energy consumption for executing the task itself, for communicating with its parent task, for transmitting the execution results for comparison, and for the process of comparing the execution results can be calculated by

$$E_{\mathrm{dy}}(t_i) = \sum\nolimits_{k=1}^{N} E_{\mathrm{dy}}(t_{i,k}, core_{apc(i,k)}, f_{apc(i,k),aef(i,k)}), \tag{7}$$

$$E_{\mathrm{co}}(t_i) = \sum\nolimits_{k=1}^{N} E_{\mathrm{co}}(t_{i,k}, core_{apc(i,k)}), \tag{8}$$

$$E_{\mathrm{rt}}(t_i) = E_{\mathrm{rt}}(t_i, core_{cer(i)}), \tag{9}$$

and

$$E_{\mathrm{vc}}(t_i) = E_{\mathrm{vc}}(t_i, core_{cer(i)}, f_{cer(i),cef(i,l)}), \tag{10}$$

respectively.

Therefore, the total energy consumption for executing task t_i is given by

$$E_{\mathrm{dcrv}}(t_i) = E_{\mathrm{dy}}(t_i) + E_{\mathrm{co}}(t_i) + E_{\mathrm{rt}}(t_i) + E_{\mathrm{vc}}(t_i). \tag{11}$$

When all the tasks with N-modular redundancy are assigned, the energy consumption of the application is given by

$$E_{\mathrm{dcrv}}(G) = \sum\nolimits_{i=1}^{|T|} E_{\mathrm{dcrv}}(t_i). \tag{12}$$

Let $E_{\mathrm{st}}(G)$ represent the static energy consumption of the application, which is derived from all the processor cores and can be calculated by

$$E_{\mathrm{st}}(G) = \sum\nolimits_{j=1}^{M} P_{j,\mathrm{st}} \times SL(G), \tag{13}$$

where $SL(G)$ is the scheduling length of parallel application G. Based on (12) and (13), the total energy consumption of the application is given by

$$E(G) = E_{\mathrm{dcrv}}(G) + E_{\mathrm{st}}(G). \tag{14}$$

3.4. Reliability model

There are transient faults and permanent faults during the execution of the application. Since transient faults occur more commonly than permanent faults [47][48], only transient faults are considered in this study. In general, the occurrence of transient faults follows a Poisson process [35][43]. Given λ as the constant failure rate per time unit, the probability of no faults occurring (which represents system reliability) within the time interval t can be expressed as $e^{-\lambda t}$ [35][38][39]. In this paper, the reliability of a task (a copy of a task or an application) is defined as the probability that the task (copy or application) will be completed correctly before its deadline [48]. Therefore, in the subsequent discussion, we assume that a task (copy or application) can be completed before its deadline if no faults occur during its execution. If a task (copy) cannot meet its deadline constraint, the application will not be correctly scheduled by the algorithm proposed in this paper. Different cores have distinct λ values representing failure rates. Let $\lambda_{j,\max}$ represent the failure rate per time unit for $core_j$ execution with the maximum frequency, when the kth copy of task t_i is executed on $core_j$ with execution frequency $f_{j,\max}$, the reliability of this copy can be given by [35][38]

$$R(t_{i,k}, core_j, f_{j,\max}) = e^{-\lambda_{j,\max}\times w_{i,j}} \tag{15}$$

When the execution frequency is reduced to save energy, the probability of fault will increase. According to [43][47][48], the transient faults rate $\lambda_{j,l}$ of core $core_j$ with the frequency $f_{j,l}$ is given by

$$\lambda_{j,l} = \lambda_{j,\max} \times 10^{\frac{d_j\times(f_{j,\max}-f_{j,l})}{f_{j,\max}-f_{j,\min}}}, \tag{16}$$

where d_j indicates the sensitivity fault rates to voltage and frequency scaling of $core_j$, which is greater than 0.

According to Eq. (15), when the kth copy of task t_i is executed on core $core_j$ with frequency $f_{j,l}$, the reliability can be calculated by

$$R(t_{i,k}, core_j, f_{j,l}) = e^{-\lambda_{j,l}\times\frac{w_{i,j}\times f_{j,\max}}{f_{j,l}}}. \tag{17}$$

This work mainly studies the energy-efficiency scheduling problem with the TMR technique, that is, three copies of task t_i will be executed. According to Eq. (17), the reliability of the three copies is $R(t_{i,1})$, $R(t_{i,2})$, and $R(t_{i,3})$ respectively. Then the reliability of task t_i can be calculated by

$$\begin{aligned} R(t_i) = {} & R(t_{i,1})R(t_{i,2})(1 - R(t_{i,3})) \\ & + R(t_{i,1})R(t_{i,3})(1 - R(t_{i,2})) \\ & + R(t_{i,2})R(t_{i,3})(1 - R(t_{i,1})) \\ & \quad + R(t_{i,1})R(t_{i,2})R(t_{i,3}). \end{aligned} \tag{18}$$

The reliability of the task can be significantly improved by using the TMR technique. For example, when the reliability of each copy of a task is 0.9, the reliability of this task with TMR will increase to 0.972.

When the reliability of all tasks is calculated, the application's reliability can be given by

$$R(G) = \prod_{i=1}^{|T|} R(t_i). \tag{19}$$

3.5. Problem description

Consider a parallel application G using TMR execution on the heterogeneous multi-core platform, in which each core supports the DVFS technique. When three copies of all tasks in G are assigned to the processor core to execute at the appropriate frequency, the schedule length of the application is $SL(G)$. The problem in this paper is to minimize the energy consumption of the system while the schedule length $SL(G)$ is less than or equal to the given deadline $DL(G)$ and the reliability of the application G is higher than the given reliability requirement $R_{\text{req}}(G)$.

4. Energy-efficient TMR scheduling framework

To solve the problem of minimizing energy consumption under scheduling length and reliability constraints, we first design an algorithm that minimizes scheduling length while satisfying system reliability requirement, which is an improved version of the well-known HEFT algorithm (IHEFT). IHEFT can be used to determine whether the application can meet the scheduling length and reliability constraints with the TMR mechanism. Then we designed three energy-efficient algorithms to minimize energy consumption, which are the Extending Execution Time of the copies (EET) algorithm, Minimizing the Execution Overhead of the Third Copy (MEOTC) algorithm, and the Online Energy Management (OEM) algorithm. The specific introduction of these algorithms is as follows.

4.1. The IHEFT algorithm

4.1.1. Introduction to key principles

The primary objective of the IHEFT algorithm is to minimize the scheduling length while satisfying the reliability requirement of the application. Unlike the original HEFT algorithm, IHEFT initially transforms the application-level reliability requirement into corresponding task-level reliability requirements. Subsequently, it calculates the reliability requirement for each replica of a task within the context of the TMR framework. Following this, the algorithm assigns each copy to processor cores with the aim of achieving the minimum finish time. Finally, after all copies have been assigned and executed, a voting mechanism is implemented where the results from the three replicas of each task are compared and validated.

4.1.2. Computing reliability requirements for task copies

To make the reliability of the application satisfy the requirements, the reliability of each copy of the task should not be lower than the given critical value R_{critical} when using the TMR technique. Therefore, we should obtain R_{critical} based on the reliability requirement of the application.

When the reliability requirement $R_{\text{req}}(G)$ of the application G is given, the reliability requirement of the task $R_{\text{req}}(t_i)$ can be obtained by different methods [38][45][46]. For simplicity, this paper uses the method in literature [38], and the reliability requirement of the task is given by

$$R_{\text{req}}(t_i) = \sqrt[|T|]{R_{\text{req}}(G)}. \tag{20}$$

When assigning tasks, the reliability requirement of task t_i can be given by

$$R_{\text{req}}(t_i) \geq \frac{R_{\text{req}}(G)}{\prod_{x=1}^{i-1} R(t_x) \times \prod_{x=i+1}^{|T|} R_{\text{req}}(t_x)}, \tag{21}$$

where item $\prod_{x=1}^{i-1} R(t_x)$ is the reliability obtained by $i-1$ tasks that have been assigned, and item $\prod_{x=i+1}^{|T|} R_{\text{req}}(t_x)$ is the reliability expected to be obtained for tasks that have not yet been assigned.

Assuming that $R(t_{i,1}) = R(t_{i,2}) = R(t_{i,3}) = R_{\text{critical}}$ can make the task t_i just satisfy the reliability requirement, Eq. (18) can be rewritten as an univariate cubic equation

$$2R_{\text{critical}}^3 - 3R_{\text{critical}}^2 + R_{\text{req}}(t_i) = 0. \tag{22}$$

Based on recent derivations of the cubic solution [15], let R_{critical} be represented as

$$R_{\text{critical}} = x - \frac{-3}{3\times 2} = x + \frac{1}{2}, \tag{23}$$

and then the univariate cubic equation (22) can be expressed as

$$x^3 + px + q = 0, \tag{24}$$

where $p = -\frac{3}{4}$ and $q = -\frac{1}{4} + \frac{1}{2}R_{\text{req}}(t_i)$. Then we calculate the discriminant D using the formula $D = \frac{q^2}{4} + \frac{p^3}{27}$. Due to reliability requirement $0 < R_{\text{req}}(t_i) < 1$, it is easy to know that $D < 0$, so the univariate cubic equation (24) has three distinct real roots, which are

$$x_1 = 2\sqrt{-\frac{p}{3}}\cos\left(\frac{\pi}{6}+\frac{\theta}{3}\right) \tag{25}$$

$$x_2 = -2\sqrt{-\frac{p}{3}}\cos\left(\frac{\pi}{6}-\frac{\theta}{3}\right) \tag{26}$$

$$x_3 = 2\sqrt{-\frac{p}{3}}\sin\frac{\theta}{3} \tag{27}$$

where $\theta = \tan^{-1}\left(\frac{q}{2\sqrt{-D}}\right)$ and $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$. According to Eq. (23), we know that $R_{\text{critical}} = x + 0.5$. Because the reliability must be greater than 0 and less than 1, only x_3 provides a valid solution which ensures that R_{critical} falls within the required range of reliability. For x_1 and x_2, the corresponding reliabilities exceed 1 and fall below 0 respectively. To calculate the R_{critical}, for instance, if the $R_{\text{req}}(t_i) = 0.99$, then we can get $R_{\text{critical}} = 0.941097$. For more details on solving univariate cubic equations, please refer to reference [15].

4.1.3. *Determining minimum finish times for task copies*

HEFT [32] is a well-known algorithm to solve the problem of minimizing the scheduling length of a parallel application executing on heterogeneous systems, which has high performance and relatively short scheduling length. HEFT uses the concept of *Rank* to obtain the scheduling order of tasks, the *Rank* value of task t_i is defined as

$$Rank(t_i) = \overline{w_i} + \max_{t_j \in child(t_i)} \{c_{i,j} + Rank(t_j)\}, \tag{28}$$

where $\overline{w_i} = \left(\sum_{j=1}^{M} w_{i,j}\right) \Big/ M$ represents the average WCET of the task t_i on each core. The tasks scheduled in non-ascending order of *Rank* value can meet the requirements of execution order. Assuming that the task t_i mentioned below has been sorted by non-ascending order.

When an application executes on a heterogeneous system with TMR technique, for the kth copy of task t_i is assigned to core j, let $EST(t_{i,k}, core_j)$ represent the earliest start time in the worst-case scenario and $EFT(t_{i,k}, core_j)$ represent the earliest finish time. In addition, let $ST(t_{i,k})$ represent the actual start time of the kth copy of task t_i and $FT(t_{i,k})$ represent the actual finish time of the kth copy of task t_i. The start time and finish time for comparing the execution results of three copies of task t_i on $core_j$ can be calculated by

$$STC(t_i, core_j) = \max_{1\le k\le 3}\{FT(t_{i,k}) + rtt_{i,k,j}\} \tag{29}$$

and

$$FTC(t_i, core_j) = STC(t_i, core_j) + vct_{i,j,l}, \tag{30}$$

respectively. During the scheduling, let $FTC(t_i)$ represent the finish time of comparing the execution results of the three copies of task t_i. When the copy $t_{i,k}$ is assigned to $core_j$, its earliest start time is related to the predecessor task. Assuming that the communication time overheads from the three copies $t_{x,1}$, $t_{x,2}$, and $t_{x,3}$ of the predecessor task t_x to $core_j$ are ct_1, ct_2, and ct_3, respectively, and $ct_1 \ge ct_2 \ge ct_3$. Because the copy $t_{i,k}$ can obtain data from any correctly executed copy when executing on $core_j$, the worst-case scenario is a transient fault occurred in $t_{x,3}$. In this case, $t_{i,k}$ obtains data from $t_{x,2}$, resulting in a worst-case communication time overhead of ct_2. Since we do not know which copy will occur transient faults, this article calculates the earliest start time of $t_{i,k}$ based on the worst-case communication time as

$$\begin{cases} EST(t_{\text{entry},k}, core_j) = 0 \\ EST(t_{i,k}, core_j) = \max\left\{ \begin{array}{l} avail[j], \\ \max\limits_{t_x \in parent(t_i)} \{FTC(t_x) + wcct_{x,i,j}\} \end{array} \right\} \end{cases} \tag{31}$$

where $avail[j]$ represents the earliest available time of $core_j$ and $wcct_{x,i,j}$ represents the worst-case communication time of data transmitting from task t_x to $core_j$ for executing the copy of t_i.

Based on Eq. (31), the earliest finish time of $t_{i,k}$ on $core_j$ can be calculated by

$$EFT(t_{x,k}, core_j) = EST(t_{x,k}, core_j) + w_{x,j}. \tag{32}$$

Finally, the scheduling length of the application G is given by

$$SL(G) = FTC(t_{\text{exit}}). \tag{33}$$

4.1.4. *Detailed design of the IHEFT algorithm*

Based on the previous analysis, we designed the IHEFT algorithm as shown in Algorithm 1.

The IHEFT algorithm first determines the priority of the task (Line 1), and then assigns three copies of the task to the appropriate core for execution. For each task, IHEFT first calculates its reliability requirement (Line 5) and then assigns its three copies (Lines 6-15). To avoid assigning different copies of a task to the same core, a matrix S with n rows and 3 columns is defined to represent the processor cores to which each copy of the task is assigned, where n denotes the number of tasks. The i-th row of the matrix corresponds to the processors to which the three copies of task t_i are assigned. The IHEFT algorithm initializes all elements in matrix S to 0 before assigning tasks (Line 2). When assigning a copy to a core, it first determines whether the core has been occupied by other copies. If the processor core is unoccupied (Line 8), IHEFT calculates the finish time and reliability of the copy (Lines 9-10). Then IHEFT assigns the copy to the core that can complete it earliest while satisfying the reliability requirement. After a copy is assigned, IHEFT adds the corresponding core to the matrix S (Line 14). Finally, the execution results of the three copies are compared (Line 16), and if the completion time for this comparison exceeds the application's deadline, the algorithm returns a 'false' value (Lines 17-19).

Algorithm 1 IHEFT.

```
Input: $PC = \{core_1, core_2, ...core_M\}$, application $G$, and $R_{\text{req}}(G)$
Output: the start time and finish time of all copies
1: sort the tasks to queue $ReadyQ$ by non-ascending order of $Rank$
2: initialize all elements in matrix $S$ to 0
3: while $ReadyQ$ is not empty do
4:     $t_i \leftarrow ReadyQ.out()$
5:     calculate $R_{\text{critical}}$ using Eq. (27)
6:     for $k \leftarrow 1$ to 3 do
7:         for $j \leftarrow 1$ to $M$ do
8:             if $j$ is not in the $i$-th row of the matrix $S$ then
9:                 calculate $EFT(t_{x,k}, core_j)$ using Eq. (32)
10:                calculate $R(t_{i,k}, core_j, f_{j,\max})$ using Eq. (15)
11:            end if
12:        end for
13:        assign copy $t_{i,k}$ to the $core_j$ that minimizes EFT of copy $t_{i,k}$ and satisfies
           $R(t_{i,k}, core_j, f_{j,\max}) \ge R_{\text{critical}}$ // the start time and finish time of the
           copy $t_{i,k}$ can be obtained
14:        assign the element at the $k$-th column of the $i$-th row in matrix $S$ to
           the value of $j$
15:    end for
16:    Transfer the execution results of the two earlier-finished copies to the
       core where the third copy is assigned for voting comparison
17:    if the completion time of the voting comparison is greater than the dead-
       line $DL(G)$ then
18:        return false
19:    end if
20: end while
```

HEFT has a time complexity of $O(|T|^2 \times M)$, three copies of each task need to be assigned when using the TMR technique, so the time complexity of IHEFT is $O((3 \times |T|)^2 \times M)$.

4.2. *The EET algorithm*

When the IHEFT algorithm is completed, the scheduling length $SL_{\text{IHEFT}}(G)$ can be obtained. If the given deadline $DL(G) > SL_{\text{IHEFT}}(G)$, there is a slack time between each task. The slack time can be used to reduce the execution frequency and thus save energy. To more intuitively describe the degree of relaxation, the slack ratio (SR) is defined as the ratio of the given deadline to the scheduling length generated by IHEFT, which is expressed as

$$SR(G) = \frac{DL(G)}{SL_{\text{IHEFT}}(G)}. \tag{34}$$

After all copies of all tasks are scheduled with the IHEFT algorithm, the start time and finish time of each copy can be obtained. If the execution time of the task is extended by $SR(G)$ times, the application can still meet the deadline requirements. Therefore, we can reduce the execution frequency and extend the execution time of all copies for improving energy efficiency. The extended finish time (ExFT) of the kth copy of task t_i is given by

$$ExFT(t_{i,k}) = FT(t_{i,k}) \times SR(G). \tag{35}$$

For any i and k, if the finish time of the copy $t_{i,k}$ does not exceed $ExFT(t_{i,k})$, the application can meet the deadline. Because the tasks with extended execution time may not be suitable for the frequency level requirements or may not satisfy the reliability requirements, it is necessary to adjust the execution frequency of all the copies of all tasks. During the scheduling, if the copy $t_{i,k}$ is assigned to $core_j$ with the execution frequency $f_{j,l}$, the finish time should not exceed $ExFT(t_{i,k})$; Otherwise, the application may not meet the deadline. Therefore, the EET algorithm for adjusting the execution frequency can be designed as Algorithm 2.

Algorithm 2 EET.

```
Input: core set $\{core_1, core_2, ...core_M\}$, $R_{\text{req}}(G)$, $SR(G)$, and the results of IHEFT
Output: $E(G)$, $SL(G)$ and $R(G)$
1: reload the ready queue $ReadyQ$
2: obtain extended finish time (ExFT) of each copy using Eq. (35)
3: while $ReadyQ$ is not empty do
4:   $t_i \leftarrow ReadyQ.out()$
5:   calculate $R_{\text{critical}}$ using Eq. (27)
6:   for $k \leftarrow 1$ to 3 do
7:     $core_j \leftarrow$ get the core that executes copy $t_{i,k}$
8:     calculate $EST(t_{i,k}, core_j)$ using Eq. (31) //the earliest start time of the copy $t_{i,k}$ on the core $core_j$
9:     for frequency $f_{j,l} \leftarrow f_{j,\text{low}}$ to $f_{j,\max}$ do
10:      if $w_{i,j} \times \frac{f_{j,max}}{f_{j,l}} \le ExFT(t_{i,k}) - EST(t_{i,k}, core_j)$ and $R(t_{i,k}, core_j, f_{j,l}) \ge R_{\text{critical}}$ then
11:        $ST(t_{i,k}) \leftarrow EST(t_{i,k}, core_j)$
12:        $FT(t_{i,k}) \leftarrow ST(t_{i,k}) + w_{i,j} \times \frac{f_{j,\max}}{f_{j,l}}$
13:        break
14:      end if
15:    end for
16:  end for
17:  Transfer the execution results of the two earlier-finished copies to the core where the third copy is assigned for voting comparison
18: end while
19: calculate $E(G)$, $SL(G)$, and $R(G)$ using Eqs. (14), (33), and (19), respectively
```

The EET algorithm saves energy by reducing the execution frequency of the tasks, the details of which are explained as follows.

The input part of the EET algorithm is core set PC, $R_{\text{req}}(G)$, $SR(G)$, and the results of IHEFT, the output part is $E(G)$ and $SL(G)$. Line 1 reloads the ready queue $ReadyQ$. Line 2 indicates that the execution time of the tasks is extended. Lines 3-18 are a nested loop, which traverses all the tasks in the application G. For any task t_i, the EET algorithm first calculates the reliability requirement and then traverses the execution frequency to make each copy meets the deadline and satisfies the reliability requirement (Lines 5-16). Line 17 compares the execution results by voting. Line 19 calculates $E(G)$, $SL(G)$, and $R(G)$ respectively.

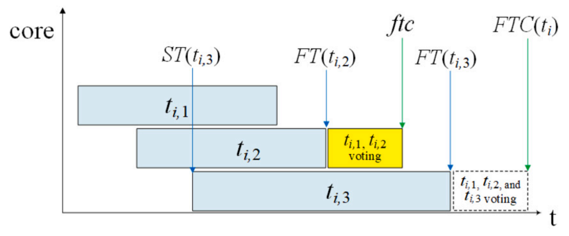


Fig. 3. Example of three copies of tasks t_i.

The EET algorithm needs to traverse all copies of all tasks. Line 8 calculates $EST(t_{i,k}, core_j)$, which needs to traverse all the precursor tasks of $t_{i,k}$. Lines 9-15 traverse all frequencies of $core_j$. Therefore, the time complexity of EET is $O(3 \times |T| \times (3 \times |T| + FL))$, where FL represents the maximum number of frequency levels of all cores.

4.3. The MEOTC algorithm

When the IHEFT algorithm is completed, the start time and finish time of all copies for all tasks can be obtained. Because the cores are heterogeneous, the start time and finish time of different copies of the same task may be different. In a TMR system, as long as two copies of a task are executed correctly, the results of the task will be correct. Therefore, it may not be necessary to complete the execution of the third copy, which can save more energy. The main design idea of the MEOTC algorithm using this mechanism is as follows.

(1) The execution frequency of the three copies should be reduced as much as possible to reduce energy consumption. When the first two copies are completed, their execution results will be voted in advance. If no transient fault occurs, terminate the execution of the third copy. Otherwise, after the third copy is completed, their execution results will be voted together again.

(2) Because the third copy may be terminated prematurely, its execution overhead should be minimized.

We will use Fig. 3 to illustrate the implementation methods of the three main design ideas mentioned above.

Fig. 3 shows an example of three copies of tasks t_i after using the IHEFT algorithm. Since the three copies are independent of each other, it is advisable to assume that copy $t_{i,3}$ has the maximum finish time of the three copies of t_i, and similarly, copy $t_{i,2}$ has the maximum finish time among the remaining two copies (i.e. $FT(t_{i,3}) \ge FT(t_{i,2}) \ge FT(t_{i,1})$). In addition, assuming that the three copies $t_{i,1}$, $t_{i,2}$, and $t_{i,3}$ are assigned to $core_a$, $core_b$, and $core_c$, respectively. The energy-efficient methods that use the TMR mechanism are as follows.

(1) If the execution results of $t_{i,1}$ and $t_{i,2}$ are voted in advance, the actual finish time of all copies should not exceed the extended finish time, otherwise it may result in the deadline not being met. Therefore, the available execution time for each copy should be determined.

For the copy $t_{i,1}$, its available execution time should not exceed $ExFT(t_{i,1}) - EST(t_{i,1}, core_a)$. If the execution results of $t_{i,1}$ and $t_{i,2}$ are voted in advance on $core_b$, it is necessary to transfer the results of $t_{i,1}$ to $core_b$. In this case, the available execution time of $t_{i,1}$ should not exceed $ExFT(t_{i,2}) - EST(t_{i,1}, core_a) - rtt_{i,1,b} - vct_{i,b,l}$. Therefore, the available execution time of $t_{i,1}$ is given by

$$aet_1 = \min \begin{pmatrix} ExFT(t_{i,1}) - EST(t_{i,1}, core_a), \\ ExFT(t_{i,2}) - EST(t_{i,1}, core_a) - rtt_{i,1,b} - vct_{i,b,l} \end{pmatrix}. \tag{36}$$

When the previous two copies $t_{i,1}$ and $t_{i,2}$ are completed, their execution results are voted on $core_b$, the available execution time of $t_{i,2}$ should not exceed $ExFT(t_{i,2}) - EST(t_{i,2}, core_b) - vct_{i,b,l}$. In addition, if a transient fault occurs in one of the copies $t_{i,1}$ and $t_{i,2}$, the execution results of these two copies need to be transmitted to $core_c$ to compare the results of the three copies again. In this case, the available execution time of $t_{i,2}$ should not exceed $ExFT(t_{i,3}) - EST(t_{i,2}, core_c) - rtt_{i,1,c} - vct_{i,b,l}$. Therefore, the available execution time of $t_{i,2}$ is given by

$$aet_2 = \min\left(\begin{array}{l} ExFT(t_{i,2}) - EST(t_{i,2}, core_b) - vct_{i,b,l}, \\ ExFT(t_{i,3}) - EST(t_{i,2}, core_b) - rtt_{i,1,c} - vct_{i,b,l} \end{array}\right). \tag{37}$$

For the copy $t_{i,3}$, its finish time should not exceed $ExFT(t_{i,3})$, so its available execution time is given by

$$aet_3 = ExFT(t_{i,3}) - EST(t_{i,3}). \tag{38}$$

(2) If both copies $t_{i,1}$ and $t_{i,2}$ have no faults and the completion time for voting on their execution results is ftc, the copy $t_{i,3}$ is not necessary to be executed in the time interval $[ftc, FT(t_{i,3})]$. In this case, the execution time et of $t_{i,3}$ is $ftc - ST(t_{i,3})$, so the energy consumption of execution $t_{i,3}$ is given by

$$EC(t_{i,3}) = \begin{cases} (P_{c,\text{ind}} + C_{c,\text{ef}} f_{c,l}^{m_c}) \times et & et > 0 \\ 0 & et \le 0 \end{cases}. \tag{39}$$

Then, we set the finish time of $t_{i,3}$ to $ExFT(t_{i,3})$ and try to increase the execution frequency of $t_{i,3}$, thereby delaying its start time and recalculating the energy consumption. In this way, the minimum energy consumption for executing $t_{i,3}$ can be found.

(3) If the first two copies cannot vote in advance due to insufficient available execution time, or if a transient fault occurs in the first two copies, the execution results of the three copies will be voted on $core_c$

Based on the above analysis, the MEOTC algorithm is designed as shown in Algorithm 3, and its details are explained as follows.

The input part of the MEOTC algorithm is the core set, the application G, and the results of the IHEFT algorithm, the output part is $E(G)$, $SL(G)$, and $R(G)$. Line 1 initializes ready queue $ReadyQ$. Line 2 obtains the extended finish time of each copy. Lines 3-38 are a nested loop, which traverses all the tasks in the application G. For any task t_i, MEOTC first obtains the available execution time of each copy (Lines 9-15), and then MEOTC enters the inner loop, which traverses the execution frequency of the core to find the minimum energy consumption of each copy (Lines 16-21). If the first two copies complete execution and immediately vote, MEOTC will reconsider the possible minimum energy consumption of the third copy (Lines 23-34). If the first two copies cannot be voted or if a transient fault occurs, the execution results of the three copies will be voted (Lines 35-37).

The time complexity of the MEOTC algorithm is analyzed as follows. The MEOTC algorithm is mainly loop nesting, the outer loop traverses each task, and the inner loop needs to calculate the earliest start time of each copy, which requires traversing each copy of the task. In addition, the inner loop also needs to traverse the core's execution frequency for each copy. Therefore, the time complexity of MEOTC is $O(3 \times |T| \times (3 \times |T| + FL))$, where FL represents the maximum number of frequency levels of all cores.

Algorithm 3 MEOTC.

```
Input: core set $\{core_1, core_2, ...core_M\}$, the application $G$, and the results of IHEFT
Output: $E(G)$, $SL(G)$ and $R(G)$
1: reload the ready queue $ReadyQ$
2: obtain extended finish time (ExFT) of each copy using Eq. (35)
3: while $ReadyQ$ is not empty do
4:   $t_i \leftarrow ReadyQ.out()$
5:   calculate $R_{\text{critical}}$ using Eq. (27)
6:   obtain the copy $t_{i,3}$, $t_{i,2}$, and $t_{i,1}$ that have $FT(t_{i,3}) \ge FT(t_{i,2}) \ge FT(t_{i,1})$
7:   $vote\_in\_advance \leftarrow$ true
8:   for $k \leftarrow 1$ to 3 do
9:     $core_j \leftarrow$ the core that executes copy $t_{i,k}$
10:    calculate $EST(t_{i,k}, core_j)$ using Eq. (31)
11:    calculate the available execution time $aet_k$ according to Eqs. (36), (37), and (38)
12:    if $aet_k < w_{i,j}$ then
13:      $aet_k \leftarrow ExFT(t_{i,k}) - EST(t_{i,k})$
14:      $vote\_in\_advance \leftarrow$ false
15:    end if
16:    for frequency $f_{j,l} \leftarrow f_{j,\text{low}}$ to $f_{j,\max}$ do
17:      if $w_{i,j} \times \frac{f_{j,max}}{f_{j,l}} \le aet_k$ and $R(t_{i,k}, core_j, f_{j,l}) \ge R_{\text{critical}}$ then
18:        $ST(t_{i,k}) \leftarrow EST(t_{i,k}, core_j)$
19:        $FT(t_{i,k}) \leftarrow ST(t_{i,k}) + w_{i,j} \times \frac{f_{j,\max}}{f_{j,l}}$
20:      end if
21:    end for
22:  end for
23:  if $vote\_in\_advance$ is true then
24:    transfer the execution result of $t_{i,1}$ to the core where $t_{i,2}$ is located for voting
25:    obtain $core_j$ to execute the copy $t_{i,3}$
26:    $ec \leftarrow +\infty$
27:    for execution frequency $f \leftarrow f_{j,l}$ to $f_{j,\max}$ do
28:      calculate $EC(t_{i,3})$ using Eq. (39)
29:      if $EC(t_{i,3}) < ec$ then
30:        $ec \leftarrow EC(t_{i,3})$
31:        mark $f_{j,l}$ and $AST(t_{i,3})$
32:      end if
33:    end for
34:  end if
35:  if $vote\_in\_advance$ is false or a transient fault occurs in the first two copies then
36:    transfer the execution results of $t_{i,1}$ and $t_{i,2}$ to the core where $t_{i,3}$ is located for voting
37:  end if
38: end while
39: calculate $E(G)$, $SL(G)$, and $R(G)$ using Eqs. (14), (33), and (19), respectively
```

4.4. Case study

The following is a case study of the proposed algorithms to execute the motivation example (see section 3.2). Assuming that the parameters of the three types of cores in the motivation example are in Table 3, where each core type has two cores and is assigned to the same group. To transmit the execution results to a certain core for voting, the transmission time $rtt_{i,k,j}$ is set as the maximum communication time between task t_i and the direct successor tasks. In addition, $rtt_{5,k,j}$ is set to 2. If the execution result is within the same group, the transmission time $rtt_{i,k,j} = 0$. The voting comparison time is set to 1, and the communication energy consumption rate cr is set to 0.2. The maximum frequency of each core is 1.0, and the difference between adjacent frequencies is 0.1. The reliability requirement of the application is $R_{\text{req}}(G) = 0.99$.

Table 4 shows the results of the IHEFT algorithm scheduling motivation application G in Fig. 2, where ST, FT, FTC, E_{co}, E_{dy}, E_{rt}, E_{vc}, and E_{dcrv} represent the start time, finish time, voting finish time, communication energy consumption with the predecessor task, dynamic energy consumption of the core, energy consumption for transmitting execution results, energy consumption of executing result voting, and the

Table 3
The parameters of the processor cores in motivation example.

Core type	P_s	P_{ind}	C_{ef}	m	$f_{\max}$	λ	d
type 1	0.001	0.03	1.2	2.9	1.0	0.0002	2.3
type 2	0.001	0.05	0.9	2.8	1.0	0.0005	2.1
type 3	0.001	0.04	1.1	3.0	1.0	0.0009	2.2

total amount of the dynamic energy consumption, respectively. Finally, scheduling length $SL(G) = 56$ and $E_{\text{dcrv}}(G) = 187.42$. The static energy consumption is $0.001 \times 6 \times 56 = 0.34$. Therefore, the total energy consumption of the application is $187.42 + 0.34 = 187.76$.

Assuming that the given $SR(G)$ is 1.4 (i.e. the deadline of the application is 78.4), the scheduling results of the EET algorithm are shown in Table 5, which shows the execution frequency of all copies of the task is reduced. Finally, the reliability of the application is 0.9915076, which is higher than 0.99. The scheduling length generated by EET is 70.03, which is less than $1.4 \times SL_{\text{IHEFT}}(G) = 78.4$. The total amount of the dynamic energy consumption of EET is 136.81, and the static en-

Table 4
The results of the motivation example using IHEFT.

Task	Core	ST	FT	FTC	Reliability	E_{co}	E_{dy}	E_{rt}	E_{vc}	E_{dcrv}
1	5	0	11	0	0.990149	0	12.54	0	0	12.54
1	6	0	11	0	0.990149	0	12.54	0	0	12.54
1	3	0	16	17	0.992032	0	15.2	1.2	0.95	17.35
2	3	19	29	0	0.995012	0	9.5	0	0	9.5
2	4	19	29	0	0.995012	0	9.5	0	0	9.5
2	1	19	33	34	0.997204	0.4	17.22	0.8	1.23	19.65
3	2	19	28	0	0.998202	0	11.07	0	0	11.07
3	5	17	29	0	0.989258	0	13.68	0	0	13.68
3	6	17	29	32	0.989258	0	13.68	0.6	1.14	15.42
4	3	29	37	0	0.996008	0	7.6	0	0	7.6
4	4	29	37	0	0.996008	0	7.6	0	0	7.6
4	2	28	41	42	0.997403	0	15.99	0.4	1.23	17.62
5	5	43	50	0	0.993720	0.6	7.98	0	0	8.58
5	6	43	50	0	0.993720	0	7.98	0	0	7.98
5	1	43	55	56	0.997603	0	14.76	0.8	1.23	16.79
$SL(G) = 56$, $E_{dcrv}(G) = 187.42$, $R(G) = 0.999436$										

Table 5
The results of the motivation example using EET after IHEFT.

Task	Core	Frequency	ST	FT	FTC	Reliability	E_{co}	E_{dy}	E_{rt}	E_{vc}	E_{dcrv}
1	5	0.9	0	12.22	0	0.977573	0	10.29	0	0	10.29
1	6	0.9	0	12.22	0	0.977573	0	10.29	0	0	10.29
1	3	0.9	0	17.78	18.78	0.982421	0	12.80	1.2	0.95	14.95
2	3	0.8	20.78	33.28	0	0.975425	0	6.65	0	0	6.65
2	4	0.8	20.78	33.28	0	0.975425	0	6.65	0	0	6.65
2	1	0.8	20.78	38.28	39.28	0.984232	0.4	11.52	0.8	1.23	13.95
3	2	0.7	20.78	33.64	0	0.975425	0	5.87	0	0	5.87
3	5	0.9	18.78	32.11	0	0.975560	0	11.23	0	0	11.23
3	6	0.9	18.78	32.11	37.64	0.975560	0	11.23	0.6	1.14	12.97
4	3	0.8	33.28	43.28	0	0.980291	0	5.32	0	0	5.32
4	4	0.8	33.28	43.28	0	0.980291	0	5.32	0	0	5.32
4	2	0.8	33.64	49.89	50.89	0.985350	0	10.70	0.4	1.23	12.33
5	5	0.8	51.89	60.64	0	0.967072	0.6	5.28	0	0	5.88
5	6	0.8	51.89	60.64	0	0.967072	0	5.28	0	0	5.28
5	1	0.7	51.89	69.03	70.03	0.967368	0	7.83	0.8	1.23	9.86
$SL(G) = 70.03$, $E_{dcrv}(G) = 136.81$, $R(G) = 0.9915076$											

ergy consumption is $0.001 \times 6 \times 70.03 = 0.42$. Therefore, the total energy consumption is $136.81 + 0.42 = 137.23$, which is 73.09% of the IHEFT algorithm.

Table 6 shows the results of the motivation example using MEOTC after IHEFT. When there are no faults in the first two completed copies of the tasks, the third copy can be terminated prematurely to save energy. For example, the copy $t_{1,3}$ can be terminated at time 13.22 when the execution of $t_{1,1}$ and $t_{1,2}$ is completed at time 12.22 and their execution results completed the voting at time 13.22. If any transient faults occur in the first two copies, the third copy $t_{1,3}$ must be executed completely and its execution results must be voted together with the execution results of the first two copies. We have marked the execution finish time and voting finish time of the third copy when fully executed in the brackets in columns 5 and 6 of Table 6, respectively. Considering that the probability of faults occurring in the first two copies is not high, the MEOTC algorithm tries to minimize the energy consumption of the third copy, so MEOTC delays the start time of the third copy to 4.62. When all copies are executed without any transient faults, the energy consumption for executing the copy $t_{1,3}$ will be reduced from 12.80 to 6.19. Finally, the reliability of the application is 0.993746, which satisfies the reliability requirement. The total amount of the dynamic energy consumption is 121.04, and the static energy consumption is $0.001 \times 6 \times 78 = 0.47$. Therefore, the total energy consumption of the application is $121.04 + 0.47 = 121.51$, which is 64.72% of the IHEFT algorithm.

It should be noted that any failure must be considered from the perspective of reliability. However, from the perspective of average energy consumption, we do not need to consider the cases in which the system tolerates a fault [26]. For example, consider the task t_1, the reliability of the copies $t_{1,1}$ and $t_{1,2}$ are 0.977573 and 0.977573, respectively. Therefore, the probability of no fault occurs during execution $t_{1,1}$ and $t_{1,2}$ is $0.977573 \times 0.977573 = 0.955649$. When no fault occurs, the energy consumption for executing task t_1 is $10.29 + 11.43 + 6.19 = 27.91$. If a fault occurs during the execution of $t_{1,1}$ and $t_{1,2}$, the copy $t_{1,3}$ must be executed completely and hence the energy consumption is $10.29 + 11.43 + 14.95 = 36.67$. From the above analysis, it can be seen that the average energy consumption for executing task t_1 is $0.955649 \times 27.91 + (1 - 0.955649) \times 36.67 = 28.30$, which is very close to the energy consumption when no faults occur (27.91).

4.5. Online energy management

We observe Table 6 and find that when the three copies of the first task are completed at time 13.22, the copies of the second task are not executed immediately, and the corresponding start times are 25.4, 25.4, and 28.7 respectively. When the task is executed online, we can make

Table 6
The results of the motivation example using MEOTC after IHEFT.

Task	Core	Frequency	ST	FT	FTC	Reliability	E_{co}	E_{dy}	E_{rt}	E_{vc}	E_{dcrv}
1	5	0.9	0	12.22	0	0.977573	0	10.29	0	0	10.29
1	6	0.9	0	12.22	13.22	0.977573	0	10.29	0	1.14	11.43
1	3	0.9	4.62	13.22 [22.4]	[23.4]	0.982421	0	6.19	0	0	6.19
2	3	0.8	25.4	37.9	0	0.975425	0.4	6.65	0	0	7.05
2	4	0.8	25.4	37.9	38.9	0.975425	0	6.65	0	0.95	7.60
2	1	0.8	28.7	38.9 [46.2]	[47.2]	0.984232	0.4	6.71	0	0	7.11
3	2	0.9	25.4	35.4	0	0.995747	0	9.14	0	0	9.14
3	5	1	23.4	35.4	39.4	0.989258	0	13.68	0.6	1.14	15.42
3	6	0.9	27.27	39.4 [40.6]	[41.6]	0.975560	0	10.22	0	0	10.22
4	3	0.8	37.9	47.9	0	0.980291	0	5.32	0	0	5.32
4	4	0.8	38.9	48.9	49.9	0.980291	0	5.32	0	0.95	6.27
4	2	0.7	38.83	49.9 [57.4]	[58.4]	0.964697	0	5.05	0	0	5.05
5	5	1	59.4	66.4	0	0.993720	0.6	7.98	0	0	8.58
5	6	0.8	59.4	68.15	69.4	0.967072	0	5.28	0	1.14	6.42
5	1	0.7	59.86	69.4 [77]	[78]	0.967368	0.6	4.36	0	0	4.96

$SL(G) = 78$, $E_{dcrv}(G) = 121.04$, $R(G) = 0.993746$

Table 7
The results of online energy management after MEOTC.

Task	Core	Frequency	ST	FT	FTC	Reliability	E_{co}	E_{dy}	E_{rt}	E_{vc}	E_{dcrv}
1	5	0.9	0	12.22	0	0.977573	0	10.29	0	0	10.29
1	6	0.9	0	12.22	13.22	0.977573	0	10.29	0	1.14	11.43
1	3	0.9	4.62	13.22	0	0.982421	0	6.19	0	0	6.19
2	3	0.8	14.22	26.72	0	0.975425	0.4	6.65	0	0	7.05
2	4	0.8	14.22	26.72	27.72	0.975425	0	6.65	0	0.95	7.60
2	1	0.8	28.70	28.70	0	0.984232	0.4	0.00	0	0	0.4
3	2	0.9	14.22	24.22	0	0.995747	0	9.14	0	0	9.14
3	5	1	13.22	25.22	29.22	0.989258	0	13.68	0.6	1.14	15.42
3	6	0.9	27.27	29.22	0	0.975560	0	1.65	0	0	1.65
4	3	0.8	26.72	36.72	0	0.980291	0	5.32	0	0	5.32
4	4	0.8	27.72	37.72	38.72	0.980291	0	5.32	0	0.95	6.27
4	2	0.7	38.83	38.83	0	0.964697	0	0.00	0	0	0
5	5	1	38.72	45.72	0	0.993720	0.6	7.98	0	0	8.58
5	6	0.8	38.72	47.47	48.72	0.967072	0	5.28	0	1.14	6.42
5	1	0.7	59.86	59.86	0	0.967368	0.6	0	0	0	0.6

$SL(G) = 48.72$, $E_{dcrv}(G) = 96.35$, $R(G) = 0.993746$

the first two copies of the second task start at 14.22 (the communication time between t_1 and t_2 is 1), and the third copy still starts at 28.7. By doing so, the first two copies can be completed in advance. If there are no faults in these two copies, the execution time of the third copy will be shortened, thus saving more energy. If any fault occurs during the execution of the first two copies, the execution result of the third copy will be voted together with the first two copies. In this way, the application can still meet the deadline. Based on this idea, when no fault occurs, the scheduling results of the online energy management (OEM) are shown in Table 7.

As can be seen from Table 7, the execution time of the third copy becomes shorter except for the first task, so they consume less energy. Even if the third copy and other copies are assigned to the same type of core, it can still save energy when scheduling online. For example, two copies $t_{3,2}$ and $t_{3,3}$ of task t_3 are assigned to the same type of cores (the third type of core, that is $core_5$ and $core_6$), the third copy $t_{3,3}$ (assigned to $core_6$) can still delay the start execution time to 27.27 and terminate at time 29.22. So the energy consumed by the copy $t_{3,3}$ can be reduced from 10.22 to 1.65. Note that some copies may not need to be executed at all, for example, the copies $t_{4,3}$ and $t_{5,3}$ do not need to be executed. Finally, the dynamic energy consumption of the application is 96.35, and the static energy consumption is $0.001 \times 6 \times 48.72 = 0.29$. Therefore, the total energy consumption is $96.35 + 0.29 = 97.64$, which is 52.00% of the IHEFT algorithm.

5. Experimental performance evaluation

5.1. Experimental parameters

We a use C++ program to simulate the proposed algorithms. The parameter values of the multi-core platform, power consumption of core, transient faults rate of core, and the WCET of a task on each core are taken from [4], [35], [36], and [37]. These parameters are shown in Table 8, where cores within the same group have the same parameters. During the experiment, we saved the core parameters, parallel application DAG parameters, and the worst-case execution time of each task on each type of core to files separately, so that different algorithms are based on the same platform and parallel applications.

Since IHEFT does not adjust the execution frequency of the core and all copies of all tasks are completely executed, IHEFT is treated as the original TMR algorithm. Because no similar algorithm is found for parallel application on heterogeneous platforms with TMR, we compared our algorithms with a recently proposed AFTSA algorithm [14], which uses task replication techniques to maximize system reliability within a given deadline on heterogeneous platforms. To be fair, we consider the

Table 8
The value range of each experimental parameter.

Experimental parameters	Value ranges
Number of groups $\|SG\|$	3
Number of cores n_{sg} within each group	4
Leakage power $P_{j,\text{in}}$	[0.03, 0.07]
Switching capacitance $C_{j,\text{sw}}$	[0.8, 1.2]
Dynamic energy exponent m_j	[2.5, 3.0]
Maximum frequency $f_{j,\max}$	1.0 GHz
Precision of frequency adjustment	0.1 GHz
Communication energy consumption rate cr	0.2 Watt
Transient faults rate per ms of $core_j$ execution with the maximum frequency	[0.000001, 0.000009]
Sensitivity fault rate d_j to frequency scaling of $core_j$	[1.0, 3.0]
WCET of task t_i assigned to $core_j$	[10, 100] ms
Communication time between t_i and t_j	[1, 10] ms
Execution results transmission (for voting) time $rtt_{i,k,j}$	the maximum communication time between task t_i and its direct successor
Reliability requirement of the application G	at least exceeds 0.99

fault detection overhead in AFTSA and the result comparison overhead in our experiments.

According to the overhead of fault detection for some benchmark applications in [26], the average time overhead of light fault detection (LFD) and heavy fault detection (HFD) mechanism is about 28% and 85% of the application execution time, respectively. To obtain the time overhead for comparing execution results (CER) of different copies of a task, we measured the execution time and result comparison time for some benchmarks by using the $clock()$ function in C++. Table 9 shows that the comparison overhead of most applications is less than 3%. Therefore, the overhead of LFD, HFD, and CER is set to 28%, 85%, and 3% of the task execution time, respectively. In addition, in order to reduce the energy consumption of AFTSA, we have made the following improvements.

(1) AFTSA improves reliability by increasing the number of copies of tasks. When the application's reliability value reaches the requirements, we will not increase the number of copies of any task.

(2) When a copy of a task is completed and no fault is found after fault detection, the execution of other copies of this task is also terminated immediately.

(3) If a task has only one copy, it will not be fault detected.

In experiments, AFTSA with LFD is called AFTSA-LFD and with HFD is called AFTSA-HFD.

Gaussian elimination and Fourier transform are used as the benchmark, which has been widely used in the field of parallel computing to evaluate the performance of algorithms [17][35][36][37]. To describe the number of tasks in applications, a parameter ρ is introduced. For Gaussian elimination applications, the total number of tasks is $|T| = \frac{\rho^2+\rho-2}{2}$. Fig. 4 shows a Gaussian elimination application with $\rho = 5$. For Fourier transform applications, the total number of tasks is $|T| = (2 \times \rho - 1) + \rho \times \log_2{}^{\rho}$ with $\rho = 2^n$, where n is a positive integer. Fig. 5 shows a Fourier transform application with $\rho = 4$.

5.2. Different deadline constraints of applications

Experiment 1: The small-scale, medium-scale, and large-scale Gaussian applications are used to evaluate different algorithms. The value ρ of these three sizes of applications is 16, 32, and 48, and the corresponding task numbers of which are 135, 527, and 1175 respectively. The reliability requirement of the application is 0.995 and the values of slack ratio SR increase from 1.1 to 2.0 with 0.1 increments. Figs. 6 and 7 show the energy consumptions and actual reliabilities generated by different algorithms.

As shown in Fig. 6, as the increase of slack ratio SR, the energy consumptions generated by IHEFT, AFTSA-LFD, and AFTSA-HFD remain

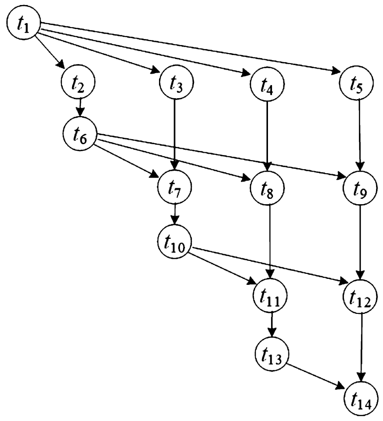


Fig. 4. A Gaussian elimination application with $\rho = 5$.

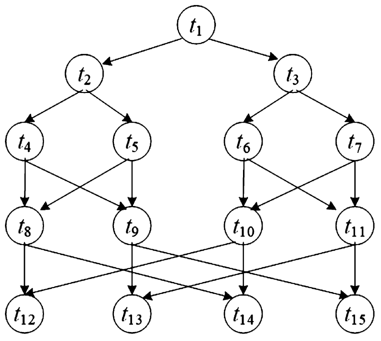


Fig. 5. A Fourier transform application with $\rho = 4$.

unchanged, and the energy consumption generated by MEOTC and OEM algorithms is significantly reduced. When $SR \geq 1.3$ the energy consumption generated by EET, MEOTC, and OEM is lower than that of AFTSA-LFD and AFTSA-HFD. Among the three algorithms EET, MEOTC, and OEM, OEM generates the lowest energy consumption and EET generates the highest energy consumption. In addition, when $SR \geq 1.3$, the energy consumption generated by EET does not significantly change as SR increases. Another interesting phenomenon in this experiment is that the energy consumption of IHEFT in Fig. 6 (a) is higher than that of AFTSA-HFD, while the energy consumption of the IHEFT algorithm in Fig. 6 (b) and (c) is lower than that of AFTSA-HFD. On average, the energy consumption generated by MEOTC and OEM is 79.0% and 68.9% of AFTSA-LFD, and 56.8% and 49.5% of AFTSA-HFD.

Fig. 7 shows the reliability generated by different algorithms, where the reliability requirements are satisfied by all algorithms. IHEFT generates the highest reliability because it does not reduce the execution frequency. AFTSA-LFD and AFTSA-HFD generate almost the same reliability, and they are slightly higher than the reliability requirement. However, the reliabilities generated by EET, MEOTC, and OEM are significantly higher than the reliability requirement when $SR < 1.3$. There is no significant difference in the reliability generated by all algorithms except IHEFT when $SR > 1.4$.

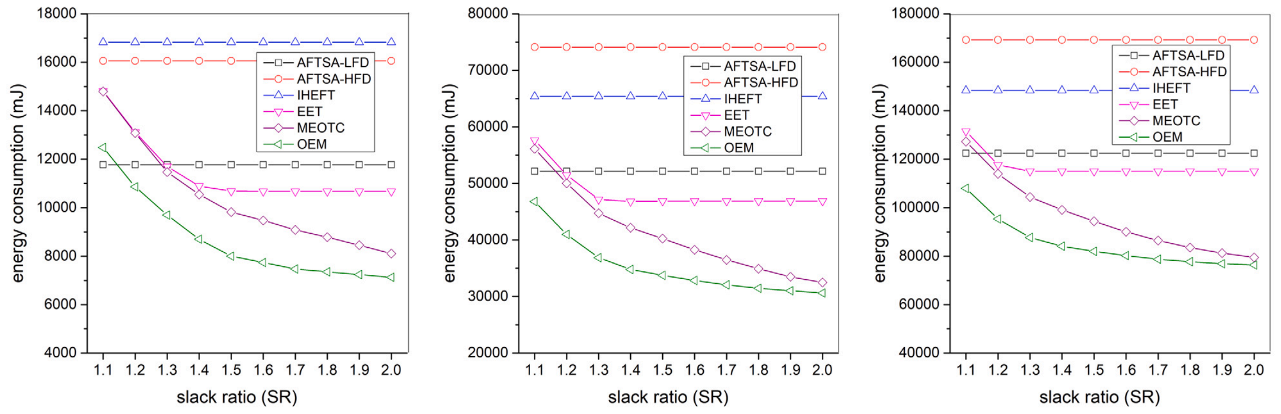
The main reasons for these results are as follows.

(1) When the reliability requirements of the application are satisfied, AFTSA-LFD and AFTSA-HFD will not increase the number of copies of any task, so the energy consumption generated by these two algorithms will not change with SR.

(2) When SR is greater than 1.0, there will be a slack time that can be used to reduce the execution frequency of the copies. EET reduces the execution frequency as much as possible while meeting the deadline and satisfying the reliability requirement of the application. MEOTC can also reduce the execution frequency and terminate the third copy in advance, so MEOTC can also effectively reduce energy consumption.

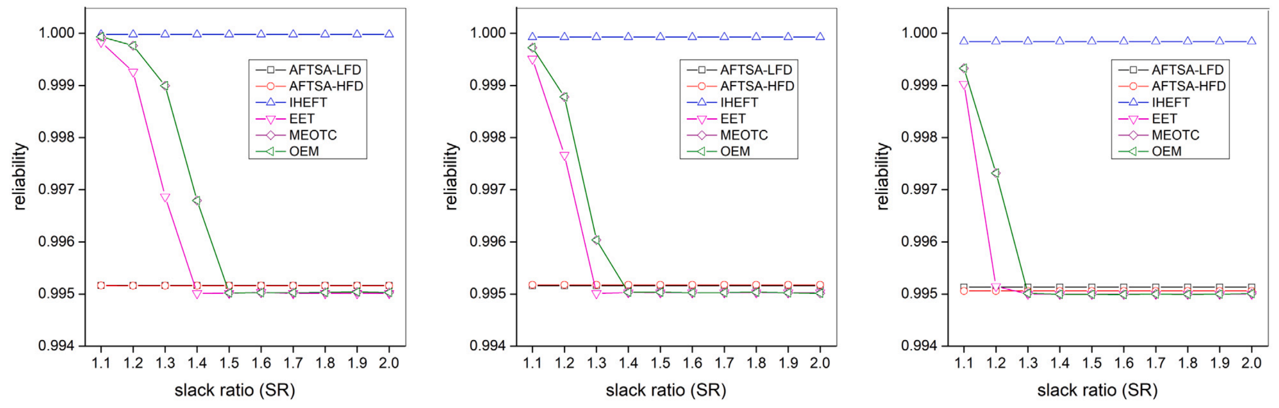
Table 9
Overhead of comparing execution results.

Benchmark	Execution Time (ms)	Comparison Time (ms)	Overhead (%)	Parameter Description
Qsort	90	2	2.2%	100000 elements of string type
Qsort	198	2	1.0%	1000000 elements of float type
Matrix Multiple	382	1	0.3%	500×500 elements of integer type
SusanCorners	192	2	1.0%	image size 640*480
SusanEdges	504	2	0.4%	image size 640*480



(a) The Gaussian application with $\rho = 16$ (b) The Gaussian application with $\rho = 32$ (c) The Gaussian application with $\rho = 48$

Fig. 6. The energy consumption generated by different algorithms.



(a) The Gaussian application with $\rho = 16$ (b) The Gaussian application with $\rho = 32$ (c) The Gaussian application with $\rho = 48$

Fig. 7. The actual reliability generated by different algorithms.

(3) EET tends to reduce the execution frequency of the copies, while MEOTC tends to delay the start time of the third copy and also reduces the execution frequency. When there are no transient faults in the first two copies of the task, MEOTC will terminate the execution of the third copy. Therefore, the energy consumption generated by MEOTC is lower than that of EET.

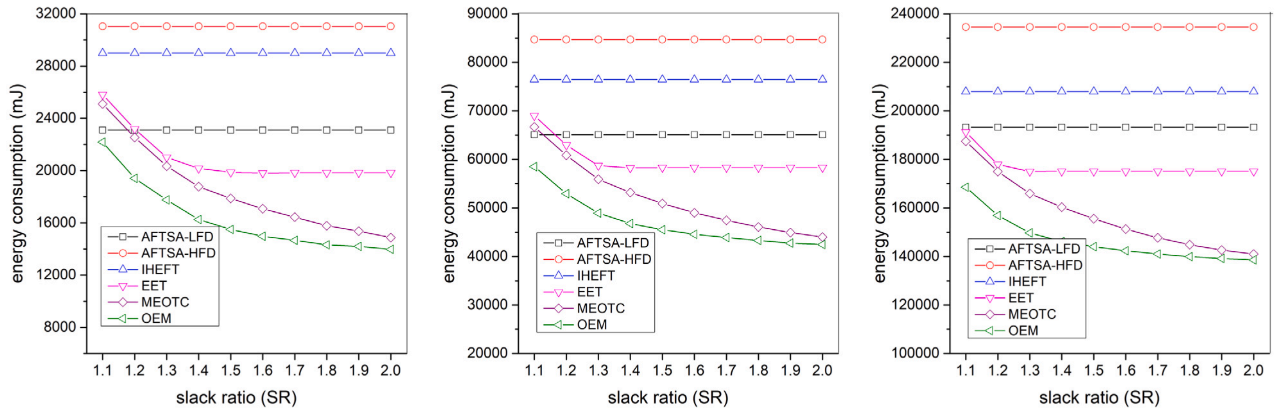
(4) When a task is completed ahead of schedule, OEM immediately executes the first two copies of the subsequent task, so that the third copy will be terminated earlier or the third copy does not need to be executed. Therefore, OEM consumes less energy than MEOTC.

(5) Due to the high fault detection overhead of the AFTSA-LFD and AFTSA-HFD algorithms and the fact that they do not use DVFS, these two algorithms generate high energy consumption.

(6) In IHEFT, even though each task has three copies to be executed, they have lower voting overhead. In AFTSA-HFD, the total number of copies of the task is less than IHEFT, but this algorithm has high fault detection overhead and the number of copies is related to the scale and reliability requirement of the application. For example, in Fig. 7 (a), the total number of tasks is 135, and there are 102 tasks that need to be replicated. The total number of tasks in Fig. 7 (b) is 527, and 499 tasks need to be replicated to satisfy the reliability requirement. Therefore, when the reliability requirement remains unchanged, as the application scale increases, the energy consumption generated by AFTSA-HFD increases more than that of IHEFT.

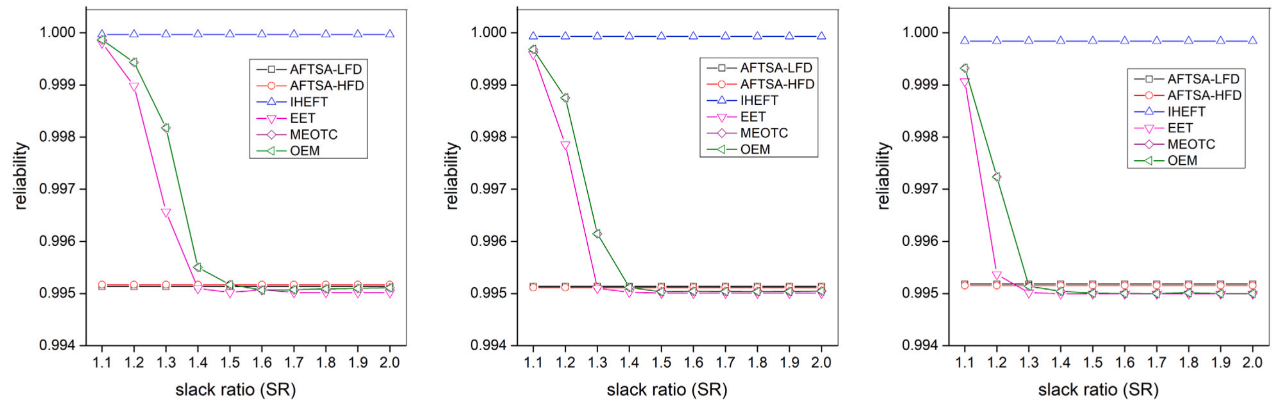
(7) When the execution frequency is reduced, the reliability of the application will also be reduced. However, due to reliability constraints, although there is enough relaxation time, the execution frequency cannot be reduced indefinitely. Therefore, the reliability change trend generated by each algorithm is not obvious when $SR > 1.3$.

Experiment 2: The small-scale, medium-scale, and large-scale Fourier applications are used to evaluate different algorithms. The value ρ of these three sizes of applications is 32, 64, and 128, and the corresponding task numbers of which are 223, 511, and 1151 respectively. The reliability requirement of the application is 0.995 and the values of slack ratio SR increase from 1.0 to 2.0 with 0.1 increments. Figs. 8



(a) The Fourier application with $\rho = 32$ (b) The Fourier application with $\rho = 64$ (c) The Fourier application with $\rho = 128$

Fig. 8. The energy consumption generated by different algorithms.



(a) The Fourier application with $\rho = 32$ (b) The Fourier application with $\rho = 64$ (c) The Fourier application with $\rho = 128$

Fig. 9. The actual reliability generated by different algorithms.

and 9 show the energy consumptions and actual reliabilities generated by different algorithms.

As shown in Fig. 8, the energy consumptions generated by EET, MEOTC, and OEM are lower than that of AFTSA-HFD and AFTSA-LFD when $SR \geq 1.2$. Among the three algorithms of EET, MEOTC, and OEM, OEM generates the lowest energy consumption. On average, the energy consumption generated by MEOTC and OEM is 80.8% and 74.6% of AFTSA-LFD, and 64.9% and 59.9% of AFTSA-HFD.

As shown in Fig. 9, all algorithms can make the application satisfy the reliability requirement. IHEFT generates the highest reliability, the reliability generated by the AFTSA-HFD and AFTSA-LFD remains unchanged and slightly higher than the reliability requirements. The reliability generated by EET, MEOTC, and OEM is significantly higher than the reliability requirement when $SR < 1.4$.

5.3. Different reliability requirements of applications

Experiment 3: The medium-scale Gaussian applications with $\rho = 32$ (527 tasks) are used to evaluate different algorithms. The value of slack ratio $SR = 1.5$ and the reliability requirements of the application increase from 0.990 to 0.999 with 0.001 increments. Fig. 10 shows the results of different algorithms with different reliability requirements.

As shown in Fig. 10 (a), AFTSA-HFD generates the highest energy consumption, while OEM generates the lowest energy consumption. The energy consumption generated by MEOTC is higher than that of OEM but lower than that of other algorithms. With the improvement of reliability requirements, the energy consumption generated by IHEFT remains unchanged, and the energy consumption generated by the other five algorithms is gradually increased. The main reason for this result is that with the improvement of reliability requirements, EET and MEOTC will run tasks at a higher frequency and AFTSA needs to assign more task copies. Therefore, the energy consumption of these algorithms will increase.

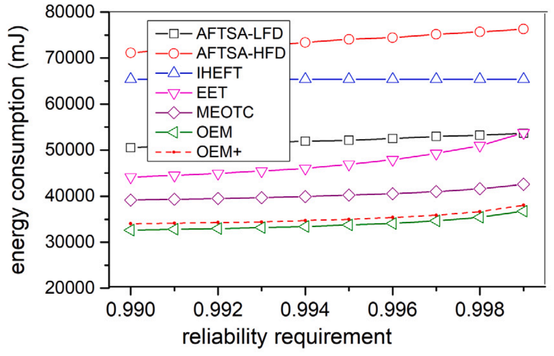
It should be noted that in Fig. 10 (a), a diagram OEM+ is added to indicate the energy consumption generated by OEM when the time overhead for comparing execution results of different copies of a task increases from 3% to 10%. At this time, the energy consumption generated by OEM is still the lowest.

As shown in Fig. 10 (b), all algorithms can make the application satisfy the reliability requirements.

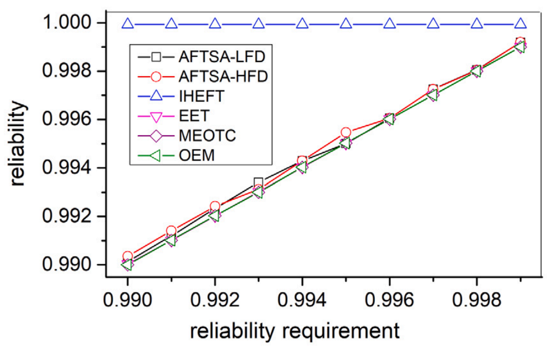
5.4. Different platform scales

Experiment 4: This experiment evaluates the energy consumption and reliability of each algorithm on different platforms. The medium-scale Fourier application with $\rho = 64$ (511 tasks) is used in this experiment, the reliability requirement of which is 0.995 and slack ratio $SR = 1.5$. There are four different types of cores within the platform, which are divided into four groups and each group has the same core type. When the number of cores in each group is 4, 8, 16, and 32 (i.e., the total number of cores is 16, 32, 64, and 128 respectively), the energy consumption and actual reliability generated by each algorithm are shown in Fig. 11.

As shown in Fig. 11 (a), as the total number of core increases, the energy consumption generated by all algorithms will decrease. The main reason for this result is that as the number of cores in each group

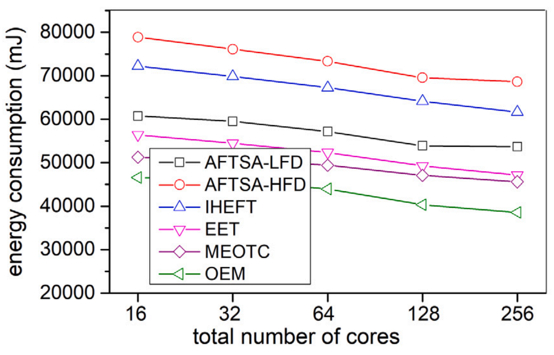


(a) Energy consumption

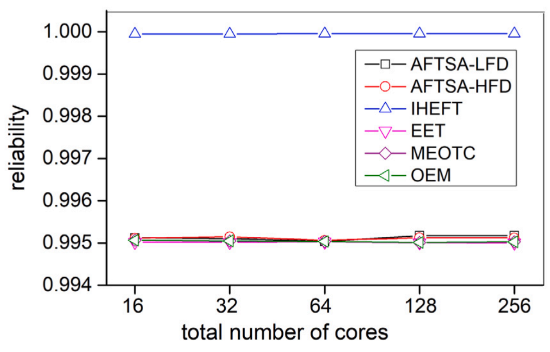


(b) Actual reliability

Fig. 10. Results of different algorithms with different reliability requirements.



(a) Energy consumption



(b) Actual reliability

Fig. 11. Results of different algorithms with different numbers of cores.

increases, the communication energy consumption between tasks will decrease. Overall, AFTSA-HFD generates the highest energy consumption, while OEM generates the lowest energy consumption.

The results of the above four experiments are summarized as follows.

- The TMR technique can also be applied to heterogeneous multi-core real-time systems.
- When the cores support DVFS, with the application's deadline increasing, the energy consumption generated by the algorithms EET, MEOTC, and OEM will be significantly reduced.
- In cases where the deadline of the application is relatively relaxed, such as when $SR > 1.3$, MEOTC exhibits obvious advantages over both AFTSA-LFD and AFTSA-HFD. To ensure that the deadline constraint and reliability requirement of the application are met, the OEM algorithm based on MEOTC can be used in the actual environment. On average, the energy consumption generated by MEOTC and OEM is 80% and 72% of that of AFTSA-LFD, and 61% and 55% of that of AFTSA-HFD.

6. Conclusion

TMR technique can perfectly tolerate the faults in task execution, but the system energy consumption will increase dramatically due to multiple copies of execution. This paper studies the problem of minimizing energy consumption for parallel applications on heterogeneous multi-core real-time systems with TMR. First, the IHEFT algorithm is designed to assign three copies of a task to different cores, and then according to the given application's deadline, the EET algorithm is designed to extend the execution time of the tasks, so as to reduce the execution frequency of copies. Based on the task assignment information of IHEFT, the MEOTC algorithm is designed. Finally, considering the actual scenarios during task execution, an online energy management method is proposed. Simulation results show that OEM based on MEOTC is more energy efficient than the original TMR method and the existing task replication algorithm AFTSA in most cases. We believe that the proposed algorithms can be applied to heterogeneous multi-core real-time systems with the TMR technique. In future work, we will discuss how to minimize energy consumption under the TMR mechanism in clusters where multiple cores share the same execution frequency while ensuring both the deadline and the reliability requirement of applications are met.

CRediT authorship contribution statement

Hongzhi Xu: Data curation, Formal analysis, Investigation, Methodology, Writing – original draft, Software. **Binlian Zhang:** Data curation, Investigation, Validation. **Chen Pan:** Investigation, Methodology, Validation, Writing – review & editing. **Keqin Li:** Supervision, Writing – review & editing.

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.

Acknowledgment

The authors are sincerely grateful to all the anonymous reviewers, whose constructive comments have significantly improved the quality of this paper. This research was partially funded by the National Natural Science Foundation of China under Grant No. 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.

[2] F. Baharvand, S.G. Miremadi, Lexact: low energy n-modular redundancy using approximate computing for real-time multicore processors, IEEE Trans. Emerg. Top. Comput. 8 (2) (2020) 431–441, https://doi.org/10.1109/TETC.2017.2737045.

[3] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, N. Guan, Energy-efficient real-time scheduling of dags on clustered multi-core platforms, in: 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, 2019, pp. 156–168.

[4] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, W. Liu, Response time bounds for typed dag parallel tasks on heterogeneous multi-cores, IEEE Trans. Parallel Distrib. Syst. 30 (11) (2019) 2567–2581.

[5] 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.

[6] 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.

[7] 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.

[8] J. Huang, R. Li, J. An, H. Zeng, W. Chang, A dvfs-weakly dependent energy-efficient scheduling approach for deadline-constrained parallel applications on heterogeneous systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 40 (12) (2021) 2481–2494, https://doi.org/10.1109/TCAD.2021.3049688.

[9] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, X. Huang, Enhanced energy-efficient scheduling for parallel applications in cloud, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), IEEE Computer Society, 2012, pp. 781–786.

[10] 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.

[11] K. Li, Optimal power and performance management for heterogeneous and arbitrary cloud servers, IEEE Access 7 (2019) 5071–5084, https://doi.org/10.1109/ACCESS.2018.2889220.

[12] 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.

[13] D. Liu, J. Spasic, G. Chen, T. Stefanov, Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs, in: 2015 13th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia), 2015, pp. 1–10.

[14] 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.

[15] J. McNamee, V. Pan, Chapter 12 - low-degree polynomials, in: J. McNamee, V. Pan (Eds.), Numerical Methods for Roots of Polynomials - Part II, in: Studies in Computational Mathematics, vol. 16, Elsevier, 2013, pp. 527–556.

[16] F. Mireshghallah, M. Bakhshalipour, M. Sadrosadati, H. Sarbazi-Azad, Energy-efficient permanent fault tolerance in hard real-time systems, IEEE Trans. Comput. 68 (10) (2019) 1539–1545, https://doi.org/10.1109/TC.2019.2912164.

[17] T. Mladenov, S. Nooshabadi, K. Kim, Implementation and evaluation of raptor codes on embedded systems, IEEE Trans. Comput. 60 (12) (2011) 1678–1691.

[18] S. Moulik, Reset: a real-time scheduler for energy and temperature aware heterogeneous multi-core systems, Integration 77 (2021) 59–69.

[19] S. Moulik, R. Devaraj, A. Sarkar, A. Shaw, A deadline-partition oriented heterogeneous multi-core scheduler for periodic tasks, in: 2017 18Th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), IEEE, 2017, pp. 204–210.

[20] S. Moulik, R. Chaudhary, Z. Das, A. Sarkar, Ea-hrt: an energy-aware scheduler for heterogeneous real-time systems, in: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, 2020, pp. 500–505.

[21] S. Moulik, Z. Das, R. Devaraj, S. Chakraborty, Seamers: a semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems, J. Syst. Archit. 114 (2021) 101953.

[22] L. Niu, D. Zhu, Reliability-aware scheduling for reducing system-wide energy consumption for weakly hard real-time systems, J. Syst. Archit. 78 (2017) 30–54.

[23] S. Paul, N. Chatterjee, P. Ghosal, J.-P. Diguet, Adaptive task allocation and scheduling on noc-based multicore platforms with multitasking processors, ACM Trans. Embed. Comput. Syst. 20 (1) (2020) 1–26.

[24] A. Roy, H. Aydin, D. Zhu, Energy-efficient fault tolerance for real-time tasks with precedence constraints on heterogeneous multicore systems, in: 2019 Tenth International Green and Sustainable Computing Conference (IGSC), IEEE, 2019, pp. 1–8.

[25] M. Salehi, M.K. Tavana, S. Rehman, F. Kriebel, M. Shafique, A. Ejlali, J. Henkel, Drvs: power-efficient reliability management through dynamic redundancy and voltage scaling under variations, in: 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), IEEE, 2015, pp. 225–230.

[26] 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, https://doi.org/10.1109/TPDS.2015.2444402.

[27] 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, 2022, pp. 501–509.

[28] 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.

[29] Y. Sharma, S. Chakraborty, S. Moulik, Eta-hp: an energy and temperature-aware real-time scheduler for heterogeneous platforms, J. Supercomput. 78 (8) (2022) 1–25.

[30] A. Simevski, R. Kraemer, M. Krstic, Investigating core-level n-modular redundancy in multiprocessors, in: 2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, 2014, pp. 175–180.

[31] 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.

[32] 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.

[33] S. Wang, K. Li, J. Mei, G. Xiao, K. Li, A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems, J. Grid Comput. 15 (1) (2017) 23–39.

[34] G. Xie, R. Li, K. Li, Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems, J. Parallel Distrib. Comput. 83 (2015) 1–12.

[35] 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.

[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] 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, https://doi.org/10.1109/TPDS.2017.2730876.

[38] 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 (03) (2018) 167–181, https://doi.org/10.1109/TSUSC.2017.2711362.

[39] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, K. Li, Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems, IEEE Trans. Serv. Comput. 13 (05) (2020) 871–886, https://doi.org/10.1109/TSC.2017.2665552.

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

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

[42] Y. Yang, W. Diao, An energy-efficient frame-based task scheduling algorithm for heterogeneous multi-core soc in iot devices, in: 2020 International Wireless Communications and Mobile Computing (IWCMC), 2020, pp. 1404–1409.

[43] 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, Inf. Sci. 319 (2015) 113–131.

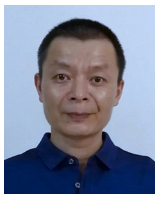
[44] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, Inf. Sci. 379 (2017) 241–256, https://doi.org/10.1016/j.ins.2016.08.003.

[45] L. Zhao, Y. Ren, Y. Xiang, K. Sakurai, Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems, in: High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on, IEEE, 2010, pp. 434–441.

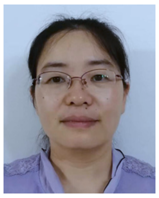
[46] L. Zhao, Y. Ren, K. Sakurai, Reliable workflow scheduling with less resource redundancy, Parallel Comput. 39 (10) (2013) 567–585.

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

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



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 PhD 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.

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).