Journal of Parallel and Distributed Computing 127 (2019) 44–57

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Minimizing energy consumption with reliability goal on heterogeneous embedded systems

Hongzhi Xu [a,b,*], Renfa Li [a], Chen Pan [c], Keqin Li [a,d]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[b] College of Software, Jishou University, Zhangjiajie 427000, China
[c] Swanson School of Engineering, University of Pittsburgh, Pittsburgh, PA 15216, USA
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

HIGHLIGHTS

- The reliability goal of the application is transformed to that of each task.
- The reliability goal transformation methods for non-DVFS and DVFS are both proposed.
- Two energy-efficient scheduling algorithms with the reliability goal are designed.

ARTICLE INFO

Article history:
Received 7 February 2018
Received in revised form 21 November 2018
Accepted 8 January 2019
Available online 17 January 2019

Keywords:
DAG-based parallel application
DVFS technique
Energy-efficient
Heterogeneous embedded system
Reliability goal

ABSTRACT

The embedded systems generally require to be low-powered and highly reliable. In order to achieve the low-power design goal, dynamic voltage frequency scaling (DVFS) technique has been widely employed in various embedded application scenarios. However, DVFS reduces execution frequency, which increases transient faults of the processor dramatically. As a result, the reliability of the application will be severely reduced. In this paper, we aim at minimizing energy consumption with reliability goal for parallel application on heterogeneous embedded systems. Since the reliability of the application is the product of the reliability of all the tasks that belong to the application, the reliability goal of the application is transformed into the reliability goal of each task. Considering that some systems may not support DVFS techniques, two methods are proposed to transform the reliability goal of the application into each task for non-DVFS and DVFS, respectively. Based on the reliability goal transformation methods, two energy-efficient scheduling algorithms with the reliability goal are designed. Experiments with real parallel applications demonstrate that the proposed algorithms have significant improvements in energy efficiency compared with the state-of-the-art algorithms.

1. Introduction

1.1. Background

With the rapid development of computer hardware technology, the cost of embedded system becomes much lower and its computing performance has been greatly improved. Therefore, embedded systems have been applied to many fields, such as aerospace, intelligent transportation, smart grid, smart home, medical care, and health monitoring of large buildings. However, the high performance of processors will also bring high energy consumption. Hence, energy management is important to the performance of embedded systems, especially for battery powered embedded systems [14,33]. In order to improve energy efficiency of embedded system, many techniques, such as dynamic power management (DPM) and dynamic voltage frequency scaling (DVFS) , have been proposed to reduce the system energy consumption by scaling the voltage and frequency in runtime [4,9–12,15,26]. However, as the frequency of the processor reduces, the probability of failure will increase. As a result, the reliability of the application will be weakened [13,34,35]. For many embedded systems, reliability is an important quality targets of the application, especially in the safety-critical real-time systems, the reliability goal of the application should be satisfied.

1.2. Motivation

Since DVFS technique may cause unreliable issues to embedded system, many researches have focused on improving energy

* Corresponding author at: College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China.
E-mail addresses: xuhongzhi9@163.com (H. Xu), lirenfa@vip.sina.com (R. Li), chen.pan@pitt.edu (C. Pan), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.jpdc.2019.01.006
0743-7315/

efficiency while guaranteeing the system reliability [13,27,34,35]. Most of these studies are only targeting on a single processor with independent tasks. In recent years, many heterogeneous embedded real-time systems have emerged. These systems use high-speed networks to interconnect different processors to execute parallel applications. From the system design perspective, parallel applications are composed of tasks with precedence-constrained, which can be represented by directed acyclic graph (DAG) [1,7,19,25]. There are some studies focus on the reliability goals of the heterogeneous systems with the DAG-based parallel application. In [31,32], the authors proposed MaxRe and RR algorithms to satisfy the reliability goal with the minimum resource consumption. In [22], MRCRG algorithm is proposed to minimize resource consumption while satisfying the reliability goal of the application. Energy is also a system resource, and these studies can also be used to improve energy efficiency. However, the above studies did not discuss the use of DVFS technique to improve energy efficiency of the system. In [23], the ESRG algorithm with DVFS technique is proposed to reduce energy consumption while satisfying the reliability goal of the application. However, without using replication technique, the ESRG algorithm cannot always satisfy the reliability goal of the applications. Therefore, this paper studies the minimization of system energy consumption while satisfying the reliability goal of a DAG-based parallel application on heterogeneous embedded system.

1.3. Main contributions

The life cycle of system development usually includes phases of analysis, design, implementation and testing [23]. This paper focuses on the design phase of the system, the main contributions are as follows.

(1) The reliability goal of the DAG-based parallel application is transformed into the reliability goal of each task. Two methods for predetermining the reliability goal of the task are proposed for non-DVFS and DVFS, respectively. When the reliability goal is less than or equal to maximum reliability that the application can be reached, all tasks can be assigned to the appropriate processors so that the reliability goal of the application can be satisfied.

(2) Based on the above reliability goal transformation methods, two energy-efficient scheduling algorithms are proposed to minimize energy consumption for DAG-based parallel application on heterogeneous embedded system. One is designed without using a DVFS technique and the other is designed with a DVFS technique.

(3) Experiments are conducted under different scenarios using real parallel applications. The experimental results show that the proposed algorithms have significant improvements in energy efficiency compared with the state-of-the-art algorithms.

The rest of the paper is organized as follows. Section 2 reviews related research. Section 3 introduces the models that are used in this paper. Section 4 presents related preliminaries. Section 5 presents detailed algorithms of minimizing energy consumption while satisfying the reliability goal requirements of the applications. Finally, Sections 6 and 7 discuss our simulation results and conclusions.

2. Related work

At present, there are many studies focus on improving energy efficiency while at the same time guaranteed the system reliability. For a single processor, Zhu et al. [35] proposed the reliability-aware power management for real-time periodic tasks, which lowers down the execution efficiency in the slack time. Lin et al. [13] presented the shared-recovery dynamic algorithm to guarantee the reliability of the system. Based on shared-recovery technique, Zhao et al. [30] presented the SHR algorithm to minimize energy consumption of the system. Fan et al. [5] presented reliability aware power management algorithm to minimize the energy consumption for single processor real-time systems. In [29], Zhang et al. presented the DLPSR algorithm for periodic tasks with shared resources, the slack time is reclaimed to save energy while preserving the system reliability.

In recent years, there are a lot of research works focuses on the reliability of multi-processor systems (or multi-core systems). In [21], a hybrid design-time/run-time framework is proposed for resource allocation on multicore embedded systems, which considers execution time, transient faults, and permanent faults due to aging effects. Haque et al. [6] solved the problem of achieving a given reliability target for a set of periodic tasks running on a DVS-enabled multicore system with minimum energy consumption. In [17], a reliability-aware scheduling is presented to maximize system reliability, which dynamically schedules applications to the different types of cores in heterogeneous multicore systems.

For parallel applications on heterogeneous distributed systems, Tang et al. [18] proposed a reliability-aware scheduling algorithm for DAG-based application. In [3], Assayad et al. presented a tri-criteria scheduling for data-flow graphs of operations onto parallel heterogeneous systems, which uses the active replication to improve the system reliability. Zhang et al. [28] presented reliability maximization with energy constraint algorithm for DAG-based application on heterogeneous systems, which balanced the tradeoff between reliability and energy consumption. In order to satisfy specific reliability goal of the applications, Zhao et al. [32] proposed the MaxRe algorithm to minimize system resources consumption, in which the reliability goal of the application is transformed to the reliability goal of each task. In [31], Zhao et al. also proposed the RR algorithm which consumes less resources to satisfy the reliability goal of the applications. In [22], the MRCRG algorithm is proposed for parallel application on heterogeneous embedded systems, which minimizing system resource consumption while satisfying the reliability goal of the application. To improve the energy efficiency of the heterogeneous embedded systems, Xie et al. [23] proposed the ESRG algorithm to reduce energy consumption while satisfying the reliability goal of the application.

3. System model and problem formulation

3.1. Power model

Similar to [13,34,35], the energy consumption of CMOS chip at frequency f is given by

$$P(f) = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m). \quad (1)$$

In (1), P_s is the static power, which is used to maintain the basic circuit and the clock. P_{ind} is the leakage power unrelated to the frequency, which is a constant and can be eliminated when the processor sleeps. P_d is the dynamic power caused by charge/discharge of gate circuit, which is related to the processor frequency. C_{ef} represents the switching capacitance, and m represents the dynamic energy exponent. When the processor is active, $\hbar = 1$; otherwise, $\hbar = 0$.

According to [13,34,35], the minimum energy-efficient frequency is given by

$$f_{mee} = \sqrt[m]{\frac{P_{ind}}{C_{ef}(m-1)}}. \quad (2)$$

When the execution frequency of the processor is less than f_{mee}, the energy consumption would be more because of the longer execution time. Assuming that the available frequency of the processor is minimum frequency f_{min} to maximum frequency f_{max}, to improve

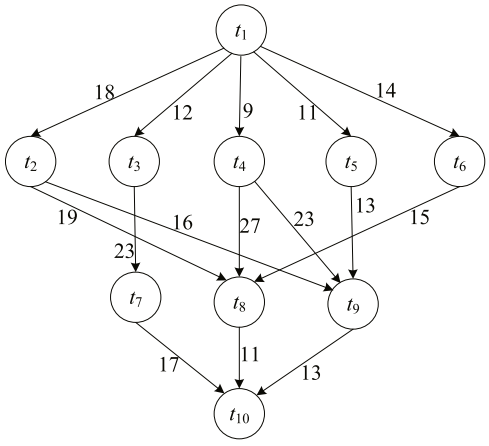


Fig. 1. Motivation example of a DAG-based parallel application.

energy efficiency, the lowest available frequency of the processor should be

$$f_{\text{low}} = \max(f_{\text{mee}}, f_{\min}). \tag{3}$$

This paper mainly studies the DAG-based parallel application execution on heterogeneous embedded systems, the processor is represented by a set as $PR = \{pr_1, pr_2, \ldots, pr_M\}$, where M is the number of processors. Assuming that different processors have different computing performance, and all processors are connected through the network. Considering that the processor's execution frequency is discrete, for processor pr_j, the available frequency set is represented as $\{f_{j,\text{low}}, f_{j,a}, f_{j,b}, \ldots, f_{j,\max}\}$.

3.2. Application model

Based on [2,20], the DAG-based parallel application is represented as DAG $G = (T, C)$, where T and C are described as follows.

$T = \{t_1, t_2, \ldots, t_N\}$ is a node set in application G, which represents task sets. C is an edge set in G, $c_{i,j} \in C$ indicates that there is a precedence constraint between t_i and t_j, t_j can be executed after the completion of t_i. The value of $c_{i,j}$ indicates the worst case communication time (WCCT) between t_i and t_j. If t_i and t_j are assigned on the same processor, then the WCCT is 0. A classic DAG-based parallel application is shown in Fig. 1 [20,22,24,25], $c_{1,2} = 18$ indicates the WCCT between t_2 and t_1. If the processor that executes t_2 is different from the processor that executes the t_1, the WCCT is 18. If t_2 and t_1 are executed on the same processor, the WCCT is 0. To describe the successively relationship between the tasks, $parent(t_i)$ is defined as the immediate predecessor task set of t_i, and $child(t_i)$ is defined as the immediate successor task set of t_i. The task without predecessor task is called t_{entry}, and the task without successor task is called t_{exit}. If there are multiple t_{entry} (or t_{exit}) in G, then the dummy task of entry (or exit) is constructed to G. For heterogeneous embedded systems, the execution time of the same task on different processors is different, the matrix W as $N \times M$ is defined to represent the worst case execution time (WCET) of each task on the different processors. $w_{i,j}$ is the WCET of task t_i that executes on processor pr_j with maximum execution frequency $f_{j,\max}$. The WCETs of the tasks in Fig. 1 on the three processors are shown in Table 1, where the WCETs of t_1 on three processors are 14, 16 and 9, respectively.

This paper focuses on the design phase of the system, assuming that all the WCCTs and WCETs are known through the WCCT and WCET analysis methods in the analysis phase [22,23].

Table 1
WCET of tasks on different processors of the motivation example.

task	pr_1	pr_2	pr_3	*Rank*
t_1	14	16	9	108
t_2	13	19	18	77
t_3	11	13	19	80
t_4	13	8	17	80
t_5	12	13	10	69
t_6	13	16	9	63.3
t_7	7	15	11	42.7
t_8	5	11	14	35.7
t_9	18	12	20	44.3
t_{10}	21	7	16	14.7

3.3. Energy consumption

Because different processors have different energy consumption parameter, let $E_{\text{dp}}(t_i, pr_j, f_{j,k})$ represent the dynamic energy consumption of task t_i on processor pr_j at frequency $f_{j,k}$, which can be calculated by

$$E_{\text{dp}}(t_i, pr_j, f_{j,k}) = (P_{j,\text{ind}} + C_{j,\text{ef}} f^{m_j}) \times w_{i,j} \times \frac{f_{j,\max}}{f_{j,k}}. \tag{4}$$

Considering that tasks t_i and $parent(t_i)$ may be executed on different processors, there is communication energy consumption when the tasks are communicated between two different processors. In this paper, the communication energy consumption is proportional to the communication time, and the energy consumption rate per time unit of communication is defined as ecr [22]. Therefore, if task t_i is assigned to processor pr_j, the energy consumption of communication can be calculated by

$$E_{\text{com}}(t_i, pr_j) = \sum_{t_x \in parent(t_i)} ecr \times c'_{x,i}. \tag{5}$$

In Eq. (5), if task t_i and t_x are assigned to the same processor then $c'_{x,i} = 0$; otherwise $c'_{x,i} = c_{x,i}$.

Based on Eqs. (4) and (5), when task t_i is assigned to the processor, the total dynamic energy consumption of the processor and communication is given by

$$E_{\text{d}}(t_i) = E_{\text{dp}}(t_i, pr_{asp(t_i)}, f_{asp(t_i),asf(t_i)}) + E_{\text{com}}(t_i, pr_{asp(t_i)}), \tag{6}$$

where $pr_{asp(t_i)}$ represents a processor which is assigned to task t_i and $f_{asp(t_i),asf(t_i)}$ represents the execution frequency of processor $pr_{asp(t_i)}$.

When all the tasks in application G are assigned, the total amount of the dynamic energy consumption is given by

$$E_{\text{d}}(G) = \sum_{i=1}^{N} E_{\text{d}}(t_i). \tag{7}$$

Let $E_{\text{s}}(G)$ represent the static energy consumption of the application, the static energy consumption is derived from the processor and network. Because the static energy consumption always exists, for the sake of simplicity, the static energy consumption of network is incorporated into the static energy consumption of the processor. Therefore, the static energy consumption of the application can be calculated by

$$E_{\text{s}}(G) = \sum_{j=1}^{M} P_{j,\text{s}} \times SL(G), \tag{8}$$

where $SL(G)$ is scheduling length. Based on Eqs. (7) and (8), the total energy consumption of the system is the sum of $E_{\text{d}}(G)$ and $E_{\text{s}}(G)$, namely

$$E(G) = E_{\text{d}}(G) + E_{\text{s}}(G). \tag{9}$$

3.4. Reliability model

The runtime faults can be divided into transient faults and permanent faults, while the transient faults happen more frequently [34,35]. Similar to [13,22,34,35], only transient faults are considered in this paper. Assume that the transient fault follows Poisson distribution [13,22,34,35], the reliability of execution a task during the duration t is denoted by $R(t) = e^{-\lambda t}$, where λ is the transient fault rate per time unit of processor. In this study, λ_j is defined as the transient fault rate per time unit of processor pr_j. If task t_i is executed on processor pr_j with maximum frequency, the reliability of task t_i can be calculated by

$$R(t_i, pr_j, f_{j,\max}) = e^{-\lambda_j \times w_{i,j}}. \tag{10}$$

According to [13,28,34,35], in a DVFS-enabled system, the transient fault rate of processor with frequency f is given by

$$\lambda(f) = \lambda_0 10^{\frac{d(f_{\max}-f)}{f_{\max}-f_{\min}}}, \tag{11}$$

where d is a constant greater than 0, which represents the sensitivity fault rate of voltage/frequency scaling, and λ_0 is the transient fault rate with the maximum frequency. For heterogeneous embedded systems, we let $\lambda_{j,\max}$ represent the transient fault rate of processor pr_j with the maximum frequency, and d_j represents the sensitivity fault rate of voltage/frequency scaling of processor pr_j, then the transient fault rate $\lambda_{j,k}$ of processor pr_j with frequency $f_{j,k}$ is given by

$$\lambda_{j,k} = \lambda_{j,\max} \times 10^{\frac{d_j \times (f_{j,\max}-f_{j,k})}{f_{j,\max}-f_{j,\min}}}. \tag{12}$$

Based on Eqs. (10) and (12), when task t_i is executed on processor pr_j with frequency $f_{j,k}$, the reliability of task t_i can be calculated by

$$R(t_i, pr_j, f_{j,k}) = e^{-\lambda_{j,\max} \times 10^{\frac{d_j \times (f_{j,\max}-f_{j,k})}{f_{j,\max}-f_{j,\min}}} \times \frac{w_{i,j} \times f_{j,\max}}{f_{j,k}}}. \tag{13}$$

Assuming that the network transmission is reliable [22], the reliability of the application G can be calculated by

$$R(G) = \prod_{i=1}^{N} R(t_i, pr_{asp(t_i)}, f_{asp(t_i),asf(t_i)}). \tag{14}$$

3.5. Problem description

Consider a DAG-based parallel application G executed on a heterogeneous embedded system, the reliability goal of the application is given as $R_{\text{goal}}(G)$. The problem of this paper is to minimize the energy consumption of the system by assigning each task in G to the appropriate processor, while satisfying the application's reliability goal. Formalized description is minimized

$$E(G) = E_{\text{d}}(G) + E_{\text{s}}(G) \tag{15}$$

subject to

$$R(G) = \prod_{i=1}^{N} R(t_i, pr_{asp(t_i)}, f_{asp(t_i),asf(t_i)}) \geq R_{\text{goal}}(G) \tag{16}$$

for all i: $1 \leq i \leq N$, $pr_{asp(t_i)} \in PR$, and $f_{asp(t_i),\text{low}} \leq f_{asp(t_i),asf(t_i)} \leq f_{asp(t_i),\max}$.

4. Preliminaries

4.1. Reliability goal

According to Eq. (13), the maximum reliability and minimum reliability of task t_i can be calculated as

$$R_{\max}(t_i) = \max_{pr_j \in PR, f_{j,\text{low}} \leq f_{j,k} \leq f_{j,\max}} \{R(t_i, pr_j, f_{j,k})\}, \tag{17}$$

and

$$R_{\min}(t_i) = \min_{pr_j \in PR, f_{j,\text{low}} \leq f_{j,k} \leq f_{j,\max}} \{R(t_i, pr_j, f_{j,k})\} \tag{18}$$

respectively.

Based on Eqs. (17) and (18), the maximum reliability and minimum reliability of the application can be calculated as

$$R_{\max}(G) = \prod_{i=1}^{N} R_{\max}(t_i), \tag{19}$$

and

$$R_{\min}(G) = \prod_{i=1}^{N} R_{\min}(t_i) \tag{20}$$

respectively.

Based on Eqs. (19) and (20), the reliability goal $R_{\text{goal}}(G)$ of the application should be less than or equal to $R_{\max}(G)$. If $R_{\text{goal}}(G)$ is less than or equal to $R_{\min}(G)$, the reliability goals can always be satisfied. Therefore, the range of reliability goal of the application in this paper is

$$R_{\min}(G) < R_{\text{goal}}(G) \leq R_{\max}(G). \tag{21}$$

4.2. Task priority

Because the tasks in application G have precedence constraints, the concept of *Rank* is used to generate task topology order, the *Rank* value of task t_i is defined as

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

where $\overline{w_i} = (\sum_{j=1}^{M} w_{i,j})/M$ represents the average WCET of the task t_i on each processor. The *Rank* values of each task in motivation example are shown in Table 1, the task with a greater the *Rank* value has a higher priority, so the assignment of the tasks according to the non-ascending order of the *Rank* values can meet the priority requirements. Assuming that the task t_i mentioned below has been arranged a non-ascending order.

4.3. Earliest start time and earliest finish time

When the DAG-based parallel application is executed on a heterogeneous embedded system, the earliest start time (EST) and the earliest finish time (EFT) of task t_i can be expressed as

$$\begin{cases} EST(t_{\text{entry}}, pr_j) = 0 \\ EST(t_i, pr_j) = \max \left\{ \begin{array}{l} avail[j], \\ \max_{t_x \in parent(t_i)} \{EFT(t_x, pr_y) + c'_{x,i}\} \end{array} \right\}, \end{cases} \tag{23}$$

and

$$EFT(t_x, pr_y) = EST(t_x, pr_y) + \frac{w_{x,y} \times f_{y,\max}}{f_{y,asf(t_x)}}. \tag{24}$$

In Eq. (23), $avail[j]$ represents the earliest available time for the processor pr_j. If t_x and t_i are assigned to the same processor, then $c'_{x,i} = 0$; otherwise, $c'_{x,i} = c_{x,i}$.

5. Proposed algorithms

The problem of this paper is minimizing the energy consumption with reliability goal for DAG-based parallel application on heterogeneous embedded systems. Based on Eq. (14), the reliability of the application is the product of the reliability of all tasks that belong to the application. Therefore, the reliability goal of the application can be transformed to the reliability goal of each task [22,23,31,32]. It is important to note that this transformation does not mean that the tasks are independent.

In this paper, we first transform the reliability goal of the application into the reliability goal of each task, and then assign the tasks to the processor that satisfies the reliability goals. The benefits of this method are as follows:

(1) If the reliability goal of all tasks can be satisfied, the reliability goal of the application will be satisfied.

(2) When the reliability goal of the task is determined, the heuristic method can be used to minimize the energy consumption of the application, thus reducing the time complexity of the scheduling algorithm.

When assigning current task, if the reliability goal of current task is too high, the probability of choosing the high energy efficiency processor will be reduced. This may reduce the energy efficiency. Similarly, if the reliability goal of current task is too low, the reliability requirements of the non-assigned tasks will be higher, which may also reduce energy efficiency. In addition, if the task's reliability goal is greater than the maximum reliability that can be reached, the task assignment will be fail. Therefore, it is challenging to transform the application's reliability goal into the reliability goal of each task.

Considering that some systems may not support DVFS technique [8,26], this paper designs two algorithms, namely, the non-DVFS energy-efficient scheduling algorithm with the reliability goal, and the DVFS energy-efficient scheduling algorithm with the reliability goal. We call these two algorithms NDERG and DERG, respectively.

5.1. The reliability goal of NDERG

When the reliability goal of the application is transformed to the reliability goal of each task, the reliability predetermined to task t_i must be made not greater than $R_{\max}(t_i)$, so $R_{\max}(t_i)$ can be used as a reference for the reliability goal of task t_i. This paper defines reliability goal ratio as Definition 1.

Definition 1 (*Reliability Goal Ratio*)**.** the reliability goal ratio *RGR* as the ratio of the reliability goal of the application to the maximum reliability of the application:

$$RGR(G) = \frac{R_{\text{goal}}(G)}{R_{\max}(G)}. \tag{25}$$

RGR describes the relationship between the reliability goal and the maximum reliability. The smaller the *RGR*, the lower the reliability requirements, the greater the *RGR*, the higher the reliability requirements. It is clear that $RGR(G)$ is less than or equal to 1. Now, Eq. (25) can be written as

$$R_{\text{goal}}(G) = RGR(G) \times R_{\max}(t_1) \times R_{\max}(t_2) \times \cdots \times R_{\max}(t_N). \tag{26}$$

Therefore, decomposing $RGR(G)$ to each task can predetermine the corresponding reliability goal of the task. Based on Eq. (26), we have:

$$\begin{cases} R_{\text{goal}}(G) = (R_{\max}(t_1) \times x_1) \times \cdots \times (R_{\max}(t_N) \times x_N) \\ x_1 \times x_2 \times \cdots \times x_N = RGR(G) \end{cases} \tag{27}$$

For any x_i ($1 \le i \le N$, $0 < x_i \le 1$), the reliability goal of task t_i can be written as $R_{\max}(t_i) \times x_i$.

In fact, the actual reliability of the assigned task is greater than or equal to the predetermined reliability goal. As a result, the tasks, that were executed earlier (high priority tasks), are predetermined with relatively low reliability goals. The tasks that will be executed later (low priority tasks), are predetermined with relatively high reliability goals. In this paper, the average WCET of the task t_i on each processor is used to predetermine the reliability goal of each task. Assume that there are N tasks in application G, the predetermined reliability goal of the ith task t_i is described as follows.

(1) Let predetermined value $pd_i = \overline{w_i}$ for all tasks t_i. According to Eq. (13), the reliability of tasks with a longer execution time will be lower, so tasks with a longer average WCET are predetermined with lower reliability goals. In addition, when task t_i is assigned, the actual reliability is generally much higher than $R_{\text{goal}}(t_i)$ because DVFS is not used to reduce processor's execution frequency. Therefore, it is necessary to further process the value of pd_i.

(2) Define a compensation sequence CS as $\{cs_1, cs_2, \ldots, cs_N\}$, where $cs_i = \overline{w_i}$ and then sort CS by non-ascending order.

(3) Let $pd_i = pd_i + cs_i$ for all tasks t_i. After the above processing, the predetermined value pd of the task that executed earlier is increased by a larger compensation value and the predetermined value pd of the task that executed later is increased by a smaller compensation value. In other words, the earlier the task executed, the greater the compensation value is added. The later the task executed, the smaller the compensation value is added. This helps to reduce the reliability goals of the tasks that executed earlier.

(4) The predetermined reliability goal of task t_i is given by

$$R_{\text{pregoal}}(t_i) = R_{\max}(t_i) \times RGR(G)^{\frac{pd_i}{S}}, \tag{28}$$

where $S = \sum_{i=1}^{N} pd_i$.

Because of $RGR(G) \le 1$ and $\frac{pd_i}{S} \le 1$, we have

$$R_{\text{pregoal}}(t_i) \le R_{\max}(t_i). \tag{29}$$

During the tasks assignment, let $\{t_1, t_2, \ldots, t_{i-1}\}$ represent the tasks that have been assigned to the processor, task t_i is currently being assigned to the processor, and $\{t_{i+1}, t_{i+2}, \ldots, t_N\}$ represents the tasks that have not been assigned to the processor. The reliability goal of task t_i is calculated by

$$R_{\text{goal}}(t_i) = \frac{R_{\text{goal}}(G)}{\prod_{x=1}^{i-1} R(t_x) \times \prod_{y=i+1}^{N} R_{\text{pregoal}}(t_y)}. \tag{30}$$

In Eq. (30), the items $\prod_{x=1}^{i-1} R(t_x)$ represent the actual reliability of tasks that have been assigned. The items $\prod_{y=i+1}^{N} R_{\text{pregoal}}(t_y)$ represent the predetermined reliability of remaining tasks that have not been assigned.

Theorem 1. *When the reliability goal of the application G is less than or equal to $R_{\max}(G)$, and Eqs. (28) and (30) are used to predetermine the reliability goal of t_i, the processors that satisfy the reliability goal of the application can always be found.*

Proof. Mathematical induction is used to prove Theorem 1.

Step 1: When the first task (i.e. $n = 1$) is currently being assigned to the processor, according to Eq. (30), the reliability goal of the first tasks is

$$R_{\text{goal}}(t_1) = \frac{R_{\text{goal}}(G)}{\prod_{y=2}^{N} R_{\text{pregoal}}(t_y)}. \tag{31}$$

Substituting Eqs. (25) and (28) into Eq. (31), yields

$$R_{\text{goal}}(t_1) = R_{\max}(t_1) \times RGR(G)^{\frac{pd(1)}{S}} \le R_{\max}(t_1). \tag{32}$$

So, the first task can be assigned to the appropriate processor. The actual reliability of the first task assigned to the processor is given by

$$R(t_1) \geq R_{\text{goal}}(t_1) = \frac{R_{\text{goal}}(G)}{\prod_{y=2}^{N} R_{\text{pregoal}}(t_y)}. \tag{33}$$

If the reliability of the rest tasks are predetermined by Eq. (28), then the reliability of application is given by

$$\begin{aligned} R(G) &= R(t_1) \times \prod_{i=2}^{N} R_{\text{pregoal}}(t_i) \\ &\geq \frac{R_{\text{goal}}(G)}{\prod_{y=2}^{N} R_{\text{pregoal}}(t_y)} \times \prod_{y=2}^{N} R_{\text{pregoal}}(t_y) \\ &= R_{\text{goal}}(G). \end{aligned} \tag{34}$$

So, when $n = 1$, Theorem 1 is correct.

Step 2: Assuming that when the kth task (i.e. $n = k$) is assigned, Theorem 1 is still correct, namely

$$R(G) = \prod_{i=1}^{k} R(t_i) \times \prod_{y=k+1}^{N} R_{\text{pregoal}}(t_y) \geq R_{\text{goal}}(G). \tag{35}$$

Step 3: If the reliability goal of $(k+1)th$ task (i.e. $n = k+1$) is predetermined by Eq. (30), namely

$$R_{\text{goal}}(t_{k+1}) = \frac{R_{\text{goal}}(G)}{\prod_{i=1}^{k} R(t_i) \times \prod_{y=k+2}^{N} R_{\text{pregoal}}(t_y)}. \tag{36}$$

Based on Eq. (35), we have

$$\prod_{i=1}^{k} R(t_i) \geq \frac{R_{\text{goal}}(G)}{\prod_{y=k+1}^{N} R_{\text{pregoal}}(t_y)}. \tag{37}$$

Substituting Eq. (37) into Eq. (36), gives

$$\begin{aligned} R_{\text{goal}}(t_{k+1}) &\leq \frac{R_{\text{goal}}(G)}{\frac{R_{\text{goal}}(G)}{\prod_{y=k+1}^{N} R_{\text{pregoal}}(t_y)} \times \prod_{y=k+2}^{N} R_{\text{pregoal}}(t_y)} \\ &= R_{\text{pregoal}}(t_{k+1}). \end{aligned} \tag{38}$$

So, the $(k+1)th$ task can be assigned to the appropriate processor and the actual reliability of the $(k+1)th$ task that assigned to the processor is given by

$$R(t_{k+1}) \geq R_{\text{goal}}(t_{k+1}) = \frac{R_{\text{goal}}(G)}{\prod_{i=1}^{k} R(t_i) \times \prod_{y=k+1}^{N} R_{\text{pregoal}}(t_y)}. \tag{39}$$

The reliability of the application should be

$$R(G) = \prod_{i=1}^{k} R(t_i) \times R(t_{k+1}) \times \prod_{y=k+2}^{N} R_{\text{pregoal}}(t_y). \tag{40}$$

Substituting Eq. (39) into Eq. (40) gives the following inequality:

$$R(G) \geq R_{\text{goal}}(G). \tag{41}$$

Therefore, when $n = k+1$, Theorem 1 is correct.

From step 1 to step 3, Theorem 1 is proved.

5.2. The NDERG algorithm

In this section, the NDERG algorithm is proposed to minimize the energy consumption while satisfying the reliability goal of the application. According to the above analysis, the actual reliability of the task t_i should be greater than or equal to $R_{\text{goal}}(t_i)$ when t_i is assigned to a processor. Therefore, the NDERG algorithm first calculates the reliability goal of the task and then selects the appropriate processor to assign the task. The NDERG algorithm is described in Algorithm 1.

Algorithm 1 NDERG

Input: $PR = \{pr_1, pr_2, \ldots, pr_M\}$ and application G
Output: $E(G)$ and $R(G)$

```
1:  sort the tasks to queue ReadyQ by non-ascending order of Rank
2:  while ReadyQ is not empty do
3:    t_i ← ReadyQ.out()
4:    calculate R_goal(t_i) using Eq. (30)
5:    energy ← ∞
6:    for each processor pr_j ∈ PR do
7:      if R(t_i, pr_j, f_j,max) ≥ R_goal(t_i) then
8:        if energy > E_d(t_i) then
9:          energy ← E_d(t_i)
10:         asp(t_i) ← j
11:       end if
12:     end if
13:   end for
14: end while
15: calculate E(G) using Eq. (9)
16: calculate R(G) using Eq. (14)
```

Table 2
Power and reliability parameters of processors.

pr_j	$P_{j,\text{s}}$	$P_{j,\text{ind}}$	$C_{j,\text{ef}}$	m_j	$\lambda_{j,\max}$	d_j
pr_1	0.001	0.003	1.2	2.9	0.0002	2.3
pr_2	0.001	0.005	1.0	2.7	0.0004	2.1
pr_3	0.001	0.007	0.8	2.5	0.0006	2.5

The main idea of the NDERG algorithm is that the reliability goal of the application is transformed to the reliability goal of each task. For any task, it will be assigned to the processor that satisfies the reliability goal and consumes the least energy. The details of NDERG algorithm are explained as follows:

The structure of the NDERG algorithm is a nested loop, and the tasks are sorted to queue *ReadyQ* by non-ascending order of *Rank* value before the loops (first line). The outer loop assigns the tasks to the appropriate processor one by one (Lines 2–14). For each task t_i, its reliability goal is calculated first by Eq. (30), and then the NDERG enters the inner loop (Lines 6–13), where the processor which satisfies reliability goal of task t_i and consumes the least energy is chosen to assign the task t_i (Lines 8–11). Lines 15–16 calculate energy consumption and actual reliability of application respectively.

The time complexity of the NDERG algorithm is analyzed as follows. The outer loop traverses all the tasks in application with time complexity of $O(N)$. The inner loop traverses all the processors to find minimum energy consumption of task t_i with the worst time complexity of $O(M \times N)$. Therefore, the worst time complexity of NDERG is $O(M \times N^2)$, which is equal to that of the HEFT [20] and MRCRG [22].

5.3. Example of the NDERG algorithm

This section describes the motivation example (see Section 3.2) using the NDERG algorithm. We assume that the power and reliability parameters for all processors are known and shown in Table 2, where the maximum frequency $f_{j,\max}$ for each processor is 1.0. The energy consumption rate of communication *ecr* is 0.5. For the sake of simplicity, this example ignores all the units of all the parameters. Thus, when the non-DVFS processor is used, we can calculate that the maximum reliability and minimum reliability of

Table 3
The results of the motivation example using the NDERG.

t_i	$R_{goal}(t_i)$	$R(t_i)$	pr_j	ST	FT	$E_{dp}(t_i, pr_j, f_{j,max})$	$E_{com}(t_i, pr_j)$	$E_d(t_i)$
t_1	0.994181	0.994615	3	0	9	7.263	0	7.263
t_3	0.994208	0.994813	2	21	34	13.065	6	19.065
t_4	0.994011	0.996805	2	34	42	8.040	4.5	12.540
t_2	0.991457	0.997403	1	27	40	15.639	9	24.639
t_5	0.989155	0.994018	3	9	19	8.070	0	8.070
t_6	0.989954	0.994615	3	19	28	7.263	0	7.263
t_9	0.988764	0.995212	2	56	68	12.060	14.5	26.560
t_7	0.989833	0.994018	2	68	83	15.075	0	15.075
t_8	0.992658	0.999000	1	69	74	6.015	21	27.015
t_{10}	0.988375	0.997204	2	85	92	7.035	5.5	12.535

$R(G) = 0.95848631 > R_{goal}(G) = 0.95$, $SL(G) = 92$, $E(G) = E_d(G) + E_s(G) = 160.301$

the application are $R_{max}(G) = 0.976286$ and $R_{min}(G) = 0.915944$ according to Eqs. (19) and (20).

When the reliability goal of the application is set to $R_{goal}(G) = 0.95$, Table 3 shows the results of the motivation example using the NDERG algorithm, where ST and FT represent the start time and the finish time of task t_i, respectively. The actual reliability $R(t_i)$ is greater than reliability goal $R_{goal}(t_i)$. Finally, the actual reliability of the application is $R(G) = 0.95848631$, which greater than $R_{goal}(G) = 0.95$, and the total energy consumption of application G is $E(G) = E_d(G) + E_s(G) = 160.301$. In addition, from Table 3, it is easy to know the scheduling length $SL(G) = 92$.

The NDERG algorithm does not scale the execution frequency of the processor. When the processor's execution frequency is scaleable, the frequency reduction may lead to further improved energy efficiency. Therefore, an algorithm that uses DVFS technique to minimize energy consumption while satisfying the reliability goal (DERG) is designed. The use of DVFS technique to reduce the execution frequency of the processor will reduce the reliability of the task. Therefore, when the reliability goal of a task is determined, the task may be executed with reduced frequency, and its actual reliability may be very close to the reliability goal. As a result, if the reliability goal of the tasks that executed earlier is too low, the tasks that executed later must have high reliability, which may reduce the energy efficiency of the system. Therefore, it is necessary to redefine the predetermined reliability goal of the task.

5.4. The reliability goal of DERG

In order to make the tasks' predetermined reliability goal more balanced, $\overline{w_i}$ is used as a reference for calculating the predetermined reliability goal of task t_i. Therefore, the predetermined reliability goal of the task in DERG is defined as

$$R'_{pregoal}(t_i) = R_{max}(t_i) \times RGR(G)^{\frac{\overline{w_i}}{S}}, \tag{42}$$

where $S = \sum_{i=1}^{N} \overline{w_i}$. At the time of task assignment, the reliability goal of the task t_i is calculated by

$$R'_{goal}(t_i) = \frac{R_{goal}(G)}{\prod_{x=1}^{i-1} R(t_x) \times \prod_{y=i+1}^{N} R'_{pregoal}(t_y)}. \tag{43}$$

The predetermined reliability goal of the task $R'_{pregoal}(t_i)$ in DERG is different from $R_{pregoal}(t_i)$ in NDERG. DERG improves the predetermined reliability goal of the tasks that executed earlier compared with the NDERG algorithm.

Similar to the reliability goal of the NDERG algorithm, when the reliability goal of the application G is less than or equal to $R_{max}(G)$, Eqs. (42) and (43) are used to predetermine the reliability of task t_i, the processors that satisfy the reliability goal of the application can always be found.

Algorithm 2 DERG

Input: $PR = \{pr_1, pr_2, ..., pr_M\}$ and application G
Output: $E(G)$ and $R(G)$
1: sort the tasks to queue *ReadyQ* by non-ascending order of *Rank*
2: **while** *ReadyQ* is not empty **do**
3: $t_i \leftarrow ReadyQ.out()$
4: calculate $R'_{goal}(t_i)$ using Eq. (43)
5: $energy \leftarrow \infty$
6: **for** each processor $pr_j \in PR$ **do**
7: **for** processor frequency $f_{j,k}$ increases from $f_{j,low}$ to $f_{j,max}$ **do**
8: **if** $R(t_i, pr_j, f_{j,k}) \geq R'_{goal}(t_i)$ **then**
9: **if** $energy > E_d(t_i)$ **then**
10: $energy \leftarrow E_d(t_i)$
11: $asp(t_i) \leftarrow j$
12: $asf(t_i) \leftarrow k$
13: **break**
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18: **end while**
19: calculate $E(G)$ using Eq. (9)
20: calculate $R(G)$ using Eq. (14)

5.5. The DERG algorithm

Based on the previous analysis, the DERG algorithm is described in Algorithm 2.

The main idea of the DERG algorithm is that the reliability goal of the application is also transformed to the reliability goal of each task. For any task, it will be assigned to the processor that satisfies the reliability goal and consumes the least energy. The DERG algorithm should reduce processor frequency as much as possible in order to improve energy efficiency.

The structure of the DERG algorithm is also a nested loop, and the tasks are sorted to queue *ReadyQ* by non-ascending order of *Rank* value before the loops (first line). The outer loop assigns the tasks to the appropriate processor one by one (Lines 2–18). For each task t_i, its reliability goal is calculated first by Eq. (43), and then the DERG enters the inner loop (Lines 6–17), the processor pr_j and frequency $f_{j,k}$, which satisfy reliability goal of task t_i and minimum energy consumption is chosen to assign the task t_i.

The structure of the DERG algorithm is the same as that of the NDERG algorithm, so the time complexity of the DERG algorithm is $O(M \times N^2 \times mfs)$, where mfs represents the maximum number of discrete frequency levels in all processors.

Table 4
The results of the motivation example using the DERG.

t_i	$R_{goal}(t_i)$	$R(t_i)$	pr_j	$f_{j,k}$	ST	FT	$E_{dp}(t_i, pr_j, f_{j,k})$	$E_{com}(t_i, pr_j)$	$E_d(t_i)$
t_1	0.994554	0.994615	3	1	0	9	7.263	0	7.263
t_3	0.994818	0.995272	1	0.9	21	33.22	10.842	6	16.842
t_4	0.994367	0.996805	2	1	18	26	8.040	4.5	12.540
t_2	0.991575	0.994415	1	0.9	33.22	47.67	12.813	9	21.813
t_5	0.992381	0.994018	3	1	9	19	8.070	0	8.070
t_6	0.993182	0.994615	3	1	19	28	7.263	0	7.263
t_9	0.991583	0.995212	2	1	63.67	75.67	12.060	14.5	26.560
t_7	0.992722	0.993444	1	0.8	47.67	56.42	5.524	0	5.524
t_8	0.996233	0.997848	1	0.9	56.42	61.97	4.928	21	25.928
t_{10}	0.992605	0.994322	2	0.9	75.67	83.45	5.891	14	19.891
$R(G) = 0.95164361 > R_{goal}(G) = 0.95, SL(G) = 83.45, E(G) = E_d(G) + E_s(G) = 151.944$									

5.6. Example of the DERG algorithm

This example still uses the parameters in the Section 5.3. The maximum frequency $f_{j,max}$ for each processor is 1.0 and the frequency precision is set at 0.1, the lowest frequency of each processor is calculated by Eq. (3).

Table 4 shows the results of the motivation example of using the DERG algorithm. The reliability goal of task t_1 is $R'_{goal}(t_1) = 0.994554$, which is greater than $R_{goal}(t_1) = 0.994181$ (see Table 3) in NDERG. Although the actual reliability of task t_1 is the same in DERG and NDERG, the reliability goal of second task $R'_{goal}(t_3)$ is still larger than $R_{goal}(t_3)$. Task t_3 executing with reduced frequency leads to higher reliability goals for t_4 to t_{10}. Because the reliability goal of the task in the DERG algorithm is designed based on the average WCET of the tasks, t_2, t_7, t_8, and t_{10} are still executed with reduced frequency to improve energy efficiency.

The final actual reliability of the application is $R(G) = 0.95164361$, which is greater than $R_{goal}(G) = 0.95$, and the total energy consumption of application is $E(G) = E_d(G) + E_s(G) = 151.944$, which is less than the $E(G)$ value (160.301) obtained by the NDERG algorithm. In addition, as shown in Tables 3 and 4, there are some tasks assigned to different processors by different algorithms. For example, t_3 is assigned to pr_2 in the NDERG algorithm, but it is assigned to pr_1 in the DERG algorithm. So, an interesting result is that although DVFS technology reduces the execution frequency and thus stretches the execution time of some tasks, the scheduling length of application is not always larger than non-DVFS scheduling. Here the scheduling length of application is $SL(G) = 83.45$, which is less than the scheduling length (the value is 92) obtained by the NDERG algorithm.

6. Experimental performance evaluation

6.1. Experimental parameters

According to the relevant parameters in [22,23,30], the system parameters are shown as follows:

(1) Application parameters: The WCET of task t_i assigned to the processor pr_j is 10 ms $\leq w_{i,j} \leq$ 100 ms, and the communication time between t_i and t_j is 10 ms $\leq c_{i,j} \leq$ 100 ms.

(2) Processor power parameters: $P_{j,s} = 0.001$, $0.03 \leq P_{j,ind} \leq 0.07$, $0.8 \leq C_{j,ef} \leq 1.2$, $2.5 \leq m_j \leq 3.0$, and $f_{j,max} = 1.0$ GHz. The frequency precision is set at 0.1 GHz, and the lowest frequency of each processor is calculated by Eq. (3).

(3) Reliability parameters: The transient fault rate with the maximum frequency of the processor is $0.000001 \leq \lambda_j \leq 0.000009$, and the sensitivity fault rate of voltage scaling is $1.0 \leq d_j \leq 3.0$.

(4) The energy consumption rate of communication $ecr = 0.5$ W.

6.2. Comparison algorithms

MRCRG algorithm [22] consumes less resource than MaxRe [32] and the RR algorithm [31] for a DAG-based parallel application on heterogeneous systems while satisfying the reliability goal requirement. MRCRG is a state-of-the-art method, which can also be used to improve energy efficiency. Considering that the use of DVFS technique, we also extend the MRCRG with DVFS enabled. We call such method as MECRG (minimizing energy consumption with reliability goal). In addition, ESRG is also energy-efficient scheduling with reliability goal for a DAG-based parallel application on heterogeneous systems [23]. Therefore, the algorithm proposed in this paper is mainly compared with the MRCRG, MECRG, and ESRG algorithms. The methods of assigning each task's reliability goal of MRCRG and ESRG algorithms are briefly introduced as follows:

MRCRG: The reliability goal of task t_i in MRCRG algorithm is calculated by

$$R_{goal}(t_i) = \frac{R_{goal}(G)}{\prod_{x=1}^{i-1} R(t_x) \times \prod_{y=i+1}^{N} R_{max}(t_y)}. \tag{44}$$

In Eq. (44), the items $\prod_{x=1}^{i-1} R(t_x)$ represent the actual reliability of tasks that have been assigned, the items $\prod_{y=i+1}^{N} R_{max}(t_y)$ represent the predetermined reliability of remaining tasks that have not been assigned.

ESRG: The ESRG algorithm calculates upper bound reliability goal of the tasks as $R_{up\_goal}(t_i) = \sqrt[N]{R_{goal}(G)}$, the reliability goal of task t_i is calculated by

$$R_{goal}(t_i) = \frac{R_{goal}(G)}{\prod_{x=1}^{i-1} R(t_x) \times \prod_{y=i+1}^{N} R_{up\_goal}(t_y)}. \tag{45}$$

In Eq. (45), the items $\prod_{x=1}^{i-1} R(t_x)$ represent the actual reliability of tasks that have been assigned, the items $\prod_{y=i+1}^{N} R_{up\_goal}(t_y)$ represent the predetermined reliability of remaining tasks that have not been assigned. Because $R_{goal}(t_i)$ may be larger than $R_{max}(t_i)$, ESRG cannot always satisfy the reliability goal requirements of the application without using replication technique.

6.3. Experimental evaluation

C++ programming is used to simulate the algorithms. Some commonly used DAG-based parallel applications in the distributed system, such as Gaussian elimination and Fourier transform applications [16,22,28], are used to simulate the algorithms, which are briefly introduced as follows.

Gaussian elimination application: Gaussian elimination is a kind of parallel applications which has precedence constraints. A parameter ρ is used to describe the size of the Gaussian elimination application, the total number of tasks in application is $N = \frac{\rho^2+\rho-2}{2}$. Fig. 2 shows a Gaussian elimination application with $\rho = 5$.

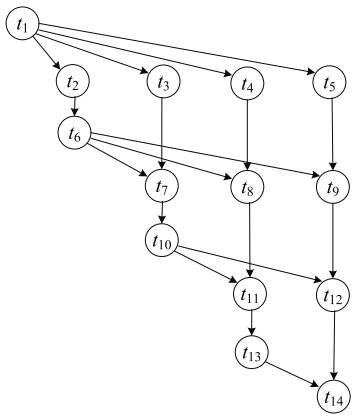


Fig. 2. Gaussian elimination application with $\rho = 5$.

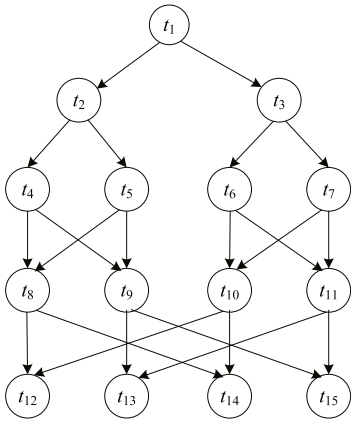


Fig. 3. Fourier transform application with $\rho = 4$.

Fourier transform application: A parameter ρ is used to describe the size of the Fourier transform application, the total number of tasks is $N = (2 \times \rho - 1) + \rho \times log_2{}^{\rho}$ with $\rho = 2^y$, where y is a positive integer. Fig. 3 shows a Fourier transform application with $\rho = 4$.

In all of the following experiments, we mainly compare the energy consumptions and actual reliabilities generated by each algorithm. In order to demonstrate how the scheduling and DVFS affect the execution time of the applications, the scheduling lengths generated by each algorithms are also compared.

6.3.1. Varying the application scales

Experiment 1: This experiment mainly tests the effects of different scales of Gaussian elimination applications on the performance of the algorithms. 32 processors are used in this experiment, the reliability requirement of application is 0.95. ρ is 16, 24, 32, 40, and 48 respectively, i.e., the total number of tasks is 135 (small scale), 299, 527, 819, and 1175 (large scale) respectively. The energy consumption, actual reliability, and the scheduling length with different algorithms are shown in Fig. 4.

Fig. 4(a) shows the energy consumptions of the different algorithms. As the number of tasks increases, the energy consumptions generated by all algorithms are increased. Overall, MECRG generates the highest energy consumption which is higher than that is generated by ESRG and MRCRG. DERG generates the lowest energy consumption. In detail, the energy consumption generated by NDERG is 92% of MRCRG, 86% of MECRG, and 89% of ESRG. The energy consumption generated by DERG is 89% of MRCRG, 83% of MECRG, and 86% of ESRG.

Fig. 4(b) shows the actual reliability of the application with different algorithms. When the number of tasks is less than or equal to 1175, all algorithms can make the application satisfy the reliability goal. When the application is scheduled with MRCRG and NDERG, the actual reliability of the application is almost the same, which is obviously greater than reliability goal. With the number of tasks increases, the actual reliabilities of the application generated by these two algorithms are gradually approaching to the reliability goal. However, when the application is scheduled with MECRG, ESRG, and DERG, the actual reliability of the application are about the same and slightly greater than reliability goal requirement.

Fig. 4(c) shows the scheduling length of the application with different algorithms. As the number of tasks increases, the scheduling lengths generated by all algorithms are increased. Overall, the scheduling lengths of the three algorithms MECRG, ESRG, and DERG are greater than that of the other two algorithms. In detail, when the scheduling length generated by the MRCRG algorithm is used as a reference, MECRG is 1.18 times of MRCRG, ESRG is 1.11 times of MRCRG, NDERG is 92% of MRCRG, and DERG is 1.02 times of MRCRG.

The main reasons for the above results are as follows.

(1) In the MRCRG and MECRG algorithms, the reliability goal of the tasks is extremely unbalanced. The predetermined reliability goal of early execution tasks is too low, the tasks executed later must have higher reliability. As a result, the chances of lately executed task choosing an energy efficient processor are reduced. Therefore, the energy consumptions generated by the MRCRG and MECRG algorithms are relatively high. Although the ESRG algorithm defines the upper bound of the reliability goal, the reliability goal of the tasks is still unbalanced, so the ESRG algorithm still generates more energy consumption. The NDERG and DERG algorithms use $\overline{w_i}$ as a reference to determine the reliability goal of task t_i, the reliability goal of the tasks is more balanced, each task has the opportunity to execute on energy-efficient processor. So the NDERG and DERG algorithms generate lower energy consumptions.

(2) The MRCRG and NDERG algorithms did not use DVFS technique. When reliability goal of application is very small compared with $R_{\max}(G)$, each task reliability goal will be lower. When these two algorithms assign tasks to the processor, the actual reliability of a task may be far greater than the reliability goal. Therefore, the actual reliability of application is far greater than the reliability goal. However, MECRG, ESRG, and DERG reduce the execution frequency as much as possible to improve energy efficiency, which weaken the actual reliability of the task. As a result the actual reliability of application is slightly greater than reliability goal requirement.

(3) The MECRG, ESRG, and DERG algorithms use DVFS technique, which extends the execution time of the task. Hence these three algorithms generate relatively long scheduling length.

Experiment 2: This experiment mainly tests the effect of different scales of Fourier transform applications on the performance of the algorithms. 32 processors are used in this experiment, the reliability requirement of application is 0.95. ρ is 8, 16, 32, 64, and 128 respectively, i.e., the total number of tasks is 39 (small scale), 95, 223, 511, and 1151 (large scale) respectively. The energy consumption, actual reliability, and the scheduling length with different algorithms are shown in Fig. 5.

Fig. 5(a) shows the energy consumption of the different algorithms. As the number of tasks increases, the energy consumptions generated by all algorithms are increased. However, the energy consumptions generated by the NDERG and DERG algorithms are relatively less.

As shown in Fig. 5(b), all algorithms can make the application satisfy the reliability goal. When the number of tasks is less than

or equal to 223, the actual reliabilities obtained by two algorithms without implementing DVFS technique in MRCRG and NDERG are far higher than the reliability goal. However, the actual reliabilities obtained by other three algorithms that implement DVFS technique are slightly higher than the reliability goal.

As shown in Fig. 5(c), when the number of tasks increases, the scheduling lengths generated by all algorithms are also increased. Among all algorithms, the MECRG algorithm generates the longest scheduling length, the NDERG algorithm generates the minimum scheduling length.

6.3.2. Varying the reliability goals

Experiment 3: This experiment mainly tests the effect of the different reliability goals of Gaussian elimination application on the performance of the algorithms. 32 processors are used in this experiment. $\rho = 32$, i.e., the total number of tasks is 527 (medium scale). When the reliability goal of the application increases from 0.93 to 0.97 with 0.01 increments, the energy consumption, actual reliability, and the scheduling length with different algorithms are shown in Fig. 6.

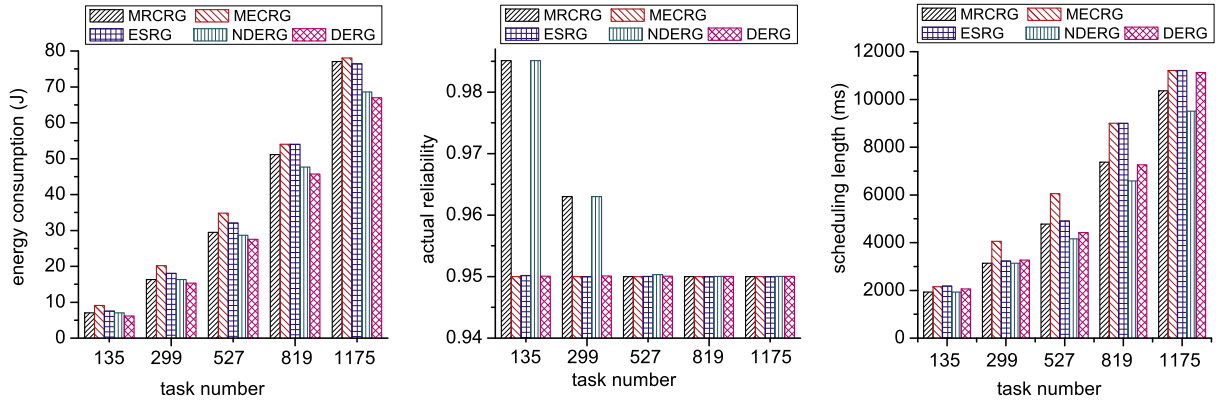
As shown in Fig. 6(a), with the improvement of the reliability goal of the application, the energy consumptions generated by all algorithms are increased, but the increase is not obvious. In the five algorithms, NDERG and DERG consume less energy than MRCRG, MECRG, and ESRG. Overall, the energy consumption generated by DERG is the least. In detail, the energy consumption generated by NDERG is 96% of MRCRG, 83% of MECRG, and 89% of ESRG, the energy consumption generated by DERG is 94% of MRCRG, 80% of MECRG, and 86% of ESRG.

As shown in Fig. 6(b), with the improvement of the reliability goal of the application, the actual reliabilities generated by MECRG, ESRG, and DERG are about the same and slightly greater than the reliability goal. Similarly, the actual reliabilities generated by MRCRG and DERG are also about the same. When the reliability goal is less than or equal to 0.97, the five algorithms can make the application satisfy the reliability goal. But the attention is required toward the reliability of the application which cannot be reached to 0.98 using ESRG algorithm.

As shown in Fig. 6(c), with the improvement of the reliability goal of the application, the scheduling lengths generated by the MRCRG, ESRG, NDERG and DERG algorithms are increased, but the scheduling length generated by the MECRG algorithm has little change. Overall, the MECRG algorithm generates the longest scheduling length, the NDERG algorithm generates the smallest scheduling length.

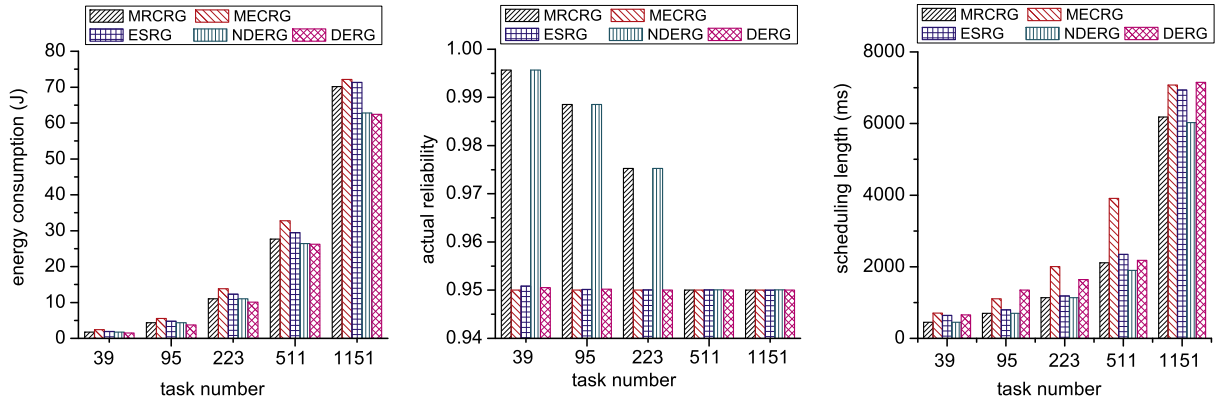
Experiment 4: This experiment mainly tests the effect of the different reliability goals of Fourier transform application on the performance of the algorithms. 32 processors are used in this experiment. $\rho = 64$, i.e., the total number of tasks is 511 (medium scale), the reliability goal of the application increases from 0.93 to 0.97 with 0.01 increments.

Fig. 7(a) shows the energy consumption with different algorithms. According to the results, as the reliability goal of the application increases, the energy consumptions generated by all algorithms are also increased. Overall, the DERG algorithm generates the lowest energy consumption.



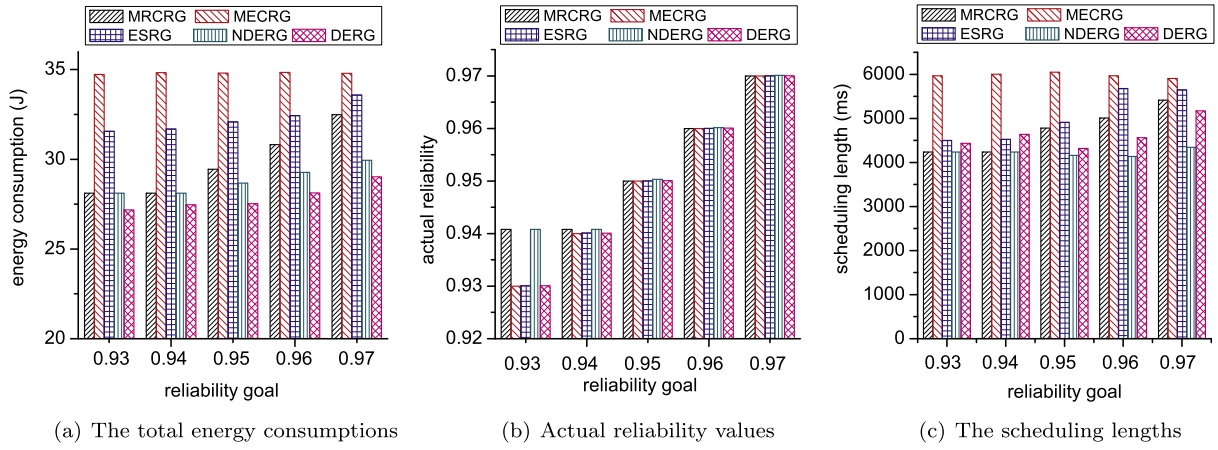
(a) The total energy consumptions (b) Actual reliability values (c) The scheduling lengths

Fig. 4. Results of the Gaussian applications with ρ are increased from 16 to 48 on reliability goal requirement $R_{goal}(G) = 0.95$ (**Experiment 1**).



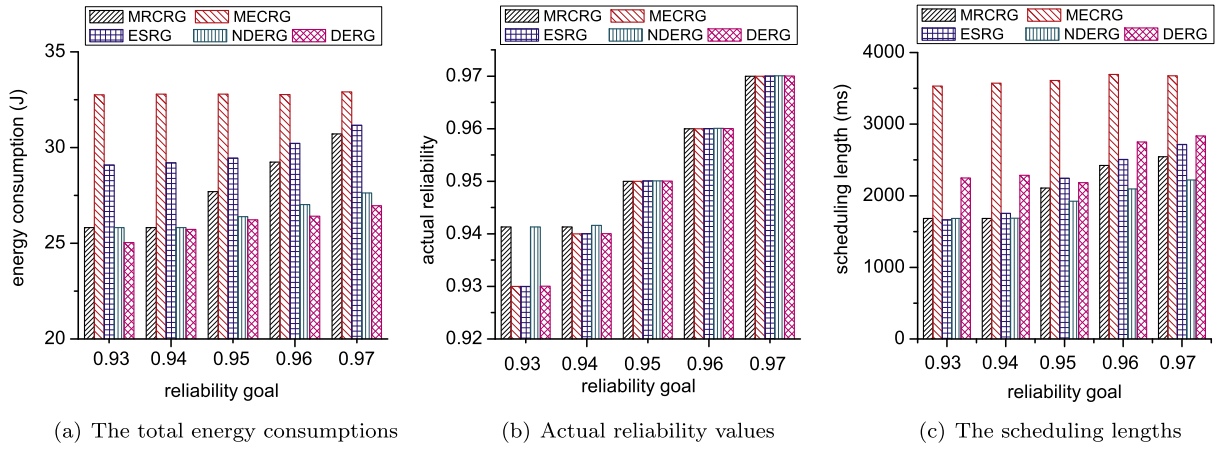
(a) The total energy consumptions (b) Actual reliability values (c) The scheduling lengths

Fig. 5. Results of the Fourier transform applications with ρ is increased from 8 to 128 on reliability goal requirement $R_{goal}(G) = 0.95$ (**Experiment 2**).



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

Fig. 6. Results of the Gaussian application with $\rho = 32$ on reliability goal requirements $R_{\text{goal}}(G)$ are increased from 0.93 to 0.97 (**Experiment 3**).



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

Fig. 7. Results of the Fourier transform application with $\rho = 64$ on reliability goal requirements $R_{\text{goal}}(G)$ are increased from 0.93 to 0.97 (**Experiment 4**).

As shown in Fig. 7(b), when the reliability goal of the application increases from 0.93 to 0.97, all the algorithms can satisfy the reliability goal, which are similar to that of **Experiment 3**.

As shown in Fig. 7(c), when the reliability goal of the application is increased, the scheduling lengths generated by the MRCRG, ESRG, NDERG, and DERG algorithms are also increased. However, the scheduling length generated by the MECRG algorithm has little change.

From **Experiment 1** to **experiment 4**, an interesting result is that when the reliability goal of the DAG-based parallel application must have to be satisfied, using DVFS technique to reduce the execution frequency of some tasks may increase the energy consumption of the system. For example, the energy consumption generated by MECRG is greater than that generated by MRCRG in all experiments. The main reasons can be explained as follows:

(1) When the task's reliability goal can be satisfied, DVFS technique reduces the execution frequency as much as possible to improve energy efficiency, which will make the actual reliability of the task closer to the reliability goal. Therefore, if the reliability goals of some tasks are too low, the reliability goals of other tasks must be higher, which will make the algorithm fall into local optimization.

(2) MECRG sets a low reliability goal of the early assigned tasks, as a result, more tasks executed later must be executed with maximum reliability and it is hard to be assigned to an energy efficiency processor. The MRCRG algorithm does not use DVFS technique, although the reliability goal of the early assigned tasks is very low, the actual reliabilities of these tasks may be much higher than the reliability goal. Therefore, the impact on the reliability goals of the tasks that are not assigned is relatively slightly.

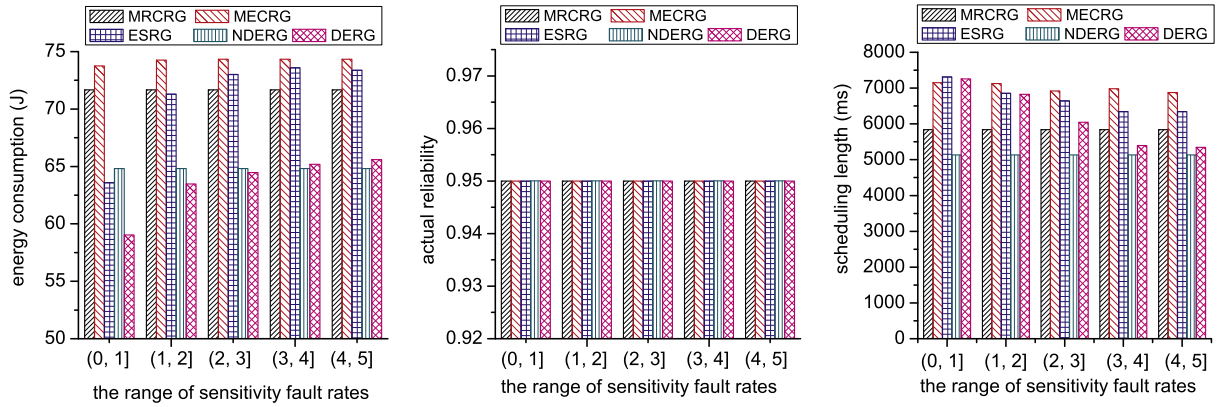
It is known from the above analysis, when DVFS technique is used to improve energy efficiency while satisfying the reliability goal of the DAG-based parallel application. For each task t_i, the $R_{\text{goal}}(t_i)$ is neither too high nor too low in respect of the maximum reliability $R_{\max}(t_i)$, it have to be carefully designed to prevent the algorithm fall into local optimization.

Reducing the execution frequency of task can improve energy efficiency, but the reliability of the task will be weakened at the same time. Intuitively, the sensitivity fault rates d (see Section 3.4) should have an impact on energy consumption when DVFS technique is used to improve energy efficiency while satisfying the reliability goal of the application, so we designed **experiment 5** and **experiment 6**.

6.3.3. *Varying the sensitivity fault rates*

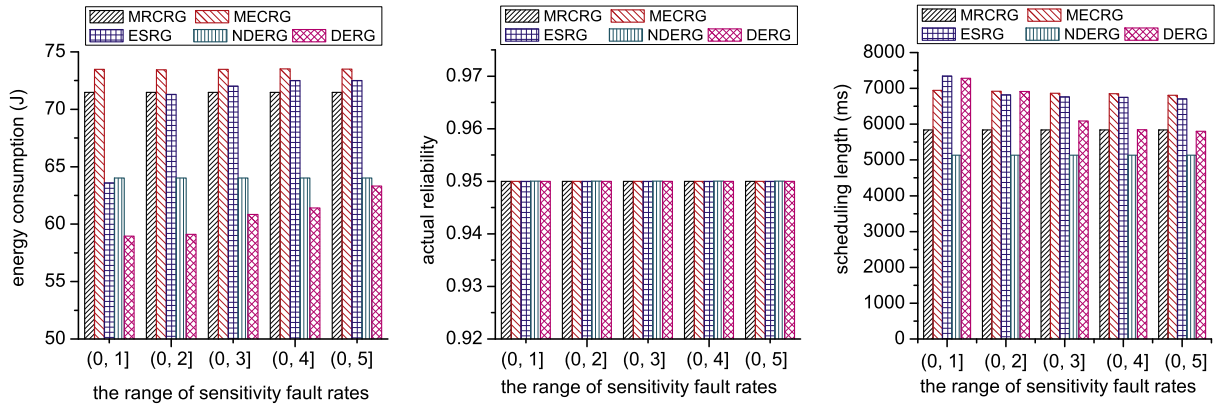
Experiment 5: This experiment mainly tests the effect of the sensitivity fault rates on the performance of the algorithms. Fourier transform application with $\rho = 128$, i.e., the total number of tasks is 1151 (large scale). 32 processors are used in this experiment. The reliability goal requirement of application is 0.95. When the sensitivity fault rate d_j is randomly generated within the range of (0, 1], (1, 2], (2, 3], (3, 4], and (4, 5] respectively, the energy consumption , actual reliability, and the scheduling length with different algorithms are shown in Fig. 8.

As shown in Fig. 8(a), in view of the overall situation, the energy consumptions of NDERG and DERG are relatively low among all



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

Fig. 8. Results of the Fourier transform applications with $\rho = 128$ on different rang of the sensitivity fault rates (**Experiment 5**).



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

Fig. 9. Results of the Fourier transform applications with $\rho = 128$ on different rang of the sensitivity fault rates (**Experiment 6**).

algorithms. When the sensitivity fault rate d_j is in range of (0, 1], ESRG and DERG have great ability to improve energy efficiency. When the sensitivity fault rate d_j is greater than 1, the energy efficiency of ESRG and DERG has been reduced. But in any case, the energy consumption of DERG is lower than that of ESRG. When the sensitivity fault rate d_j of each processor is greater than 3, the energy consumption of NDERG is the lowest among all algorithms. In fact, when the sensitivity fault rate d_j of each processor is greater than 3, because the fault rate is too high, MECRG, ESRG, and DERG are almost impossible to reduce the execution frequency for improving energy efficiency. Therefore, this experiment also shows that when the DVFS technique is not used, the method of predetermined the task reliability goal in the NDERG algorithm is effective.

As shown in Fig. 8(b), all algorithms can make the application satisfy the reliability goal requirements. Because the scale of the application is relatively large, the actual reliabilities generated by all algorithms are only slightly greater than the reliability goal.

As shown in Fig. 8(c), the scheduling lengths generated by the MRCRG and NDERG algorithms remain unchanged. As the sensitivity fault rate increases, the scheduling lengths generated by the MECRG, ESRG, and DERG algorithms have the trend of decreasing, which indicate that the number of tasks executed with reduced frequency is decreased.

Experiment 6: We only change the range of the sensitivity fault rate d_j, all the other parameters are the same as **Experiment 5**. When the sensitivity fault rate d_j is randomly generated within the range of (0, 1], (0, 2], (0, 3], (0, 4], and (0, 5] respectively, the energy consumption, actual reliability, and the scheduling length with different algorithms are shown in Fig. 9.

Experiment 6 is different from **Experiment 5**, In **Experiment 6**, the sensitivity fault rate d_j has a wider range of values, and some of the processor's d_j may be smaller. As a result, the execution frequency of these processors may be reduced to improve energy efficiency.

As shown in Fig. 9(a), the energy consumption of DERG is the lowest among all algorithms. In addition, the energy consumptions of ESRG and DERG are lower than that in Experiment 5. The main reason is that the sensitivity fault rate d_j of some processors may be very small, these two algorithms can reduce task execution frequency to improve energy efficiency, which demonstrates the performance of these algorithms in terms of energy efficiency.

Fig. 9(b) is similar to Fig. 8(b), all algorithms can make the application satisfy the reliability goal.

As shown in Fig. 9(c), the scheduling lengths generated by the MRCRG and NDERG algorithms remain unchanged. Among all algorithms, the NDERG algorithm generates the smallest scheduling length.

The results of **Experiment 1** to **Experiment 6** are summarized as follows:

(1) In general, for the algorithm with DVFS technique, the DERG algorithm generates the less energy consumption than MECRG and

ESRG. For the algorithm without using DVFS technique, the NDERG algorithm generates the less energy consumption than MRCRG.

(2) When the reliability goal of the application must have to be satisfied, using DVFS technique to reduce the execution frequency of some tasks may increase the energy consumption of the system. However, the energy consumption of DERG is less than that of NDERG for the general case.

(3) When the sensitivity fault rates of all processors are too high, and all algorithms would hardly reduce the processor execution frequency, the NDERG algorithm generates the least energy consumption.

7. Conclusions

Embedded heterogeneous systems are widely used in many fields, reducing the energy consumption of the system is an important research topic. There are many solutions that reduce processor execution frequency to improve energy efficiency by using DVFS technique. However, as the execution frequency of the processor reduces, the probability of failure in the processor will increase, and the reliability of the application will be weakened. Reliability is an important quality targets for many embedded applications, which should be satisfied. Based on the heterogeneous embedded system, the problem of satisfying the reliability goal of the DAG-based parallel application and minimizing the energy consumption of the system is studied in this paper. NDERG and DERG algorithms are proposed to minimize the energy consumption while satisfying the reliability goal. The reliability goal of the application is transformed to the reliability goal of each task, two schemes for predetermining the reliability goal of the task are proposed for above two algorithms, respectively. The experimental results show that the proposed algorithms have significant improvements in terms of energy efficiency compared with the state-of-the-art algorithms.

Acknowledgment

The research was partially funded by the National Natural Science Foundation of China (Grant No. 61173036).

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] 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.
[3] I. Assayad, A. Girault, H. Kalla, Scheduling of real-time embedded systems under reliability and power constraints, in: Complex Systems (ICCS), 2012 International Conference on, IEEE, 2012, pp. 1–6.
[4] Y. Ding, X. Qin, L. Liu, T. Wang, Energy efficient scheduling of virtual machines in cloud with deadline constraint, Future Gener. Comput. Syst. 50 (2015) 62–74.
[5] M. Fan, Q. Han, X. Yang, Energy minimization for on-line real-time scheduling with reliability awareness, J. Syst. Softw. 127 (2017) 168–176.
[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] 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.
[8] C. Kuo, Y. Lu, Task assignment with energy efficiency considerations for non-DVS heterogeneous multiprocessor systems, SIGAPP Appl. Comput. Rev. 14 (4) (2015) 8–18.
[9] K. Li, Energy-efficient task scheduling on multiple heterogeneous computers: Algorithms, analysis, and performance evaluation, IEEE Trans. Sustain. Comput. 1 (1) (2016) 7–19.
[10] K. Li, Power and performance management for parallel computations in clouds and data centers, J. Comput. System Sci. 82 (2) (2016) 174–190.
[11] K. Li, Optimal task dispatching on multiple heterogeneous multiserver systems with dynamic speed and power management, IEEE Trans. Sustain. Comput. 2 (2) (2017) 167–182.
[12] K. Li, X. Tang, K. Li, Energy-efficient stochastic task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2867–2876.
[13] 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.
[14] S. Liu, J. Lu, Q. Wu, Q. Qiu, Harvesting-aware power management for real-time systems with renewable energy, IEEE Trans. Very Large Scale Integration (VLSI) Syst. 20 (8) (2012) 1473–1486.
[15] A. Mishra, A.K. Tripathi, Energy efficient voltage scheduling for multi-core processors with software controlled dynamic voltage scaling, Appl. Math. Model. 38 (14) (2014) 3456–3466.
[16] T. Mladenov, S. Nooshabadi, K. Kim, Implementation and evaluation of Raptor codes on embedded systems, IEEE Trans. Comput. 60 (12) (2011) 1678–1691.
[17] A. Naithani, S. Eyerman, L. Eeckhout, Reliability-aware scheduling on heterogeneous multicore processors, in: 2017 IEEE International Symposium on High Performance Computer Architecture, HPCA, 2017, pp. 397–408, http://dx.doi.org/10.1109/HPCA.2017.12.
[18] X. Tang, K. Li, R. Li, B. Veeravalli, Reliability-aware scheduling strategy for heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 70 (9) (2010) 941–952.
[19] 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.
[20] 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.
[21] Y. Xiang, S. Pasricha, Soft and hard reliability-aware scheduling for multicore embedded systems with energy harvesting, IEEE Trans. Multi-Scale Comput. Syst. 1 (4) (2015) 220–235.
[22] 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. Inf. 13 (4) (2017) 1629–1640.
[23] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, K. Li, Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Sustain. Comput. 3 (3) (2018) 167–181.
[24] 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. Inf. 13 (3) (2017) 1068–1078.
[25] 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.
[26] 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.
[27] 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.
[28] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inform. Sci. 319 (2015) 113–131.
[29] Y. wen Zhang, H. zhen Zhang, C. Wang, Reliability-aware low energy scheduling in real time systems with shared resources, Microprocess. Microsyst. 52 (2017) 312–324.
[30] B. Zhao, H. Aydin, D. Zhu, Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013) 23.
[31] L. Zhao, Y. Ren, K. Sakurai, Reliable workflow scheduling with less resource redundancy, Parallel Comput. 39 (10) (2013) 567–585.
[32] 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.
[33] N. Zheng, Z. Wu, M. Lin, L.T. Yang, Enhancing battery efficiency for pervasive health-monitoring systems based on electronic textiles, IEEE Trans. Inf. Technol. Biomed. 14 (2) (2010) 350–359.
[34] D. Zhu, Reliability-aware dynamic energy management in dependable embedded real-time systems, ACM Trans. Embedded Comput. Syst. (TECS) 10 (2) (2010) 26.
[35] D. Zhu, H. Aydin, Reliability-aware energy management for periodic real-time tasks, IEEE Trans. Comput. 58 (10) (2009) 1382–1397.

Hongzhi Xu is currently working toward the Ph.D. degree in computer science and engineering at Hunan University, Changsha, China. He is an Associate Professor in the College of Software, Jishou University, Zhangjiajie, China. His research interests include embedded computing systems and cyber–physical systems.

Renfa Li is a Professor of computer science and electronic engineering, and the Dean of College of Computer Science and Electronic Engineering, Hunan University, China. He is the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. He is also an expert committee member of National Supercomputing Center in Changsha, China. His major interests include computer architectures, embedded computing systems, cyber–physical systems, and Internet of things. He is a member of the council of CCF, a senior member of IEEE, and a senior member of ACM.

Chen Pan received the M.S. degree in Telecommunication Engineering from Hunan University, Changsha, China, in 2012. He received M.S. degree in Electrical Engineering from Oklahoma State University, Stillwater, OK, USA, in 2017. He is currently a fifth-year Ph.D. student in Electrical and Computer Engineering at Swanson School of Engineering, University of Pittsburgh, PA, USA. His current research interests include low-power embedded systems, non-volatile memory optimization, low-power IoT edge computing, wireless sensor network, energy harvesting, and game theory.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber–physical systems. He has published over 485 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.