Contents lists available at ScienceDirect

Electrical Power and Energy Systems

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

Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems

Longxin Zhang [a], Kenli Li [a,*], Keqin Li [a,b], Yuming Xu [a]

[a] College of Computer Science and Electronic Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 6 August 2014
Received in revised form 23 September 2015
Accepted 25 November 2015

Keywords:
Energy management
Parallel application
Reliability enhancement
Scheduling algorithm

ABSTRACT

Voltage scaling is a fundamental technique in the energy efficient computing field. Recent studies tackling this topic show degraded system reliability as frequency scales. To address this conflict, the subject of reliability aware power management (RAPM) has been extensively explored and is still under investigation. Heterogeneous Computing Systems (HCS) provide high performance potential which attracts researchers to consider these systems. Unfortunately, the existing scheduling algorithms for precedence constrained tasks with shared deadline in HCS do not adequately consider reliability conservation. In this study, we design joint optimization schemes of energy efficiency and system reliability for directed acyclic graph (DAG) by adopting the shared recovery technique, which can achieve high system reliability and noticeable energy preservation. To the best of our knowledge, this is the first time to address the problem in HCS. The extensive comparative evaluation studies for both randomly generated and some real-world applications graphs show that our scheduling algorithms are compelling in terms of enhancement of both system reliability and energy saving.

Introduction

During the past decades, resource management and task scheduling on computer system have been well addressed. With the advent of multiprocessor and multi-core, energy management has become one of the hottest areas due to the dramatically increasing power consumption of modern heterogeneous computational systems, which consist of various set of resources. In reducing energy consumption field, one of the recent main challenges is to accomplish speed up on parallel application with regard to directed acyclic graphs (DAG). It's generally considered that task scheduling problem is an NP-complete problem [1].

The new generation of processors provides a substantial performance boost meanwhile, the energy consumption is not a well addressed problem. Although a lot of effort is being spent on saving energy of processor, the efficient and effective method has yet to be developed. The emergence of severe ecological, economic and technical issues are caused by the increasing energy consumption. Hence, it is not very hard to image the size of adverse environmental footprint left by HCS. With the growing advocacy of green computing systems, the issue of energy efficiency has recently attracted extensive research. Hardware technologies [2], energy-efficient monitors, low-power microprocessors, processor contains multiple processing element cores and a selective-associative cache memory, are employed to deal with the energy consumption problems. Comprehensive surveys can be referred to references [2–4].

Dynamic voltage frequency scaling (DVFS) is a fundamental energy management technique for modern computing systems. It has been proven that the dynamic power consumption is a strictly convex function of the CPU speed. Reducing clock speed or supply voltage of processor at active state to achieve energy saving are examples for energy consumption reduction. Various energy efficient scheduling and resource management strategies have developed for sustainable computing via taking advantage of DVFS technique. Nevertheless, the excellent schemes for these scope of applications are limited to unique processor systems [5], battery based embedded systems [6] or homogeneous computing systems [7–9].

Generally speaking, task scheduling is to select a proper processor for each task which complies with the precedence rules. In the meanwhile, the main goal is to minimize scheduling length. One relative research is the famous Dynamic-Level Scheduling (DLS) algorithm [10] for homogeneous systems. During the past few years, some list scheduling algorithms are proposed for heterogeneous

* Corresponding author.
E-mail addresses: longxinzhang@hnu.edu.cn (L. Zhang), lkl@hnu.edu.cn (K. Li), lik@newpaltz.edu (K. Li), xxl1205@163.com (Y. Xu).

http://dx.doi.org/10.1016/j.ijepes.2015.11.102
0142-0615/© 2015 Elsevier Ltd. All rights reserved.

systems, such as Constrained earliest finish time (CEFT) algorithm [11], Critical-Path-On-a-Processor (CPOP) algorithm [12], and heterogeneous earliest finish time (HEFT) algorithm [12]. HEFT and CPOP are famous for its low-complexity and performance-effective capability. CEFT is capable of accomplishing tasks earlier without sacrificing too much time complexity.

Another important challenge is to avoid communication between processors while transferring data for task nodes with dependence. An advance technique, named node duplication, is considered as an effective solution to the stated problem. Based on this, some recently reported researches, i.e., HCPFD [13] and HLD [14], are proposed to achieve better performance. The main idea of these works is to duplicate the immediate parent of each task if possible and necessary so as to diminish unnecessary communication between parent and child nodes, which are allocated to the distinct processors. The extensive comparisons, in [13,14], demonstrate that algorithms with idea of duplication significantly surpass other strategies, such as DLS [10] and HEFT [12].

Recently, Zhang et al. [15] presented algorithms to enhance task reliability when a dynamic voltage scaling technique is applied to achieve low power consumption in heterogeneous computing systems. Xu et al. [16] proposed a hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. Huang et al. [17] developed novel heuristic speculative execution strategies in heterogeneous distributed environments. Yan et al. [18] developed an intelligent particle swarm optimization (IPSO) algorithm for short-term traffic flow predictors to tackle time-invariant assumptions in the on-road sensor systems.

Unfortunately, most of these approaches are on the basis of simple system models. It is inadequate in two respects. On one hand, the employed system model does not precisely reflect the real parallel computation system. On the other hand, the assumption that each task will always execute successfully is not occur with real application. For various reasons, transient faults arise more frequently than permanent faults. In particular, transient fault is a small probability event but inevitably occurs in the real situation. The side effect of transient fault is that, it may cause seriously adverse hurt on the running application, sometimes it even leads to fatal errors. Low-power consumption and high-system reliability, availability, and utility are major concerns of modern high-performance computing system development. There are a few studies focusing on providing fault tolerance capabilities to embedded systems [19]. Studies in [20,21] explore the interplay between reliability and energy consumption. However, they are exclusively confined to the field of real-time system. With regard to HCS, Tang et al. [22] developed a reliability-driven scheduling architecture and devised a reliability-aware scheduling scheme for precedence constrained tasks. Lee and Zomaya [23] proposed two energy-conscious scheduling algorithms which effectively balance the quality of schedules and energy consumption with DVFS technique. Li [24] analyzed the performance of heuristic power allocation and scheduling algorithms for precedence constrained sequential tasks. Notwithstanding, none of them incorporates energy consumption and reliability together. In most cases, the smaller scheduling length is not the most important objective. The co-management of system reliability and energy saving has drawn researchers' attention only very recently [19,25,26]. Zhao et al. [19] adopted recovery technique to reexecute fault application for tasks with dependence, but it is only applied to unique core.

In this study, we develop energy efficient and reliability conservation for precedence constrained tasks with shared deadline in heterogeneous system. It's also a combinatorial optimization problem. To the best of our knowledge, this is the first time the problem is dealt with in HCS. Two kinds of techniques, dynamical voltage scaling and shared recovery are employed. Our scheduling problems comprise five nontrivial subproblems, namely, precedence constraining, deadline constraining, energy conservation, reliability maximizing and task recovery.

- *Precedence Constraining*. Compared to independent task set, tasks with precedence constraints make devise and analysis of heuristic scheduling algorithms particularly complicated.
- *Deadline Constraining*. The entire task set share a common deadline.
- *Energy Conserving*. Tasks should be supplied with appropriate powers and energy efficient execution speeds, such that the schedule length is modest and the energy consumption is minimal.
- *Reliability Maximizing*. Tasks should be run at a relatively high speed without exceeding the maximum frequency of processor, such that the system reliability can achieve an optimal value.
- *Task Recovery*. After finish each task in the prior queue, error detection would be started up. Once a fault occurs, then the error will be recovered at the maximum frequency.

The above subproblems should be solved efficiently so that heuristic algorithms with overall fine performance can be explored. By adopting DVFS technique, three algorithms will be presented in this paper: Shared Recovery for reliability aware of Heterogeneous Earliest Finish Time (SHRHEFT) algorithm, Shared recovery for Reliability aware of Critical Path On a Processor (SHRCPOP) algorithm and Shared Recovery for Energy Efficiency and system Reliability Maximization (SHREERM) algorithm. Each of these three algorithms comprises three phases. Task priority establishment phase builds a proper topological order for the application task. Frequency selection phase chooses an energy efficient frequency to execute each task. Processor assignment phase allocates the candidate task to a suitable processor. Finally, task recovery phase detects the transient fault and recovers the error task, so as to get higher system reliability with lower total energy consumption.

The paper proceeds as follows: Section 'Models' gives a briefly introduction of models used in this paper which include the system, power, energy and reliability. Based on reliability and energy model, SHRMEC algorithm and revised algorithms SHRHEFT and SHRCPOP are presented in Section 'The proposed algorithms'. Extensive experiment results are discussed in Section 'Experiment s and results'. Section 'Conclusion' summarizes the conclusion and shows the future direction of this work.

Models

In this section, we provide a brief background on the system, power, fault, and application models used in this paper.

System model

The system model used in this paper is composed of a set P that includes $|p|$ heterogeneous cores/processors. All the cores are encapsulated in a chip. Each core of them is available for DVFS technology; namely, each core can run at different speed (i.e., different supply voltage levels) in any time. For each processor p belongs to the set P, with $|f|$ available frequency levels (*AFLs*), follows a random and uniformly distribution among the four different sets of operation voltages/frequency. As clock frequency is switched, overheads take an inappreciable amount of time (e.g. 10–150 μs) [27,28], and this overheads are neglected in our study. The communications among inter-processor are supposed to perform at the same speed on all links without contentions. Our paper is based on the premise that the target system consists of a set of fully connected processors, which implies that each processor has a direct communication link to every other processor. Inter-processor communication is performed by a dedicated communication subsystem, in a way that is completely free of contention.

Power model

In DVFS technique, the clock frequency is reduced alongside with the supply voltage for the approximate linear relationship between the supply voltage and operation frequency [29]. For the promising capability for energy saving, DVFS technique is adopted in our study. It enables processor element dynamically adjust available frequency levels. To present system-level power model, we adopt the classic one proposed in [30], and the system power consumption is given as following [31]:

$$P = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{eff} f^{\alpha}), \tag{1}$$

where P_s is the static power consumption, P_{ind} refers to the frequency-independent active power, and P_d represents the frequency-dependent dynamic power. The static power term, includes the power to maintain the basic circuits, keeps the clock working and the memory in sleep mode. It can be removed only by turning off the whole system. P_s is a constant, independent of system operation frequency (i.e., the power consumption occurs while accessing external devices like main memory, I/O and so on). It can be decreased to a very small value by setting the system to standby mode [29]. P_d is the dynamic power dissipation, the dominant component of energy consumption in widely popular CMOS technology. It can be given by $P_d = C_{eff} \cdot V_{dd}^2 \cdot f$, where C_{eff} is the effective loading capacitance, V_{dd} is the supply voltage and f is the clock frequency. Since $f \propto v^{\gamma}$, $(0 < \gamma < 1)$ [24], in other words, $v \propto f^{1/\gamma}$, we reckon that the frequency dependent active power consumption is $P_d \propto f^{\alpha}$, where $\alpha = 1 + 2/\gamma \geqslant 3$. In our studies, we have $P_d = C_{eff} \cdot f^{\alpha}$. And $\hbar$ indicates the system mode and represents whether active power consumption occurs present. Particularly, $\hbar = 1$ signifies that the system is currently active. Otherwise, $\hbar = 0$ refers to that system is in sleep mode. In what follows, all frequencies are normalized with respect to the maximum frequency $f_{\max}$ (i.e., $f_{\max} = 1.0$). And the energy consumption of task n_i can be calculated according to the following expression:

$$E_i(f_i) = P_{ind_i} \cdot \frac{c_i}{f_i} + C_{eff} \cdot c_i \cdot f_i^2, \tag{2}$$

where c_i is the computational cost at executing frequency of f_i.

Application model

A parallel application program consists of a precedence constrained tasks can be represented by a directed acyclic graph (DAG). A DAG, $G = <N, E>$, where N is the task set which comprises $|n|$ tasks that can be executed on any available processors. Set E is composed of the edges which represent tasks precedence constrains. An edge $w_{i,j} \in E$ between task nodes i and j, each of which performs on different processor, denotes the intertask communication.

For each node n_i in a given task graph, the direct predecessors of which are denoted by $parent(n_i)$. There is set $parent(n_i) = \{\forall n_p \in N | e_{p,i} \in E\}$. And its direct successors are denoted as $child(n_i)$. If a task without any predecessor, namely, $parent(n_i) = \varnothing$, is called an entry task. Likewise, if a task without any successor, namely, $child(n_i) = \varnothing$, is called an exit task. Without loss of generality, we assume that the DAG in our study exactly exists or can be transformed to one entry task n_{entry} and one exit task n_{exit}.

The weight on task n_i is denoted as w_i, which represents the computation cost. In addition, the execution time of task n_i on processor p_j refers to $w_{i,j}$ and its average computation cost is denoted by $\bar{w}_i$. Similarly, the weight $c_{i,j}$ is assigned to an edge represents the communication cost between two tasks n_i and n_j. However, the communication takes place when the two nodes are scheduled to two distinct processors. In other word, there is no communication cost when the two nodes are assigned to the same processor (see Table 1).

Consider the graph with the eleven nodes as shown in Fig. 1, the edges, which are labeled with different weights, reflect the communication cost of corresponding nodes in different processor. In our study, the target system comprises of a set P of $|p|$ heterogeneous cores which are fully interconnected. And each core is DVFS enabled. As shown in Table 2, each core can run at different available frequency levels (AFLs). For each processor element $p_i \in P$, the voltage-relative frequency AFLs is selected randomly among the distinct sets. The execution costs of each node on different processor are shown in Table 3, which under the condition of each task run at available maximum frequency. According to the previous study [32], frequency switching takes a negligible amount to time, about 189–300 µs. These overheads are not considered in this study while applying the DVFS technique. Besides, communication among processor elements are also considered to perform at the same speed on all links without the limitation of bandwidth.

Fault model

During the course of application execution, a fault maybe hard to avoid owing to various reasons, such as hardware failure, software bugs, devices exposed to extreme temperatures, and external interference. As a consequence, transient fault occurs more frequently than permanent failure [33,34]. In this study, we pay more attention to transient faults, and devise a feasible and efficient scheduling with DVFS technology to maximize overall system reliability.

Extensive works have been done on fault management. The transition fault is modeled by Poisson distribution with an average arrival rate λ [20]. Following with most previous studies, we consider the transition fault happens during the execution of each task as independent. Nevertheless, with the effect of dynamic voltage

Table 1
Definitions of notations.

Notation	Definition
N	A set of task nodes
P	A set of processors
V	A set of supply voltages
F	A set of supply frequencies
$w_{i,j}$	The computation cost of task $n_i \in N$ executed on processor $p_j \in P$
$c_{i,j}$	The communication cost between nodes n_i and n_j
$\bar{w}_i$	The average computational time of a task when it is executed on different processors
$child(n_i)$	The set of immediate successors of task n_i
$parent(n_i)$	The set of immediate predecessors of task n_i
$EST(n_i, p_j)$	The earliest execution start time of task n_i on processor p_j
$EFT(n_i, p_j)$	The data ready time of task n_i on processor p_j
$AFLs$	Available frequency levels
D_{share}	The common deadline of the task set
$D_{i,j}^e$	The effective deadline of task node n_i execute on processor p_j
SHRHEFT	Shared Recovery for reliability aware of Heterogeneous Earliest Finish Time
SHRCPOP	Shared Recovery for reliability aware of Critical Path On a Processor
SHREERM	Shared Recovery for Energy Efficiency and system Reliability Maximization
S_n	The number of nodes in a specific task graph

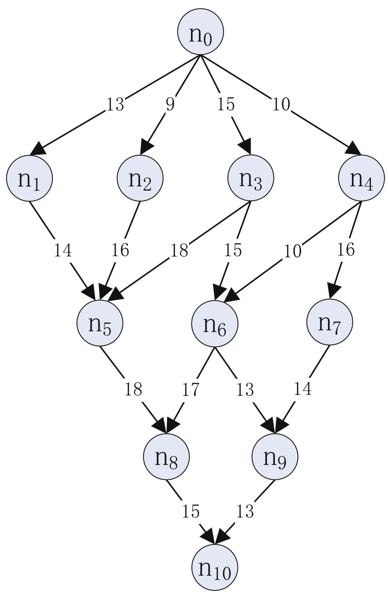

Fig. 1. A simple precedence-constrained application DAG.

Table 2
Voltage-relative frequency pairs.

Level	Pair 1		Pair 2		Pair 3	
	Voltage	Frequency	Voltage	Frequency	Voltage	Frequency
0	1.75	1.00	1.50	1.00	2.20	1.00
1	1.50	0.80	1.40	0.90	1.90	0.85
2	1.40	0.70	1.30	0.80	1.60	0.65
3	1.20	0.60	1.20	0.70	1.30	0.50
4	1.00	0.50	1.10	0.60	1.00	0.35
5	0.90	0.40	1.00	0.50		
6			0.90	0.40		

Table 3
Computation cost with AFL 0.

Node num	P1	P2	P2
0	11	16	12
1	14	12	11
2	14	12	15
3	18	12	8
4	15	20	17
5	12	14	15
6	14	15	13
7	16	17	10
8	17	18	16
9	16	15	12
10	14	12	11

and frequency scaling, transition fault's average arrival rate depends on the system processing frequency f, and v is the corresponding voltage. Hence the fault rate can be modeled as follows:

$$\lambda(f) = \lambda_0 \cdot g(f). \tag{3}$$

Let $f_{\max}$ be the average fault rate with the corresponding maximum frequency(and operation voltage is $v_{\max}$). Specially, we can derive $g(f_{\max}) = 1$.

Traditionally, it has been recognized as an exponential relationship between the transient fault rate and the circuit's critical cost [35]. We adopt the exponential model proposed in [30] for our scheduling model and experiment analysis. It can be expressed as:

$$\lambda(f) = \lambda_0 \cdot g(f) = \lambda_0 \cdot 10^{\frac{d(1-f)}{1-f_{\min}}}, \tag{4}$$

where λ_0 stands for the average fault rate as mentioned before. d is a constant and greater than zero, which represents the degree of fault rate to frequency and voltage scaling. It can be easy to get that the fault rate increases exponentially as the frequency decreases for energy conservation. In other word, $\lambda(f)$ is a strictly decreasing function. Hence, the maximum average fault rate is $\lambda_{\max} = \lambda_0 \cdot 10^d$ with corresponding available minimum frequency.

To ensure the feasibility and high efficiency of an algorithm, the heuristic scheduling strategy must be adhered strictly to the precedence constraint. The following definitions are very important to the algorithms which are presented in the next section.

Definition 1. The data ready time of a node $n_i \in N$ on processor $p_j \in P$ is

$$DRT(n_i,\ p_j) = \max_{n_k \in pred(n_i)} \{EFT(n_k + c_{k,i},\ Available(p_j))\}, \tag{5}$$

where $c_{k,i}$ is the communication cost between nodes n_j and n_k. This item is only required provided two tasks nodes are assigned to different processors.

Definition 2. The reliability of a task as the probability of executing the task successfully. The transient fault follows a Poisson distribution, the reliability of node n_i with the corresponding calculation cost c_i is [27]

$$R_i(f_i) = e^{-\lambda(f_i) \times \frac{c_i}{f_i}}, \tag{6}$$

where f_i denotes the processing frequency and c_i refers to the corresponding computational cost.

Definition 3. The system reliability R_{sys} denotes the reliability of the entire task set which consists of n tasks. Then the total reliability of system nodes is

$$R_{sys} = \Pi_{i=1}^{n} R_i(f_i). \tag{7}$$

Definition 4. The effective deadline $D_{i,j}^e$ of task node n_i on processor p_j indicates that n_i has to complete well before it. The expression can be stated as:

$$D_{i,j}^e = \begin{cases} D_{share}, & Succ(n_i, p_j) = \varnothing \\ \min\{D_{share} - D_{k,j}^e\}, & n_k \in Succ(n_i, p_j), \end{cases} \tag{8}$$

where $Succ(n_i, p_j)$ represents the task nodes mapped to the processor p_j, whose earliest start time are later than node n_i. And D_{share} denotes the common deadline of the task set.

Problem Description: Taking the negative effect of DVFS on system reliability into consideration, for frame-based dependent real-time tasks with individual deadlines and a common period that are represented by a DAG, the problem to be addressed is to minimize energy consumption while preserving the system original reliability without violating task's deadline and precedence constraints. Before presenting the details of our new shared-recovery based RAPM schemes, in what follows, we firstly illustrate the conservativeness of the existing individual-recovery based RAPM schemes through a concrete motivation example.

A motivational example

As shown in Fig. 1, an application with eleven tasks, the corresponding feasible schedule when all tasks run the maximum available voltage under the famous HEFT scheme, is given in Fig. 2(a). That is, there is no power management and no reliability awareness.

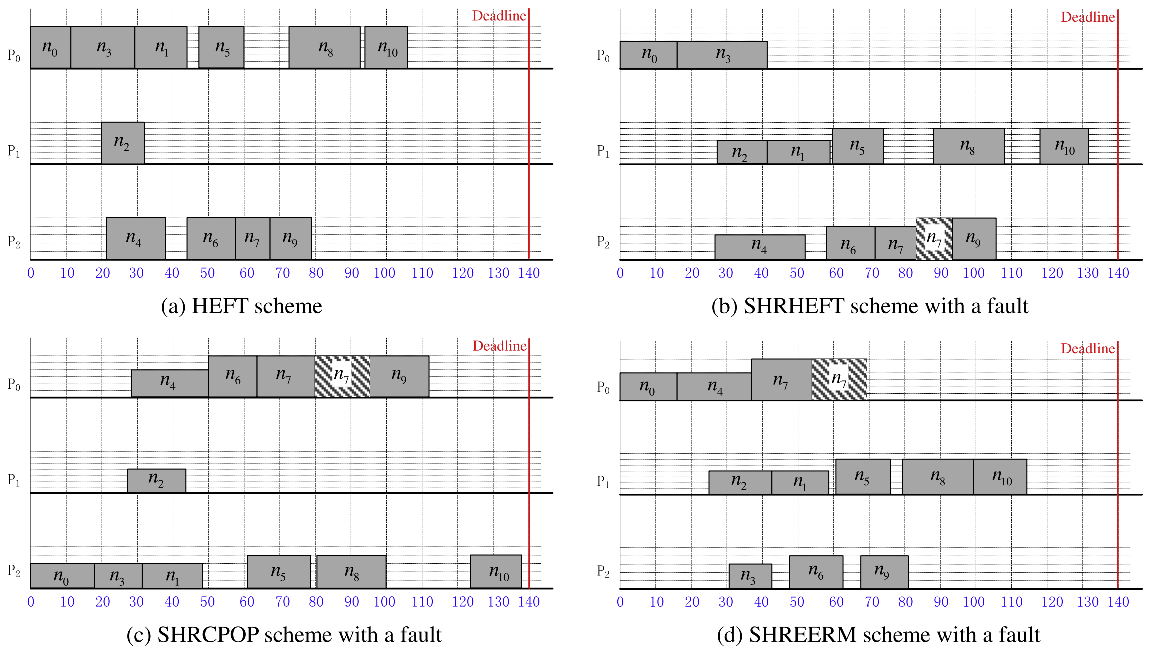

Fig. 2. Scheduling of task graphs in Fig. 1 under different algorithms.

Each task is performed in the increasing order of *blevel* [12] and the schedule must be guaranteed that each one's precedence constraints and the entire tasks' shared deadline are fulfilled. Assuming that the environment parameter is set as $\lambda_0 = 10^{-6}$, $d = 5$, and the minimal frequency is 0.1. As we can see from Fig. 2(a), the system still has access to 140–106 = 34 of slack, which can be employed to scale down the execution of tasks to gain energy saving. Furthermore, and also the most important, transient fault is always inevitable. Once a fault occurs during the performance of tasks, under such circumstances, it would lead to an error for application. To keep application efficient and promote the robustness of systems, measures must be taken to tackle the unexpected accident. Shared recovery (SHR) technique, first proposed by Zhao et al. [36] and the extension to DAG with precedence constraint on unique core at [19], has been proven to be efficient to enhance system reliability. We extended the SHR technique and applying it to our task set on multiple processors platform without violating a shared deadline constraint. In what follows, we define that each processor reserve a recovery block in case it encounters soft errors in the process of task scheduling. For instance, we can have one recovery task of size 11 at the very beginning. And the remaining slack can be utilized to scaled down the following tasks. The recovery is large enough to cover task n_0 once its soft error arises. After n_0 execute successfully at the scaled efficient frequency, then the recovery task size can be adjusted to 18 on processor 0, which is large enough to recover task n_3 by occupying part of the rest of slacks. This procedure repeats until all the tasks are scheduled. When soft errors are detected and the recovery block is employed to execute the error task at f_{max}. The remainder tasks are mapped to the incident processor, which turn into the mode of emergency, and run at the full frequency. If not, the system may not have sufficient slack to recovery the further error task and the original reliability can be ensured. As illustrate in Fig. 2(b), applying the SHRHEFT scheme, a scenario that an transient fault occurs while performing task n_7. Then the recovery block is used to recover n_7 and the following mapped task n_9 run at the full frequency. Besides, under other two strategies, SHRCPOP and SHRECRM, the corresponding Gantt charts are shown in Fig. 2(c) and (d) for the case that the error of task n_7 arises. In case that no soft fault occurs, the energy consumption of three strategy, SHRHEFT, SHRCPOP and SHRECRM are 321.86, 310.63, 301.86, respectively. When the transient default happens while performing task n_7, as captured in Fig. 2(b)–(d), the energy consumption associated with the three schema are 346.86, 350.63, 341.86 respectively. And the system reliability in such situation are 0.999993892441768, 0.999993646500958 and 0.999995209531709 respectively. By elaborating the all the single-error scenarios on each processor, the three algorithms perform better performance even when they encounter a transient fault while executing the tasks.

The proposed algorithms

System reliability and energy saving are two primary metrics in modern computing system. DVFS, which is regarded as one of the most efficient technologies, is used widely to reduce energy consumption. However, it also brings negative effect on system reliability. A meaningful study is to guarantee higher reliability for parallel tasks without violating their shared deadline constraint even when transient faults occur. In heterogeneous systems, parallel tasks scheduling contains two stages, i.e., task priority establishment and task-processor mapping. For the first stage, an upward rank is used to calculate the priority queue. Efficient strategies are used to map each task in the priority queue to a best fit processor while fulfilling the deadline constraint and processing the potential transient faults in the second stage.

This section devotes to develop three algorithms for addressing the above system reliability issue in task scheduling. The first algorithm to be proposed is the Shared Recovery for Reliability aware of Heterogeneous Earliest Finish Time (SHRHEFT) algorithm, which selects the best fit processor for each ready task to assure higher system reliability and obey the deadline constraint. The second algorithm to be presented is the Shared Recovery for reliability aware of Critical Path On a Processor (SHRCPOP) algorithm. Compared with SHRHEFT, the SHRCPOP is only different in the processor mapping. The tasks in the critical path of the application under SHRCPOP have the highest priority to be mapped to the same fastest processor. In the third one, namely Shared Recovery for Energy Efficiency and system Reliability Maximization (SHREERM) algorithm, an efficient combination of task-processor, which aims at energy saving and maximum reliability, is used in the pre-scheduling stage. After then, the slots which exist in the processors are reclaimed to save more energy.

Before expounding the details of three strategies, a feature with respect to the relationship between system reliability and energy efficient and some important phases firstly are presented.

Relationship between reliability and energy

In this section, we will present an Energy Efficient and Reliability Conservation (SHREERC) algorithm. The algorithm SHREERC, which will be introduced in the following, based upon the effective deadline, aims at achieving energy efficiency and reliability preservation in the condition of energy constrained without increasing makespan during the scheduling procedure.

Due to the frequency-independent active power, the power consumption no longer varies monotonically as the frequency increases. Observed from Eq. (2), it can be easily to deduce that $E_i(f_i)$ is a strictly convex function and reaches its extreme point at $f_i = f_{ee} = \sqrt[3]{P_{ind}/2C_{eff}}$ (energy efficient frequency) [31]. It can be deduced that, lower frequencies don't always take effect for energy saving and there must exists an optimal voltage-frequency pair to achieve minimum energy consumption. On the other hand, as processing frequency increases, the reliability of task improved monotonically. It is safely to draw such a conclusion that the high reliability and energy efficiency are almost two contradictive goals for scheduling. Designing a novel strategy for a tradeoff of them is a non-trivial work. And we will attempt to schedule tasks aim at enhancing system reliability and pursuing energy efficiency while guaranteeing common deadline constraint, transient faults diagnosis and soft errors treatment.

Critical phases

Each of the three algorithms which will be presented in the next part, is mainly composed of four phases. It involves the following stages: **task priority establishment phase** to provide a valid topological order of application task, **frequency selection phase** to choose a feasible and efficient frequency to perform the data ready task, **error detection and task recovery phase** to diagnose transient fault and reexecute the one encounter soft error, and **processor assignment phase** to allocate the candidate task to the "best" processor in the order of priorities and low energy consumption. In what follows, we introduce two important phases of them.

Task priority establishment phase: To meet the requirement of task scheduling, the prior order is established in this phase. Each task is set with its upward rank, *URank*, which is computed recursively according to

$$URank(n_i) = \overline{w_i} + \max_{n_j \in child(n_i)} (c_{i,j} + URank(n_j)), \tag{9}$$

where $child(n_i)$ is the set of immediate child of task node n_i. As the rank is computed recursively by traversing from the bottom of DAG to the top. Some literatures also call it bottom level. It should be apparent to draw such a conclusion $URank(n_i) = \overline{w_{exit}}$. *URank* is an effective approach to offer a topologic order to depend tasks. As shown in Table 3, the computation cost of each node represents concrete value, that running a task on specific processor at the fast speed (AFL 0). Akin to *URank*, the downward rank *DRank* is defined as

$$DRank(n_i) = \max_{n_j \in parent(n_i)} (\overline{w_j} + c_{i,j} + DRank(n_j)), \tag{10}$$

where $parent(n_i)$ is the direct parents of task node n_i. Provided there is no parent for the entry node, it is easy to conclude that the *DRank* of entry node is equal to zero. According to the computation cost given by Table 3, a priority queue for the simple DAG shown in Fig. 1 is maintained for the following three various algorithms. The values of task priority using the *DRank* and *URank* method are summarized in Table 4.

Definition 5. The immediate neighboring frequency f_{in} of f_{ee} processor p_j, is the one which consumes less energy.

According to expression $f_{ee} = \sqrt[3]{P_{ind}/2C_{eff}}$, f_{ee} only depends on P_{ind}. The particular P_{ind} value for each task obeys uniform distribution and is determined randomly with its range of 0.2–2 in this study. So it is not difficult to find that f_{ee} varies on different processors for each task n_i.

Frequency selection: As stated above, the "best" frequency-voltage pairs to achieve lower energy and higher reliability appear are nearest neighbors of f_{ee} on processor for each task node. A binary search tree is used to find these two immediate neighbors. The whole process takes a time complexity of $O(\lg n)$. Where n is the number of task nodes. Calculate the active energy using Eq. (2), the neighbor frequency which consumes less energy will be selected to perform the data ready task node.

The proposed algorithms

Inspired by two famous algorithms HEFT and CPOP [12], another two new strategies are to be developed so as to yield an efficient scheduling even the processor has suffered a soft error. The first one to be discussed is reliability aware of heterogeneous earliest finish time (SHRHEFT).

The SHRHEFT algorithm

The SHRHEFT algorithm is an application which schedule task with bounded number of heterogeneous processors. As shown in Algorithm 1, the *URank* and energy-efficient frequency f_{ee} of each task are computed in the first two steps. The priority queue is created in Step 3. Steps 4–18 are the outside loop of SHRHEFT to ensure the effective scheduling of each task. An appropriate immediate neighboring frequency is selected at Step 10, and temporarily marked it in the memory. After the end of the inside loop of all the processors from Steps 6–12, the combination involves a most suitable frequency and a "best" fit processor is obtained. Then assigning the candidate schedule task node to the marked best processor, with the executing frequency in Step 14. Each time perform loop in Step 10, the new marked frequency and processor indicates that a new best scheduling alternative is found. While a task is scheduled, the **SoftErrorDectect** procedure is invoked to detect the transient fault and deal with it in Step 13. The reliability and energy consumption of task are calculated in Steps 15 and 16, respectively. In SHRHEFT algorithm, it takes $O(\lg f)$ time for frequency selection phase in Step 10. The SHRHEFT algorithm has a time complexity of $O(n \times p \times \lg f)$ for n task nodes and p processors. Each processor consists of f levels DVFS enable frequencies.

Table 4
Task priorities.

Node no.	*URank*	*DRank*
0	134.67	0.00
1	102.33	26.00
2	105.67	22.00
3	106.67	28.00
4	102.67	23.00
5	76.00	58.67
6	75.33	55.67
7	68.00	56.33
8	44.33	90.33
9	39.67	84.67
10	12.33	122.33

Algorithm 1. SHRHEFT

```
Input: A DAG $G = <N, E>$ and a set $P$ of DVFS available processors
Output: A schedule $S$ of $G$ onto $P$
 1: compute URank of $n_i \in N$ by traversing graph from the exit node
 2: compute energy-efficient frequency for each node in set $N$
 3: sort the tasks in a non-increasing order by $URank(n_i)$ value and establish a priority queue $Queue_{URank}$ for the sorted tasks
 4: while the priority queue $Queue_{URank}$ is not empty do
 5:   $n_i \leftarrow$ the head node in $Queue_{URank}$;
 6:   for each processor $p_j \in P$ do
 7:     if errors had happened or $D^e_{i,j} = D_{share}$ then
 8:       $f_{j,k} \leftarrow 1$
 9:     else
10:       find the immediate neighboring frequency $f_{in}$ of $f_{ee}$ on processor $p_j$ for task $n_i$, mark the one which consumes less energy
11:     end if
12:   end for
13: call SoftErrorDetect$(n_i, p_j)$ ▹ error detection and task recovery once it happens
14: assign the marked frequency $f_{in}$ to task $n_i$ on the marked processor
15: compute the reliability of task $n_i$ using Eq. (6)
16: compute the energy consumption for task $n_i$ using Eq. (2)
17: delete the head node $n_i$ in $Queue_{URank}$
18: end while
```

Procedure SoftErrorDetect(n_i, p_j)

For selecting an optimal frequency of task n_i on processor p_j, we assume that only single fault scenario exists in each processor during task scheduling. It infers that at most one error can be encountered for each processor. Each time a task has executed, the system invokes SoftErrorDetect(n_i, p_j) procedure. Once the algorithm detects that transient fault occurs and it leads to failure of task execution, then recovery the fault task at once with the maximum available frequency, otherwise, nothing would be done.

Procedure. SoftErrorDetect(n_i, p_j)

```
Input: task $n_i$ and processor $p_j$
Output: soft error detection and error task recovery if needed
1: detect whether task encounters soft error
2: if soft error arises then
3:   recover task $n_i$ on $p_j$ with the maximum frequency
4: end if
```

The SHRCPOP algorithm

The second algorithm is reliability aware of critical path on a processor with shared recovery technology (SHRCPOP). Algorithm 2 shows the pseudo-code of SHRCPOP. In this algorithm, the *DRank* and *URank* of each node is firstly computed. The priority queue of the entire task set is established on the basis of nodes' *URank*. An important sorted list named $list_{CP}$, consists of the task node on the critical path with the minimal summation of *DRank* and *URank*, is set up after the first loop at Step 8. As the energy-efficient frequency f_{ee} of each node only depends on its frequency-independent active power, and the procedure is finished at Step 9. When the condition that the critical processor which take the least time is fulfilled, all the tasks in $list_{CP}$ are get ready to map onto it. The critical processor which take the least time to execute all the tasks is selected at Step 10. And Steps 12–29 are the main loop body. While scheduling a task from the priority queue, the node on critical path would only be assigned to the critical processor. Then pick up a "best" immediate neighboring frequency f_{in} for it. The node in other path can also be assigned to critical processor provided there is free a slack time slot, otherwise, it will be assigned to a non-critical processor. The latter procedure is the same with the CP nodes. When soft error is discovered for any of the executed task, then reexecute it at the f_{ee}. For all data ready tasks, the feasible frequency searching takes time $O(\lg f)$. So the complexity of the SHRCPOP algorithm is $O(n \times p \times \lg f)$. The detailed definition of each item in the expression is the same as mention before.

Algorithm 2. SHRCPOP

```
Input: A DAG $G = <N, E>$ and a set $P$ of DVFS available processors
Output: A schedule $S$ of $G$ onto $P$
 1: compute $URank(n_i)$ of each node $n_i \in N$ by traversing graph upward from the exit node
 2: compute $DRank(n_i)$ of each node $n_i \in N$ by traversing graph downward from the entry node
 3: $|CP| \leftarrow Max\{\forall n_i \in N | URank(n_i) + DRank(n_i)\}$
 4: for each node $n_i \in N$ do
 5:   if the summation of $URank(n_i)$ and $DRank(n_i)$ equals to $|CP|$ then
 6:     add node $n_i$ to CP set $list_{CP}$
 7:   end if
 8: end for
 9: compute energy-efficient frequency for each node in set $N$
10: select the CP processor $P_{CP}$ which has the minimal summation computation cost of nodes in $list_{CP}$
11: sort the tasks in a non-increasing order by $URank(n_i)$ value and establish a priority queue $Queue_{URank}$ for the sorted tasks
12: while not all nodes in $Queue_{URank}$ have been scheduled do
13:   $n_i \leftarrow$ the head node in $Queue_{URank}$
14:   if $n_i \in list_{CP}$ then
15:     assign processor $P_{CP}$ to $n_i$, and compute the immediate neighboring frequency $f_{in}$ of $f_{ee}$ on processor
16:   else
17:     assign the processor which consumes less energy with immediate neighboring frequency of $f_{in}$ its $f_{ee}$ for $n_i$
18:   end if
19:   if errors had happened or $D^e_{i,j} = D_{share}$ then
20:     $f_{j,k} \leftarrow 1$
21:   else
22:     find the immediate neighboring frequency $f_{in}$ of $f_{ee}$ on processor $p_j$ for $n_i$ task, mark the one which consumes less energy
23:   end if
24:   call SoftErrorDetect$(n_i, p_j)$ ▹ error detection and task recovery once it happens
25:   assign task $n_i$ to the marked processor and specify its executing frequency $f_{in}$
26:   compute reliability of the task node $n_i$ using Eq. (6)
27:   compute energy consumption of the task node $n_i$ using Eq. (2)
28:   delete the head node $n_i$ in $Queue_{URank}$
29: end while
```

The SHREERM algorithm

Algorithm 3. SHREERM

```
Input: A DAG $G = <N, E>$ and a set $P$ of DVFS available processors
Output: A schedule $S$ of $G$ onto $P$
 1: compute $URank(n_i)$ of each node in DAG $G$
 2: sort task $N$ in a non-increasing order by $URank(n_i)$ to establish priority queue $Queue_{URank}$
 3: while the priority queue $Queue_{URank}$ is not empty do
 4:   $n_i \leftarrow$ the head node in $Queue_{URank}$
 5:   for each processor $p_j$ in set $p$ do
 6:     compute the nearest frequency level of processor $p_j$ which has the minimal energy consumption
 7:     search the best combination of processor $p_j$ and frequency level of which keeps the max value $EERM(n_i, p_j, f_k)$
 8:   end for
 9:   if errors had happened or $D^e_{i,j} = D_{share}$ then
10:     $f_{j,k} \leftarrow 1$
11:   else
12:     find the immediate neighboring frequency $f_{in}$ of $f_{ee}$ on processor $p_j$ for $n_i$ task while guaranteeing shared deadline constraint, mark the one which consumes less energy
13:   end if
14:     assign the best combination of $p_j$ to node $n_i$
15:   call SoftErrorDetect$(n_i, p_j)$ $\triangleright$ error detection and task recovery once it happens
16:   compute the node reliability of task $n_i$
17: end while
18: let $S'$ denotes the scheduling derived from the above procedure
19: while the scheduling list $S'$ is not empty do
20:   $v'_i \leftarrow$ the head node in scheduling list $S'$
21:   for each processor $p'_j$ in set $P$ do
22:     while there is an available slack slot in the processor which satisfies the precedence priority of tasks do
23:       compute makespan, node reliability
24:       if makespan does not increase, node reliability promotes and there is an available time slot to accommodate $v'_i$ then
25:         replace $v'_i$ with the better combination processor $p'_j$ and frequency $f'_k$
26:       end if
27:       update node reliability and compute the node energy consumption
28:     end while
29:   end for
30: end while
```

Akin to the above two algorithms, another novel algorithm, shared recovery for energy efficiency and system reliability maximization (SHREERM) algorithm, is composed mainly of four phases. As described in Algorithm 3, the priority queue of task phase comprises Steps 1 and 2. The topological order of task is established in this phase. The optimal frequency selection phase contains Steps 5–8 are at the inner loop body of algorithm. Enlightened by the energy-conscious mind proposed in [23], the energy efficiency reliability maximum (EERM) strategy is defined as

$$\mathrm{EERM}(n_i,p_j,f_{j,k},p_l,f_{l,m}) = -\left(\frac{R(n_i,p_j,f_{j,k})-R(n_i,p_l,f_{l,m})}{R(n_i,p_j,f_{j,k})}\right) + \left(\frac{E(n_i,p_j,f_{j,k})-E(n_i,p_l,f_{l,m})}{E(n_i,p_j,f_{j,k})-\min\{E(n_i,p_j,f_{j,k})-E(n_i,p_l,f_{l,m})\}}\right). \quad (11)$$

It is exploited to allocated most favorable processor and frequency combination, the one with maximum value, for each data ready task node. As describes in Eq. (11), n_i represents the data ready task which in the head of priority queue, $f_{j,k}$ and $f_{l,m}$ are the available discrete frequency on the candidate processors p_j and processor p_l, respectively.

Steps 9–13 are included to implement the third phase. Node in priority queue is assigned to the marked processor with the selected optimum frequency at this stage. Step 15 is concerned to handle the execution error. As slack occurs inevitably due to the interprocessor communication for DAG, Steps 18–29 are involved to make full use of the time slots among processors, without prolonging schedule length and hurting system reliability. This procedure can be regarded as local optimum without increasing computational complexity. So the complexity of SHREERM algorithm is $O(2 \times n \times p \times \lg f)$.

Experiments and results

To evaluate the performance of these scheduling algorithms, two extensive explored sets of task graphs, randomly generated and real-world applications, are employed in our experiment. The real parallel applications involve the Fast Fourier Transformation (FFT), the Laplace, and Gaussian-elimination. In addition, a large number of variables are made for these task sets. It makes more comprehensive in some cases.

Specially, the random task graph set consists of 100 base task graphs, generated with six different sizes, and includes five kinds of dissimilar CCRs and five distinctly heterogeneous processors setting. The particular parameters are illustrated in Table 5. For each configuration, the specific P_{ind} value is determined randomly according to uniform distribution among the range of [0.2,2]. For simplicity, the value of C_{eff} is set to one throughout the whole experiment. With regard to robustness, provided that there is no special provision for reliability, it would cause certain serious error once transient faults occur. To simulate the occurrence of transient fault event and estimate the performance of the algorithms, every particular task runs a thousand times. The experiment results are the average value of outputs.

The experiments are performed on a workstation at National Supercomputing Center in Changsha. It contains an Intel Core i3-540 dual-core CPU, 8 GB DRAM, and 1 TB hard disk respectively. The proposed algorithms are implemented in the framework of the simulator [15] which runs with Windows 7 (64 bit OS) SP1. This simulator provides implementations of some famous algorithms, which involve CEFT, HLD, HCPFD, and so on. Well-known benchmarks including a large amount of FFT, Laplace, GE (shown below) graphs are used to verify the performance of the proposed algorithms.

In our study, each processor is supposed to execute a non-preemptive task from the prior queue. The computation and

Table 5
Configuration parameter for the random graphs.

Parameter	Possible values
CCR	0.2, 0.5, 2, 3, 10
Number of processors	3, 6, 9, 12, 15
Size	50, 100, 125, 150, 175, 200

communication costs, which follow a uniform distribution with the mean value to the specific average costs of computation and communication, respectively, are generated randomly for each task node. The occurrence of transient fault obeys a Poisson distribution, whose expression is given by Eq. (4), where $d = 3$ and $\lambda_0 = 10^{-9}$. For the three task sets of real-world applications, the number of tasks can range from about 14 to 300 as the input number of points or the sizes of matrix change, respectively.

Performance metrics

Makespan

Makespan, or scheduling length, is defined as

$$makespan = FT(n_{exit}), \tag{12}$$

where $FT(n_{exit})$ is the earliest finish time of exit task in the scheduling queue.

Scheduling Length Ratio (SLR)

In the presence of most scheduling algorithms, makespan, or scheduling length, is one of the major metrics of performance evaluation for a scheduling strategy. As a large set of application graphs with different properties is employed, it is necessary to normalize the makespan to a lower bound, which is named scheduling length ration. Without taking the communication cost into account, the calculation expression of SLR can be stated as

$$\mathrm{SLR} = \frac{makespan}{\sum_{n_i \in CP} \min_{p_j \in P}\{w_{i,j}\}}. \tag{13}$$

Energy Consumption Ratio (ECR)

For the sake of fair comparison, we consider energy consumption and the failure of probability for scheduled task set as one of the two most important performance measure. With regard to a given task, specially, the energy consumption ratio without considering communication cost is defined as

$$\mathrm{ECR} = \frac{E_{total}}{\sum_{n_i \in CP} \min_{p_j \in P}\{P_{ind_i} \cdot \frac{c_i}{f_i} + C_{eff} \cdot c_i \cdot f_i^2\}}, \tag{14}$$

where E_{total} represents the total energy consumption of the scheduled task.

Probability of Failure (POF)

The probability of failure is the other one of the two primary performance metrics in our comparison. Formally, the POF value of task set, with allocated frequency to each particular task by a scheduling algorithm, is given by

$$\mathrm{POF} = 1 - R = 1 - \Pi_{i=1}^{n} R_i(f_i). \tag{15}$$

Randomly generated DAG

In our study, the random DAG graphs, which are generated using the probability for an edge between any two nodes of the graph, are firstly considered. Such kind of weighted application DAGs with diverse characteristics that have a close relation on several input parameters, are presented below.

- DAG size (n): the number of task nodes in graph.
- Communication to computation ratio (CCR). In some experiment, CCR is utilized to characterize the workload of the task graph. It's the ratio with its value equals to the average of communication cost over the average of computation cost. With the help of CCR, one can judge the importance of communication or computation in the task graph. A DAG with very low value can be viewed as a computation intensive application. On the contrary, the DAG with a high CCR value is a communication heavy application.
- Average out degree:the average value of the out degree in DAG graph.
- Computation capacity heterogeneity factor (h) [37]. Heterogeneity in essence reflects the variance of processing speed. A high h refers to wider margin in the computation costs for a task, and vice versa. And each task will has the same computation cost if its factor is set to zero. The average computation cost of each task $\overline{w(n_i)}$ is selected randomly from a uniform distribution generator, whose mean value W can be specified by user. And its range can be written as $\left[\overline{w(n_i)} \times (1 - \frac{h}{2}), \overline{w(n_i)} \times (1 + h2)\right]$. The value of W has no influence to the performance result of scheduling.

In what follows, graphs are generated with the combination of above mentioned characters. The size of graph varies from 50 to 200, with step size by 25. The communication edge connect two arbitrary nodes is generated with an identical probability, which lies on the average number of edge for each node.

The random application performance analysis

The major purpose of these experiments is to evaluate the performance of the presented algorithms. For the first set of simulation studies, the results are captured in Figs. 3–5, their data points come from the average of experiment output obtained by a thousand times experiments. As can be seen from these three figures, SHREERM outperforms the other two algorithms significantly with respect to the different task size and diverse CCRs. For Fig. 3, where the application is a computation intensive with CCR = 0.2, the average of SLR of SHREERM is remarkably close to that of SHRHEFT. The reason for this is that SLR is not the metrics of most concern in this study. SHREERM improves the reliability of task execution with minimum energy consumption to the greatest extent. Again, it also can be observed from Fig. 3 that SHREERM outperforms SHRHEFT and SHRCPOP in terms of average ECR and average probability of failure (POF). We attribute the marginally better performance of SHREERM over other two algorithms to the fact that SHREERM is a reliability adaptive strategy and it assigns every task judiciously to the processor with an appropriate execution frequency according to its computational time and execution reliability. In the processor selection phase, SHRHEFT chooses the processor the earliest finish time, and SHRCPOP picks the critical processor for critical task. Both of them do not take enough account of reliability. Thus, their ECR and POF are worse than SHREERM.

The impact of large CCR value for the random graphs are shown in Fig. 5. The gaps among the three algorithms shrink gradually with the increasing of CCR value. For CCR = 1 and CCR = 5, the improvement of SHREERM evinces significantly, with respect to the quality of average ECR and average POF. Specially, as CCR equals five, it infers that communication dominate the whole application. SHREERM clearly exceeds SHRHEFT and SHRCPOP by (6.58%, 9.06%) in term of the average ECR, and (6.50%, 6.34%) in term of the average POF, respectively. SHREERM is an energy efficient and reliability conservation algorithm. The results distinctly demonstrate that SHREERM surpasses SHRHEFT and SHRCPOP. Based on the above analysis, we can draw an inclusion that SHREERM performs better as the CCR increases. In other words, it is more suitable for data intensive application.

Fig. 6 reveals the performance comparison of three algorithms under the condition of various numbers of processors. While the random graph with 200 task nodes and CCR equals to five. As before, the SHREERM strategy continues to perform better than

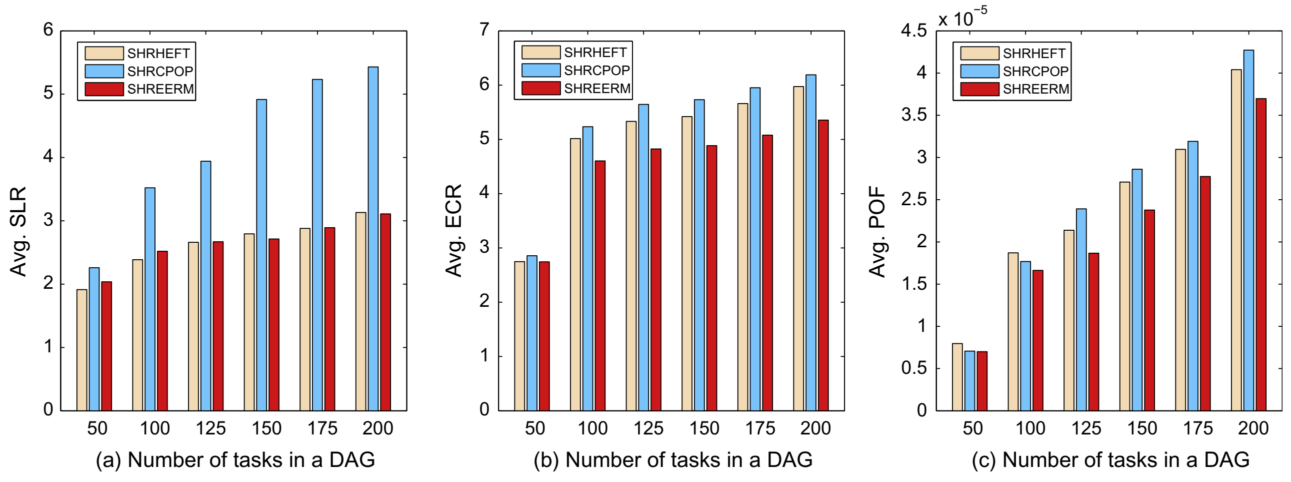

Fig. 3. Average SLR, ECR and POF of the three algorithms for CCR = 0.5.

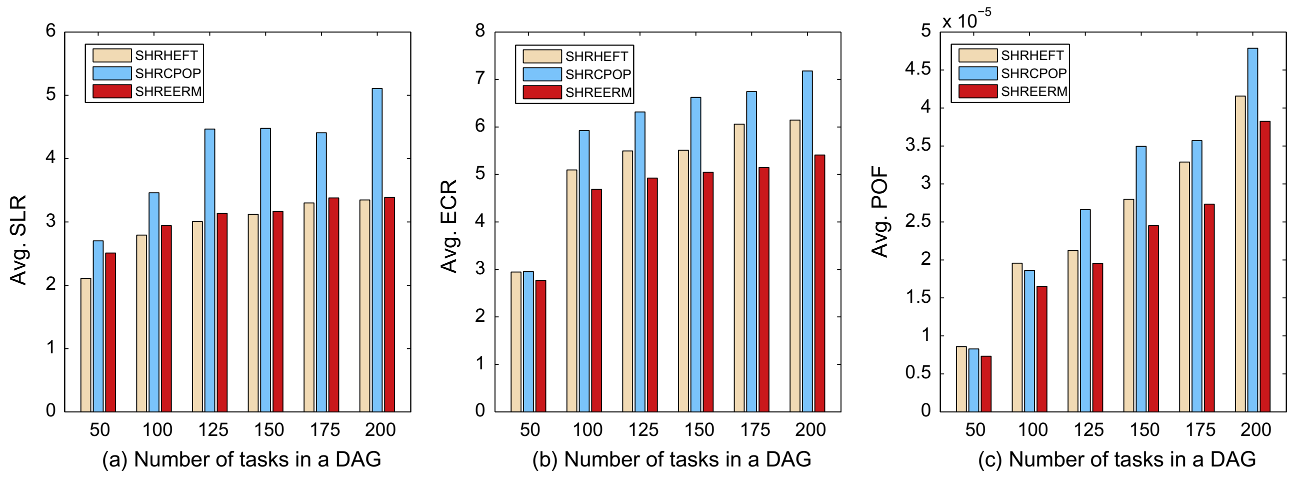

Fig. 4. Average SLR, ECR and POF of the three algorithms for CCR = 1.

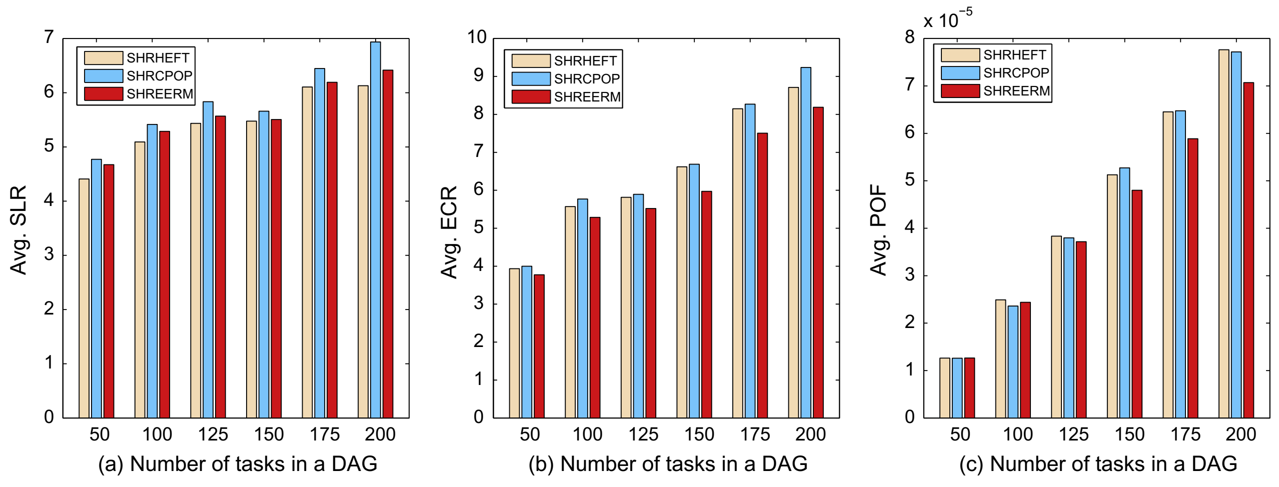

Fig. 5. Average SLR, ECR and POF of the three algorithms for CCR = 5.

other two algorithms with the increasing of the number of processors. The difference of average SLR among the three algorithms decreases gradually. Note that, the point of nine processors is a important one in the presented chart. When the number of processors is great than nine, the performance, including the average of SLR, ECR, and POF, don't benefit from the increasing number of processors. It's due to the fact that the scale of graph is not large enough. In a word, for a particular number of task nodes, the average of ECR and the average of POF show a slightly declined trend as the number of processors grows.

Performance analysis on graphs of real-world applications

Without loss of generality, it is necessary to use real applications to estimate the performance of algorithms. Therefore, three real-world applications, which are applied widely, are used to evaluate the performance of algorithms. There are real-world problems, including Fast Fourier transformation (FFT), Laplace equation solver (Laplace) and Gaussian elimination (GE).

Fast Fourier Transformation (FFT)

The fast Fourier transform is an algorithm to compute the discrete Fourier transform and its inverse transform. FFT plays a significant role in information field, as it computes such rapidly that it is used extensively for many application in science, engineering and mathematics. The computation of FFT is comprised of two sections, involving the input vector and the butterfly operation. Fig. 7 (a) shows a FFT task graph with four points. It consists of two parts, the tasks above the dashdotted line are the recursive invocation of tasks and the below ones are the butterfly operation tasks [37]. The parameters adopted in experiment are summarized in Table 6.

Let S_n be the number of nodes in FFT task graph, then we have $S_{4n} = 2^{n+3}$ $(n \geqslant 1)$, where the subscript of S_n is the size of FFT. When n changes from 1 to 6 with step by 1, S_n varies among set {16, 32, 64, 128, 256, 512}. For each kind of experiment configuration, CCR values and the number of processors are involved to combine together. Under each configuration parameter, our proposed algorithms are tested for one thousand times. The following results are assessed with the average value of outputs, including two main metrics POF and ECR.

The experiment result in terms of average ECR and POF respectively, are captured in Figs. 8 and 9. Obviously, the SHREERM is capable of yielding competitive consistently over SHRHEFT and SHRCPOP with regard to ECR and POF. Due to the comprehensive precedence constraints of most task nodes as the size of graph grows, the margin of both ECR and POF for three algorithms are not large. In such case, increasing the processor will not bring any benefit for improving performance. The overall performance of the SHREERM strategy for FFT graphs in terms of average POF is 8.19%, 13.50% better over SHRHEFT and SHRCPOP, respectively. With regard to average ECR, SHREERM performs 8.08%, 11.50% better than SHRHEFT and SHRCPOP. And on average SLR dimension, SHREERM surpasses SHRHEFT and SHRCPOP at an average of 6.14%, 21.3%.

Laplace

Laplace equation solver, is used broadly in solving mathematical equations and information technique. As shown in Fig. 7(b), it works by transforming a matrix into a production of lower and upper triangular matrixes. Table 7 gives a summation of the used parameters in the experiment of Laplace task graph. The number

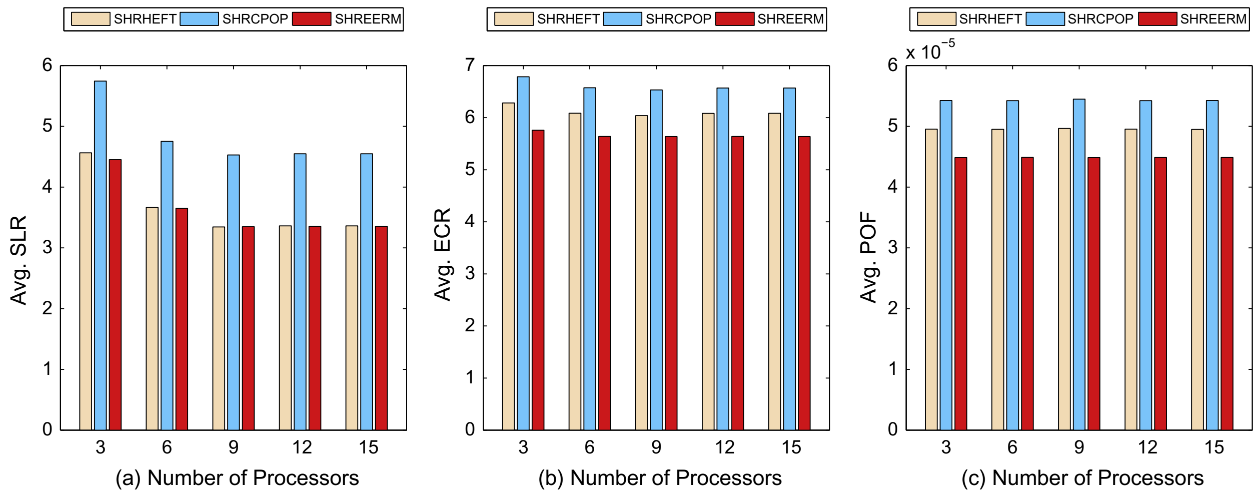

Fig. 6. Average SLR, ECR and POF of the three algorithms for CCR = 5, DAG size = 500.

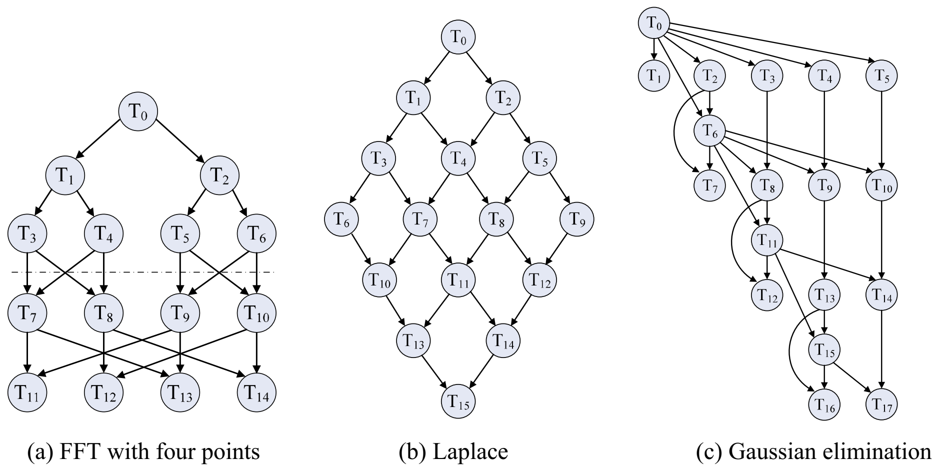

Fig. 7. Three kinds of generated DAG.

Table 6
Configuration parameter for the FFT task graphs.

Parameter	Possible values
CCR	0.2, 0.5, 1, 2, 5, 10
Number of processors	3, 6, 9, 12, 15
Size	4, 8, 12, 16, 20, 24

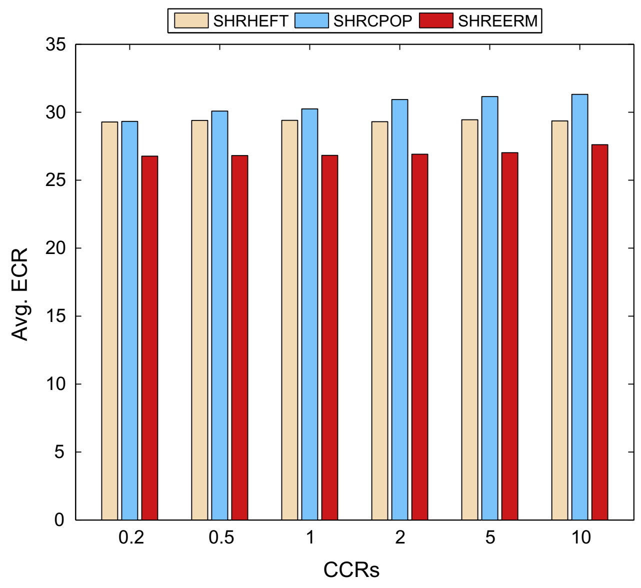

Fig. 8. Average ECR of the FFT task graph with different CCRs.

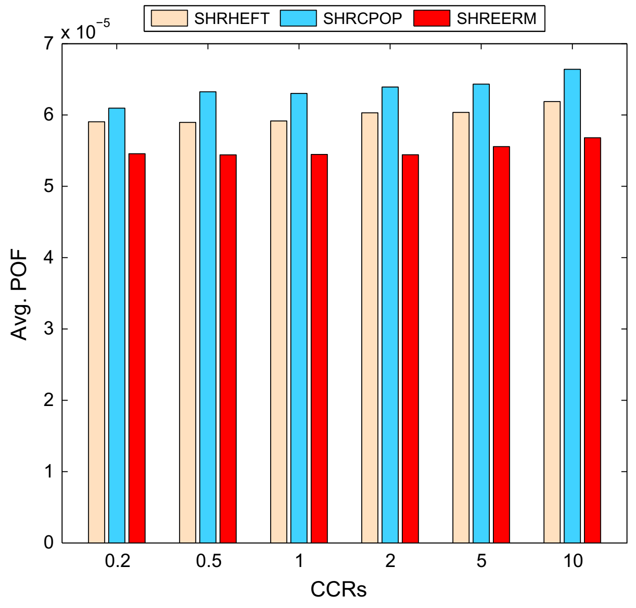

Fig. 9. Average POF of the FFT task graph with different CCRs.

Table 7
Configuration parameter for the Laplace decomposition task graphs.

Parameter	Possible values
CCR	0.2, 0.5, 1, 2, 5, 10
Number of processors	3, 6, 9, 12, 15
Size	5, 6, 7, . . ., 20, 21

of task nodes S_n satisfies expression $S_n = n^2$ $(n \geqslant 3)$, where n is the size. When the size of input matrix varies from 5 to 21 with the increment step by 1, S_n changes from 25 to 441. In addition, the number of processors varies from 3 to 15 with step by 3 and the value of CCR varies among the set {0.2, 0.5, 1, 2, 5, 10}. These three kinds of parameters in Table 7 are combined together. The presented algorithms in this study runs for one thousand times under each combination. The final experiment result is the average value of output, including the average of SLR, ECR and POF as CCR increases.

As illustrated in Figs. 10 and 11, the SHREERM strategy outperforms SHRHEFT and SHRCPOP strategies for all values of the CCR parameter. For both small and large CCR value, SHREERM algorithm consistently produces very low ECR values, especially for small CCR value. The overall performance evaluation of the SHREERM strategy for Laplace graphs is 6.21%, 10.2% in terms of average POF better over SHRHEFT and SHRCPOP respectively. With regard to average ECR, SHREERM surpasses SHRHEFT and SHRCPOP at an average of 8.65%, 10.77% respectively. And on average SLR dimension, SHREERM performs 1.35%, 16.1% better than SHRHEFT and SHRCPOP.

Gaussian-elimination

As a well known method, the Gaussian-elimination is widely used in mathematics for solving systems of equations. Let n be the size of matrix which depicts the Gaussian-elimination task graph. The number of tasks in GE graph follows the expression, $S_n = \frac{n^2+n-2}{2}$, $(n \geqslant 5)$. As the size of n varies from 5 to 22 with step by 1, S_n ranges from 19 to 252. The detailed parameter configurations are given in Table 8. The CCR, the number of processors and S_n are combined together. The developed algorithms run for one thousand time under each combination. The ultimate results are evaluated with the average value of the outputs.

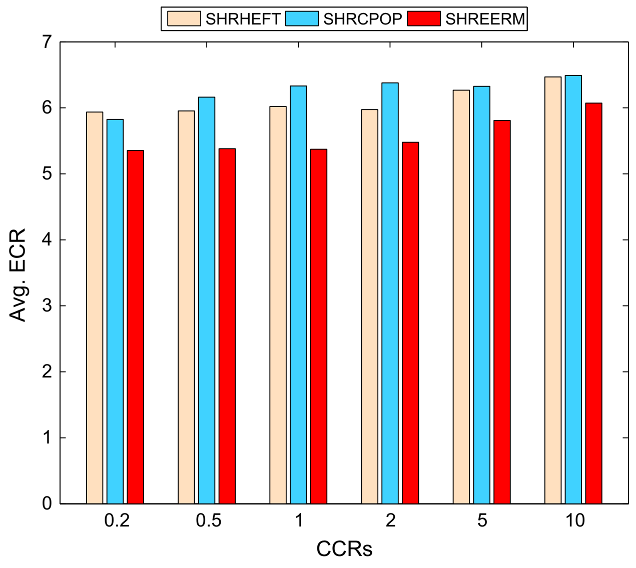

Fig. 10. Average ECR of the Laplace task graph with different CCRs.

For the presence of comparison for both of ECR and POF, the input matrix size varies from 5 to 22, with an increment step of 1. And corresponding task nodes range change from 19 to 252. It can be seen from Figs. 12 and 13 that SHREERM strategy surpasses the SHRHEFT and SHRCPOP scheduling strategies for all the CCR, in term of average ECR and average POF respectively. As CCR increases, the average ECR of three algorithms grows slightly. After a thousand times execution for GE graph in each particular size,

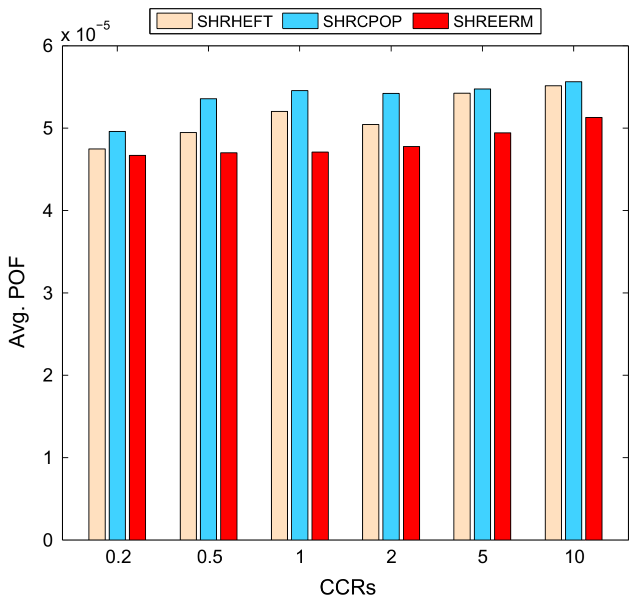

Fig. 11. Average POF of the Laplace task graph with different CCRs.

Table 8
Configuration parameter for the Gaussian elimination task graphs.

Parameter	Possible values
CCR	0.2, 0.5, 1, 2, 5, 10
Number of processors	3, 6, 9, 12, 15
Size	5, 6, 7, . . . , 20, 21, 22

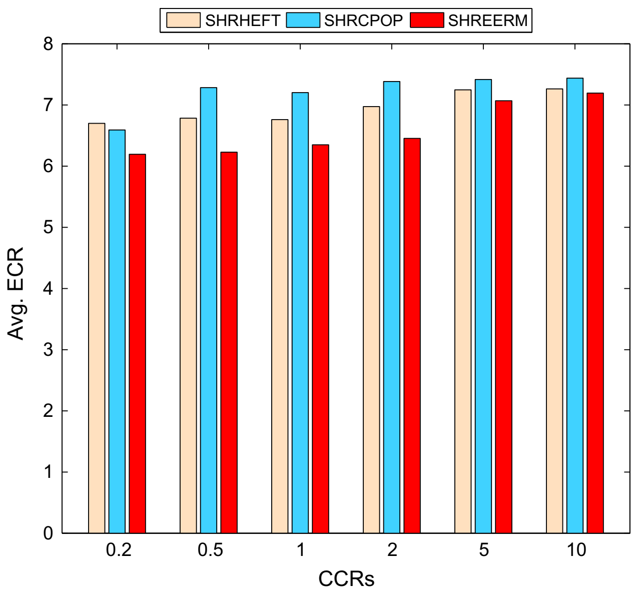

Fig. 12. Average ECR of the Gaussian-elimination task graph with different CCRs.

transient fault event simulator imitates the real environment. Once soft error arises, shared recovery block reexecute the fault task. The overall performance of the SHREERM algorithm for the Gaussian elimination graph is 6.25%, 12.4% in terms of average POF better than SHRHEFT and SHRCPOP respectively. On average ECR, RMEC performs 5.48%, 8.81% better than SHRHEFT and SHRCPOP respectively. And on average SLR dimension, SHREERM performs 1.15%, 19.76% better than SHRHEFT and SHRCPOP respectively.

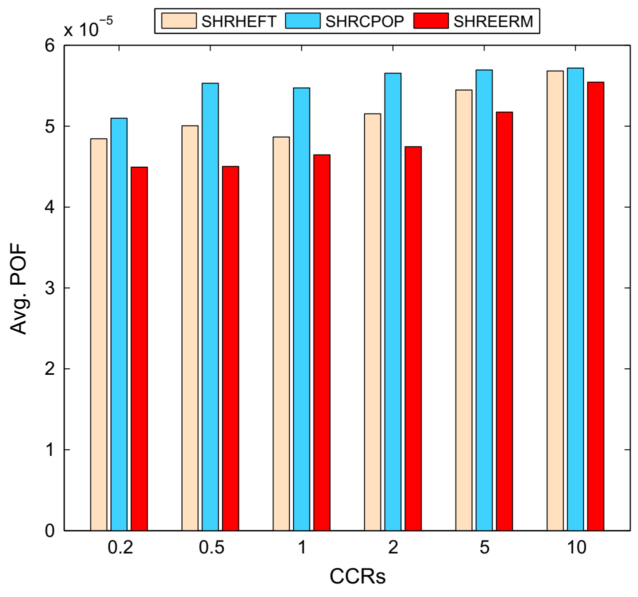

Fig. 13. Average POF of the Gaussian-elimination task graph with different CCRs.

Conclusion

As green computing is the major trend in Heterogeneous computing system (HCS), DVFS is exploited extensively to obtain energy saving. The usual approach is to reduce the execution frequency. Its side effect would increase the probability of transient fault. And task's reliability can degrade significantly without special provisions. Hence, it is essential to enhance system reliability while managing energy in HCS.

The main goal of this study is to promote system reliability while applying DVFS technique to achieve lower energy consumption for HCS. To the best of our knowledge, this problem has not been addressed in the research literature in the past. Thus, we proposed energy efficient and reliability conservative algorithms for tasks with precedence and common deadline constrains, involving SHREERM, SHRHEFT, SHRCPOP. Firstly, the effective deadline of each task depends on the precedence constrains of tasks set and follows its downward rank rule. Based upon the shared recovery technique, the scaled tasks share a single recovery block in each candidate processor. The evaluation of performance for the presented algorithms are conducted with both randomly generated graphs and the graphs of some real world applications, including fast Fourier transformation (FFT), Laplace equation solver and LU-decomposition. While scheduling a task graph with 200 nodes to 3 processors, even in the case that each processor encounters an soft error, the system reliability is greater than 0.9999. The simulation results show that these algorithms address the reliability and energy well especially when transient faults arise. And SHREERM algorithm significantly surpasses both SHRHEFT and SHRCPOP in terms of reliability conservation and energy consumption. Future studies in this area are two-fold. First, we are intent to explore the mathematical relationship between the system reliability and energy consumption. Second, we intend to develop the bi-objective optimization of reliability enhancement and energy saving.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers for their insightful comments. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos.

61370095, 61472124), the National Science Foundation for Distinguished Young Scholars of Hunan (Grant No. 12JJ1011), and the Research Foundation of Education Bureau of Hunan Province (No. 15C0400).

References

[1] Garey MR, Johnson DS. Computers and intractability: an introduction to the theory of NP-completeness, San Francisco; 1979.
[2] Venkatachalam V, Franz M. Power reduction techniques for microprocessor systems. ACM Comput Surv (CSUR) 2005;37(3):195–237.
[3] Albers S. Energy-efficient algorithms. Commun ACM 2010;53(5):86–96.
[4] Benini L, Bogliolo A, De Micheli G. A survey of design techniques for system-level dynamic power management. IEEE Trans Very Large Scale Integr (VLSI) Syst 2000;8(3):299–316.
[5] Zhong X, Xu C-Z. Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. IEEE Trans Comput 2007;56 (3):358–72.
[6] Tian Y, Boangoat J, Ekici E, Ozguner F. Real-time task mapping and scheduling for collaborative in-network processing in DVS-enabled wireless sensor networks. In: The 20th IEEE international parallel distributed processing symposium; 2006. p. 10–9.
[7] Kyong Hoon K, Buyya R, Jong K. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In: The seventh IEEE international symposium on cluster computing and the grid; 2007. p. 541–8.
[8] Ge R, Feng X, Cameron KW. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In: The 2005 ACM/IEEE conference on supercomputing; 2005. p. 34–44.
[9] Zhu D, Melhem R, Childers BR. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. IEEE Trans Parall Distrib Syst 2003;14(7):686–700.
[10] Sih GC, Lee EA. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans Parallel Distrib Syst 1993;4(2):175–87.
[11] Khan M A. Scheduling for heterogeneous systems using constrained critical paths. Parall Comput 2012;38(4):175–93.
[12] Topcuoglu H, Hariri S, Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parall Distrib Syst 2002;13(3):260–74.
[13] Topcuoglu H, Hariri S, Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parall Distrib Syst 2002;13(3):260–74.
[14] Bansal S, Kumar P, Singh K. Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. J Parall Distrib Comput 2005;65(4):479–91.
[15] Zhang L, Li K, Xu Y, Mei J, Zhang F, Li K. Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. Inf Sci 2015;319:113–31.
[16] Xu Y, Li K, He L, Zhang L, Li K. A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. IEEE Trans Parall Distrib Syst 2014;PP(99):1.
[17] Huang X, Zhang L, Li R, Wan L, Li K. Novel heuristic speculative execution strategies in heterogeneous distributed environments. Comput Electr Eng 2015. http://dx.doi.org/10.1016/j.compeleceng.2015.06.013.
[18] Kit Yan C, Dillon TS, Chang E. An intelligent particle swarm optimization for short-term traffic flow forecasting using on-road sensor systems. IEEE Trans Ind Electron 2013;60(10):4714–25.
[19] Zhao B, Aydin H, Zhu D. Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. ACM Trans Des Autom Electron Syst (TODAES) 2013;18(2):23–43.
[20] Zhang Y, Chakrabarty K. Energy-aware adaptive checkpointing in embedded real-time systems. In: Design, automation and test in Europe conference and exhibition; 2003. p. 918–23.
[21] Melhem R, Moss D, Elnozahy E. The interplay of power management and fault recovery in real-time systems. IEEE Trans Comput 2004;53(2):217–31.
[22] Tang X, Li K, Li R, Veeravalli B. Reliability-aware scheduling strategy for heterogeneous distributed computing systems. J Parall Distrib Comput 2010;70(9):941–52.
[23] Lee YC, Zomaya AY. Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans Parall Distrib Syst 2011;22(8):1374–81.
[24] Li K. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. IEEE Trans Comput 2012;61(12):1668–81.
[25] Izosimov V, Eles P, Peng Z. Value-based scheduling of distributed fault-tolerant real-time systems with soft and hard timing constraints. In: Embedded systems for real-time multimedia (ESTIMedia), 2010 8th IEEE workshop on; 2010. p. 31–40.
[26] Acharya S, Mahapatra R. A dynamic slack management technique for real-time distributed embedded systems. IEEE Trans Comput 2008;57(2):215–30.
[27] Intel. Intel pentium m processor datasheet; 2004. <http://download.intel.com/support/processors/mobile/pm/sb/25261203.pdf>.
[28] Min R, Furrer T, Chandrakasan A. Dynamic voltage scaling techniques for distributed microsensor networks. In: The very large scale integration workshop; 2000. p. 43–6.
[29] Burd TD, Brodersen RW. Energy efficient CMOS microprocessor design. In: The 28th Hawaii international conference on system sciences; 1995. p. 288–97.
[30] Zhu D, Melhem R, Moss D. The effects of energy management on reliability in real-time embedded systems. In: The IEEE/ACM international conference on computer aided design; 2004. p. 35–40.
[31] Zhao B, Aydin H, Zhu D. On maximizing reliability of real-time embedded applications under hard energy constraint. IEEE Trans Ind Inform 2010;6 (3):316–28.
[32] Ishihara T, Yamaguchi S, Ishitobi Y, Matsumura T, Kunitake Y, Oyama Y, et al. AMPLE: an adaptive multi- performance processor for low-energy embedded applications. In: The international symposium on application specific processors; 2008. p. 83–8.
[33] Kartik S, Murthy C S R. Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Trans Comput 1997;46(6):719–24.
[34] Sridharan R, Gupta N, Mahapatra R. Feedback-controlled reliability-aware power management for real-time embedded systems. In: The 45th ACM/IEEE design automation conference; 2008. p. 185–90.
[35] Izosimov V, Pop P, Eles P, Peng Z. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In: The conference on design, automation and test in Europe, vol. 2; 2005. p. 864–9.
[36] Zhao B, Aydin H, Zhu D. Enhanced reliability-aware power management through shared recovery technique. In: The 2009 international conference on computer-aided design; 2009. p. 63–70.
[37] Daoud M I, Kharma N. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J Parall Distrib Comput 2008;68(4):399–409.