Received August 29, 2018, accepted September 28, 2018, date of publication October 4, 2018, date of current version October 29, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2873641

Energy-Efficient Fault-Tolerant Mapping and Scheduling on Heterogeneous Multiprocessor Real-Time Systems

KAI HUANG[1], XIAOWEN JIANG[1], XIAOMENG ZHANG[1], RONGJIE YAN[2], KE WANG[1], DONGLIANG XIONG[1], AND XIAOLANG YAN[1]

[1]Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China

[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

Corresponding author: Xiaowen Jiang (xiaowen_jiang@zju.edu.cn)

This work was supported by the National Key R&D Program of China under Grant 2018YFB0904900 and Grant 2018YFB0904902.

ABSTRACT Energy saving and system reliability are two crucial issues for designing modern multiprocessor systems. There has been reliability-aware power management with dynamic voltage–frequency scaling (DVFS) schemes in recent studies. However, they are limited to optimization under the impact of DVFS on energy and reliability and have not considered reducing the non-negligible leakage energy consumption. In this paper, we focus on co-management of system reliability and total energy for applications with precedence constrained tasks on heterogeneous multiprocessor real-time systems. We first investigate the impact of energy management techniques on both reliability and energy of the systems using task recovery for fault tolerance and then propose an Energy-efficient Fault-tolerant Scheduling (EFS) scheme integrated with power mode management, which can mitigate the negative impact of DVFS on system reliability. To obtain the optimal energy-efficient reliability-guaranteed scheduling for pre-mapped applications on systems considering various realistic issues, we build mixed integer linear programing formulations with the proposed EFS scheme. To address mapping and scheduling for energy-efficiency and fault-tolerance, we finally develop a framework implemented by a List-based Binary Particle Swarm Optimization algorithm. The extensive comparative evaluations for synthetic and realistic benchmarks show that our approaches outperform several related studies in terms of energy consumption and system reliability.

INDEX TERMS Energy, map and scheduling, multiprocessor real-time systems, reliability.

I. INTRODUCTION

During the past decade, the complexity of applications increased dramatically. In modern computational systems, heterogeneous multiprocessor systems can provide high performance potential which have attracted researchers to design these systems. When applications such as real-time image recognition, avionics, automotive and medical control, are executed on such systems, energy management has become one of the hottest research areas. Several techniques have been developed to manage power consumption in task scheduling. Among them, Dynamic Voltage Frequency Scaling (DVFS) and Power Mode Management (PMM) are two commonly used techniques. DVFS reduces power consumption by scaling down supply voltage/frequency of processors, whereas PMM saves energy by switching idle processors to low-power inactive state.

Reliability is another major issue for designing the systems. Faults occur unpredictably during runtime due to various reasons, such as hardware failures, electromagnetic interferences and the effects of cosmic ray radiations [1]. If such faults are not dealt with in time, it may lead to a disaster. Transient faults are known to occur much more frequently in practice [2] and shown to be dominant especially with scaled manufacturing technology process [3]. Since the transient faults are non-persistent, task recovery has been employed to provide system fault-tolerance and enhance reliability. For instance, the shared-recovery reserves system slacks (or idle time interval when processor is not running) and lets recovery tasks share a single recovery block to recover the faulty task [4].

The joint optimization of reliability and energy has caught researchers' attention recently. Related researches have focused on reliability-aware power management with DVFS algorithms [1], [5]–[9]. DVFS has a negative impact on system reliability, i.e., the transient fault of a system increases when a processor runs at a reduced voltage/frequency to save power consumption [5], [10]. In order to guarantee system reliability as high as that without voltage/frequency scaling, only a few reliability-aware DVFS schemes which consider task recovery for fault-tolerance have been studied [6], [11], [12]. However, all these works are limited to optimization under the impact of DVFS on energy savings and system reliability. Therefore, it is necessary to design a scheduling scheme which can mitigate the negative impact of DVFS on system reliability to achieve low total energy consumption and high reliability simultaneously.

Furthermore, total energy consumption cannot be efficiently optimized with reliability-aware DVFS schemes, as the reduction in dynamic energy consumption does not represent the reduction in total energy consumption. Static energy, also called leakage energy, is consumed whenever the processor is on. Leakage energy is comparable to the dynamic energy in modern processors. According to [13], the leakage energy occupies more than 50% of total energy. Therefore, leakage energy consumption cannot be ignored anymore and needs to be considered together with dynamic energy consumption. Although leakage energy has been considered in [4], [5], [11], [12], processors are always active and not put into low-power state during scheduling, and no appropriate measures have been taken to optimize it. Moreover, careful consideration must be taken when energy optimization and task recovery techniques are used in scheduling with deadline constraint. Since the techniques compete for slacks, a trade-off is necessary between system energy savings and reliability requirement. In addition, from a practical point of view, various issues such as transition overheads caused by processor mode switching, voltage/frequency scaling and inter-processor communications also need consideration.

In this paper, we study the problem of mapping and scheduling real-time applications on heterogeneous multiprocessor systems for energy and reliability co-design. We aim to achieve low energy consumption and provide high-level reliability for the systems. It is generally appreciated that mapping and scheduling problems on multiprocessors are interdependent and NP-hard problems [14]. Integer linear programming has the advantages of reachable optimality, easy modeling, and easy access to various solving tools. The problems have been traditionally modeled as the integer linear programming problems, and solved by solvers. Although the integer linear programming is a convenient formal modeling method, there is consensus on the fact that the method is only applicable to small problem instances, because of its high computational cost. Moreover, when all processors are the same in the systems, the problem will reduce to the case for homogeneous multiprocessor systems, which has been shown to be NP-hard [9]. Therefore, finding the optimal solution for the problem is intractable. The entire problem is often addressed by decomposing it into two sub-problems, i.e., mapping and scheduling [6], [15]–[18]. For easy understanding, in this paper, we first address energy-efficient scheduling in the case of given mapping with system reliability requirement. Then, we present and implement a framework of fault-tolerant static mapping and scheduling for energy-efficiency. Specifically, the paper makes the following contributions:

- Based on the analysis of relationship between energy and reliability, we propose an Energy-efficient Fault-tolerant Scheduling (EFS) scheme integrated with PMM to alleviate the negative impact of decreasing voltage/frequency on system reliability.
- To address the energy-efficient scheduling of pre-mapped applications on heterogeneous multiprocessor systems with reliability requirements, we build Mixed Integer Linear Programing (MILP) formulations with EFS scheme to obtain the optimal solution. In our formulation, the effects of various practical issues are considered.
- To address the mapping and scheduling problem for energy savings and fault-tolerance, we develop an Energy-efficient Fault-tolerant Mapping and Scheduling (EFMS) framework. In the framework, a List-based Binary Particle Swarm Optimization (LBPSO) algorithm is proposed to find the best solution through globally iterative and mutual optimization.
- We conduct simulations on a set of benchmarks in different types from both synthetic task graphs and real-life applications. Extensive experiments demonstrate the efficiency of our approach.

To the best of our knowledge, this is the first research effort that tries to solve mapping and scheduling of real-time applications on heterogeneous multiprocessor systems by integrating PMM with reliability-aware DVFS scheme for energy-efficiency and reliability conservation. We believe the investigation of such problem will prove useful in practice.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents the models used in this paper and gives the formal definition of the problem. Section IV presents our proposed EFS scheme and MILP formulation, and Section V describes the detailed proposed EFMS framework and its implementation. Then Section VI discusses the experimental evaluations, and Section VII gives the conclusion and future work.

II. RELATED WORK

Energy-efficient scheduling on multiprocessors has received tremendous research endeavors. DVFS is one of the most commonly used energy management techniques, and earlier works focused on reducing dynamic power consumption by using the DVFS and neglected leakage power consumption completely. Since transistor density continues to increase with the advent of deep sub-micron and nano-scale technologies, modern processors have non-negligible leakage power dissipation in recent years. The energy efficient scheduling and leakage energy management were studied in [19]–[21]. Further, combined DVFS and PMM for energy optimization is regarded as the preferable method. To name a few, for uni-processor, Devadas and Aydins [22] combined DVFS and PMM to minimize energy consumption for independent frame-based tasks, where DVFS is used for the uni-processor and PMM is applied to peripheral devices. Based on [22], Gerards and Kuper [23] proposed a schedule that globally minimizes total energy consumption. However, their approaches are not suitable for multiprocessor systems. For multiprocessors, Awan and Zarandi [24] proposed a two-phase approach with mixed DVFS and PMM to address the energy optimization problem of task-to-core allocation onto heterogeneous multicore platforms. They first allocated tasks to cores to reduce dynamic energy consumption and then refined the allocation to achieve better sleep states by trading off the dynamic energy consumption with the reduction in leakage energy consumption. Chen *et al.* [15] introduced a key technique to model the idle interval of cores and applied DVFS+PMM policy to find an optimal time-triggered scheduling for energy minimization. However, the existing approaches for energy management do not consider reliability.

Recent works have considered both energy and reliability simultaneously. DVFS is shown to have a negative impact on system reliability due to increased number of transient faults at lower frequencies [5]. Considering the reliability loss due to the energy management, Zhu [1] and Zhu and Aydin [7] proposed a reliability aware power management (RAPM) scheme to reserve a portion of the available slack to schedule a recovery task for any task whose execution is scaled down through DVFS. Fan *et al.* [25] presented an online power management approach to achieve energy savings for single processor real-time scheduling under reliability constraint. Compared with [1] and [7], Zhao *et al.* [4] proposed the shared recovery-based RAPM scheme to avoid the offline allocation of individual recovery tasks to the scaled tasks by assigning a shared recovery block that can be used by any tasks at run-time. The scheme can reserve less slack for recovery and more slack for energy management techniques to improve energy savings. However, the presented scheme cannot handle tasks with precedence constraints. The scheme was further extended to tasks with precedence-constraints [8]. Nevertheless, the aforementioned approaches in [1], [4], [7], [8], and [25] are for uni-processor.

However, only a few researches have been devoted to mapping and scheduling of dependent tasks on heterogeneous multiprocessor systems with reliability requirement. Xie *et al.* [12] implemented energy-efficient scheduling algorithms for reliable parallel applications on heterogeneous distributed embedded systems. However, they do not consider timing constraints for real-time embedded applications. Pop *et al.* [6] addressed task scheduling and voltage scaling for hard real-time applications statically mapped on heterogeneous distributed embedded systems. They proposed a constrained logic programming-based RAPM algorithm to determine the voltage levels and start time of tasks to tolerate transient faults and minimize energy consumption, while meeting timing constraints of the applications. Similar to [6], Guo *et al.* [9] explored the RAPM problem of mapping and scheduling real-time applications with precedence constraints and presented both offline and online task recovery based heuristic schemes. Based on [4], Zhang *et al.* [11] proposed joint optimization between the energy consumption and reliability for mapping and scheduling an application with a deadline constraint for heterogeneous computing systems. By applying DVFS and cooling slacks, Abdi and Zarandi [26] introduced a hybrid scheduling approach to jointly optimize performance, lifetime reliability, energy consumption and temperature of heterogeneous multiprocessors statically. However, all the above works focus on reliability-guaranteed algorithms with DVFS technique only, and reduce dynamic energy consumption under the negative impact of DVFS on system reliability. Moreover, a reduced dynamic energy consumption does not always reduce total energy consumption, as they assumed that processors are not put into sleep during scheduling for the sake of simplicity. Thus they cannot effectively reduce total energy consumption. Reference [27] is the first study that tried to derive better energy performance with combination of DVFS and PMM techniques with the reliability constraints. Nevertheless, this work only focuses on independent tasks and uni-processor, and cannot provide optimal solutions.

In this paper, we explore the problem of real-time applications with precedence constraints running on heterogeneous multiprocessor systems for energy-efficiency and fault-tolerance. From the above works, we know that existing studies, either do not consider the reliability requirements or address the problem only under DVFS settings, or the models are different from what we are addressing. Therefore the approaches proposed by previous works cannot be directly applied to our problem.

III. MODELS AND PROBLEM FORMULATION

In this section, we first introduce the models of system architecture, application, energy and reliability used in this study, and then formulate the problem based on the models.

A. ARCHITECTURE MODELING

We consider a typical distributed architecture with a set of M heterogeneous processors ($U = \{u_0, u_1, \ldots, u_{M-1}\}$), as shown in Fig. 1. The architecture is gaining popularity and a number of studies have been presented to date [11], [12], [15], [24]. In the architecture, each processor has its own local memory and communication link interface to every other processor, i.e., it has a fully connected network. The communication procedure among inter-processors relies solely on message-passing [28]. If the data that a task needs to read is not available in the local memory, inter-processor communication happens. The real communication cost occurs only in inter-processor communications where dependent tasks mapped on different processors. Note that when the tasks are allocated to the same processor, the communication cost becomes zero as the intra-processor communication can

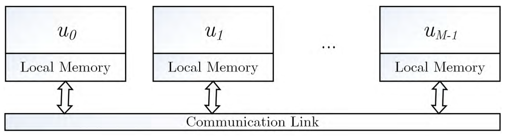


FIGURE 1. Architecture model.

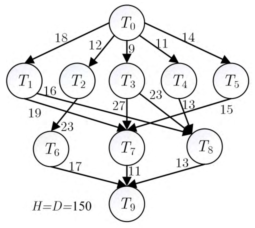
be ignored. In addition, any two tasks mapped to the same processor must not overlap in time. Communications are supposed to perform at the same speed on all links without contentions. Each processor has independent I/O unit that allows for communication and computation to be performed simultaneously. Besides, the processors in the architecture are available for DVFS and PMM, namely, each processor can run at different speeds (i.e., different supply voltage/ frequency levels) and can enter into the sleep mode.

B. APPLICATION MODELING

An application can be typically decomposed into small tasks with precedence constraints and represented as a directed acyclic graph (DAG). Nodes in the DAG are associated with the decomposed tasks and edges are associated with precedencies between the connected tasks. In this paper, the terms of application, DAG and task graph are interchangeably used. The input application is modeled as a task graph $G = (T, E, H, D)$, where T is a task set which comprises N tasks, and E is a set composed of edges representing tasks precedence constraints. The period of G is H, and we assume the deadline D of the application is equal to H. A task $T_i \in T$ $(0 \leq i \leq N-1)$ must be executed sequentially without preemption in the same processor. Each edge $e_{i,j} \in E$ denotes the inter-task communication between connected task T_i and T_j, which means that task T_i has to transmit the result to task T_j before T_j starts its execution.

We use $w_{i,m}$ to represent the Worst Case Execution Time (WCET) of task T_i on processor $u_m \in U$ under maximum available frequency level f_{max}. Assuming that processor u_m can only operate at L different discrete frequency levels $f_1, f_2, \ldots, f_L$, where $f_1 = f_{max}$ and $f_L = f_{min}$ are the maximum frequency and minimum frequency, respectively. In the DAG, each edge $e_{i,j}$ is associated with a weight $c_{i,j}$ representing the communication time between task T_i and T_j. We assume all information of the application including WCETs profile, communication costs and task dependencies are known a priori.

A simple example of an application modeled by a task graph G is illustrated on the left side of Fig. 2. The task graph has 10 dependent tasks. T_0 and T_9 are entry task and exit task, respectively. The period and deadline of G are equal to 150. The edges represent communications between corresponding connected tasks. The weight 18 of the edge between T_0 and T_1 denotes the real communication time, if task T_0 and T_1 are not allocated to the same processor. The task WCETs of the application running on a tri-processor architecture represented by $U (U = u_0, u_1, u_2)$ are shown



T_i	u_0	u_1	u_2
T_0	14	16	9
T_1	13	19	18
T_2	11	13	19
T_3	13	8	17
T_4	12	13	10
T_5	13	16	9
T_6	7	15	11
T_7	5	11	14
T_8	18	12	20
T_9	21	7	16

FIGURE 2. A DAG-based application and task WCETs [12], [29].

on the right side of Fig. 2. Due to the heterogeneity of the processors, the same task has different WCETs on different processors. For example, the value 14, 16 and 9 of task T_0 on processor u_0, u_1 and u_2 represent the WCETs of the task denoted by $w_{0,0} = 14$, $w_{0,1} = 16$ and $w_{0,2} = 9$, respectively.

C. POWER AND ENERGY MODELING

We adopt a classic power model [5], [30], [31], whose accuracy has been verified by SPICE simulation. DVFS and PMM techniques will be used to exploit voltage/frequency scaling and low-power states to maximize energy savings. We adopt inter-task DVFS in this paper [5], [6], [11], [12], [15], i.e., the frequency of a processor stays constant for entire duration of a task's execution. The supply voltage is scaled down in approximates linear relationship with the processing frequency. To avoid confusion, we use the term frequency change to represent changing the supply voltage and processing frequency simultaneously. The dynamic power P_d includes a frequency-dependent power component P_{dep} (determined by frequency levels) and a frequency-independent power component P_{ind} (driven by modules such as memory and I/O subsystem in the active state) [5], i.e., $P_d = P_{dep} + P_{ind}$. P_{dep} and operation frequency f are given by:

$$P_{dep} = C_{ef} \times V_{dd} \times f \tag{1}$$

$$f = k \times [(V_{dd} - V_t)]^2 / V_{dd} \tag{2}$$

where C_{ef} is the effective switching capacitance, V_{dd} is the supply voltage, k is a circuit dependent constant and V_t is the threshold voltage. The static power P_s is given by:

$$P_s = V_{dd} \times I_{leak} \tag{3}$$

where I_{leak} denotes leakage current. Let P_{on} be the intrinsic power that is needed to keep the processor on. Thus, the power of a processor when it is active can be approximated by:

$$P_{active} = P_d + P_s + P_{on}. \tag{4}$$

Each processor has three modes, i.e., active, idle and sleep state. When a processor does not execute any task (idle state), its power consumption is primarily determined by the idle power. We assume that P_{idle} and P_{sleep} respectively represent the idle power and sleep power. Normally, we have $P_{idle} > P_{sleep}$. Considering the overhead of switching the processor between active mode and sleep mode, the definition of processor break-even time is given as follows.

Definition 1: Break-even-time. The minimum time interval for a processor entering sleep state is more effective (energy-wise) when compared to the idle state, despite an extra time and energy overhead associated to mode switch between active and sleep state. The processor should keep in idle with P_{idle} if $t_{idle} < T_{bet}$, otherwise, the processor enters sleep with P_{sleep}. The break-even-time T_{bet} is defined as:

$$T_{bet} = \max\{t_{ms}, (E_{ms} - P_{sleep} \times t_{ms})/(P_{idle} - P_{sleep})\} \quad (5)$$

where t_{ms} and E_{ms} are time and energy overhead of the processor mode switching, respectively.

The energy consumption of executing task T_i on processor u_m under frequency level f_l can be represented as:

$$E_{active}(T_i, f_l) = w^l_{i,m} \times P^l_{active} \quad (6)$$

where $w^l_{i,m}$ is the execution time of T_i on u_m under f_l and P^l_{active} is active power of the processor under f_l. The energy consumed in idle mode (E_{idle}) and sleep mode (E_{sleep}) are calculated as:

$$E_{idle} = P_{idle} \times t_{idle} \quad (7)$$

$$E_{sleep} = P_{sleep} \times t_{sleep} \quad (8)$$

Therefore, given a static schedule S, the total energy consumption of the system is calculated as:

$$E_{total}(S) = E_{active}(S) + E_{idle}(S) + E_{sleep}(S) + E_{cov}(S) \quad (9)$$

where E_{cov}(S) is the energy consumption due to the overhead of processor mode switches.

Definition 2: Critical frequency. The critical frequency f^{cri}_m is the processing frequency of each processor u_m that minimizes the total energy consumption [7], [30]. Not all the tasks on u_m can run with f^{cri}_m, as some tasks may miss deadline if all tasks are executed at f^{cri}_m. We assume that the frequency of the processor varies in $[f_{min}, f_{max}]$. Then all actual effective frequencies of the tasks on the processor should belong to the scope of $[f_{low}, f_{max}]$ where $f_{low} = \max\{f_{min}, f^{cri}_m\}$.

D. RELIABILITY AND FAULT TOLERANCE MODELING

In this paper, we concentrate on transient faults since they occur much more frequently than permanent faults. The failure rate at a scaled frequency f can be modeled as [5]: $\lambda(f) = \lambda_0 \times 10^{d(f_{max}-f)/(f_{max}-f_{low})}$, where λ_0 is the average fault rate corresponding to the maximum frequency f_{max}, and $d(> 0)$ is a constant, which represents the sensitivity of failure rates caused by transient faults. From the formula, one can easily derive that $\lambda(f)$ is a strictly decreasing function. And reducing the processing frequency will result in exponentially increased fault rates. When f is equal to f_{low}, the maximum average transient fault rate can be expressed as $\lambda_{max} = \lambda_0 \times 10^d$.

The reliability of a task is defined as probability of executing the task successfully in the absence of transient faults. It is subject to Poisson distribution with an average fault rate $\lambda(f)$ [10]. Thus, the reliability of a task T_i on processor $u_m \in U$ with WCET $w_{i,m}$ at frequency f_l is expressed as:

$$R_i = e^{-\lambda(f_l) \times w_{i,m} \times f_{max}/f_l} \quad (10)$$

The overall reliability of a system depends on the correct execution of all tasks [1], [6], [8], [11]. For N tasks in the application model, the reliability of the system is defined as:

$$R(G) = \prod_{i=0}^{N-1} R_i. \quad (11)$$

Definition 3: Original reliability. The original reliability of a task T_i on u_m is defined as the reliability level obtained when executed at f_{max} and is given as $R^o_i = e^{-\lambda(f_{max}) \times w_{i,m}}$ [5]. As the correctness of application G generally depends on the successful execution of all its tasks, the system's original reliability is defined as follows [1]:

$$R^o_{sys} = \prod_{i=0}^{N-1} R^o_i. \quad (12)$$

Since the transient fault may occur at run-time, system fault tolerance solutions are required. The shared-recovery technique provides a protection for single-fault scenarios that are typically much more likely than multiple-fault scenarios. As in [1], [4], [7], and [8], we assume that the recovery takes the form of re-execution of the faulty task at f_{max} once a transient fault occurs. Thus, the overall reliability of a task T_i executed at a scaled frequency $f_l (< f_{max})$ is given as:

$$R'_i = R_i + (1 - R_i) \times R^o_i \quad (13)$$

In (13), R_i is the probability that the primary scaled task completes correctly, and $(1-R_i) \times R^o_i$ indicates the probability of the recovery executing at f_{max} correctly when the primary task fails. Obviously, the task T_i's original reliability R^o_i is preserved ($R'_i \geq R^o_i$). It has been proved that the shared-recovery can preserve the system's original reliability in every period [4]. Soft errors caused by transient fault can be detected by sanity checks at the end of a task's execution [32]. The overhead of fault detection is assumed to be included into tasks' WCETs.

Considering that the probability that a task fails during execution is rather small, that is, the recovery task is rarely invoked, we focus on the primary tasks (i.e., without considering the energy consumption of recovery tasks) and aim to minimize the fault-free energy consumption.

E. PROBLEM STATEMENT

The problem addressed in this paper is defined as follows: *Given a DVFS- and PMM-enabled heterogeneous multiprocessor system, a periodic application, and task WCETs profile information, the energy-efficient fault-tolerant mapping and scheduling problem is to determine 1) where each task in the application should be allocated to, 2) which frequency level and when each task should be executed, so that the fault-free total energy consumption E_{total} is minimized and*

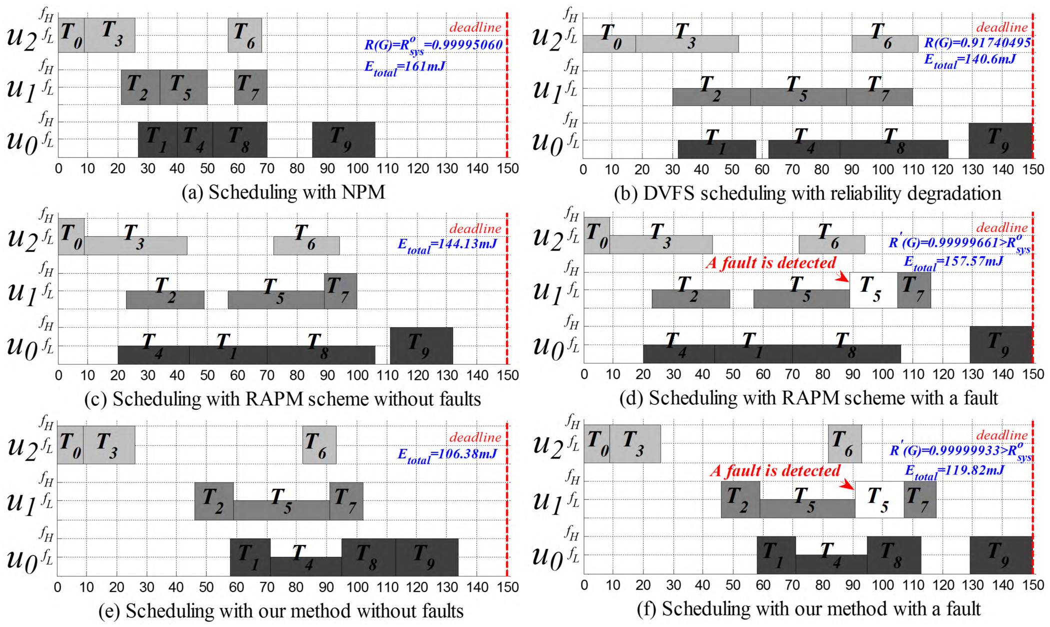


FIGURE 3. Schedules of application *G* in Fig. 2 under different schemes. (a) Scheduling with NPM. (b) DVFS scheduling with reliability degradation. (c) Scheduling with RAPM scheme without faults. (d) Scheduling with RAPM scheme with faults. (e) Scheduling with our method without faults. (f) Scheduling with our method with faults.

TABLE 1. Power parameters of processors.

u_i	P^H_{dep}	P^L_{dep}	P_{ind}	P^H_s	P^L_s	P_{on}	P_{idle}	P_{sleep}	E_{ms}	T_{bet}
u_0	0.4	0.1	0.04	0.12	0.08	0.15	0.19	0.0008	1	25
u_1	0.5	0.13	0.03	0.13	0.09	0.18	0.21	0.0005	0.8	20
u_2	0.3	0.07	0.05	0.16	0.08	0.12	0.17	0.0004	0.7	18

system's original reliability is preserved while task's deadline and precedence constraints are guaranteed.

IV. ENERGY-EFFICIENT FAULT-TOLERANT SCHEDULING

In this section, we first give a motivating example to show that state-of-art RAPM-based schemes may not work well on the problem. Then, we analyze the relationship between energy and reliability, and propose our EFS scheme. Finally, we build the MILP formulation with EFS scheme to obtain the optimal reliability-guaranteed scheduling under given mapping for energy-minimization.

A. MOTIVATING EXAMPLE

As the case shown in Fig. 2, the application is executed on a tri-processor architecture. We assume that each processor has two discrete frequency levels, i.e., high level f_H and low level f_L. f_H and f_L are normalized as 1 and 0.5, respectively. The clock cycle at f_H and f_L are 1 and 2 time units, respectively. For simplicity, time unit is $1ms$ and the energy unit is $1mJ$. P_{dep} and P_s under f_H (f_L) are denoted as P^H_{dep} (P^L_{dep}) and P^H_s (P^L_s), respectively. The power parameters of corresponding processors are based on [15] which are shown in Table 1.

Given the mapping $\Theta_{u_0} = \{T_1, T_4, T_8, T_9\}$, $\Theta_{u_1} = \{T_2, T_5, T_7\}$ and $\Theta_{u_2} = \{T_0, T_3, T_6\}$, a feasible schedule is shown in Fig. 3(a). In the schedule, all tasks are executed with the maximum frequency, i.e., with no power management. Without loss of generality, assume that $d = 3$, and λ_0 for u_0, u_1 and u_2 are 3e-7, 2e-7 and 6e-7, respectively. Then the minimum and maximum reliability values can be calculated as 90.592387% and 99.995060%, respectively, and we assume that the reliability requirement is 95%. From Fig. 3(a), the system's original reliability is $R^o_{sys} = 99.995060\%$ and the total energy consumption is calculated as $161mJ$. We can see that the system still has access to 150-106=44 (ms) of slack, and the execution of tasks can be scaled down to further reduce energy consumption. Fig. 3(b) shows a scheduling with DVFS which fully utilizes the slack to gain energy savings [18]. The total energy consumption is calculated as $140.6mJ$. However, the system reliability $R(G)$ is calculated as 91.740495% which is lower than the reliability requirement. The schedule does not consider reliability degradation and cannot provide fault tolerance.

Considering the system reliability requirement, a schedule under RAPM-based schemes in case that no fault occurs is illustrated in Fig. 3(c). The fault-free total energy consumption in the schedule is $144.13mJ$, which is larger than the schedule in Fig. 3(b). Fig. 3(d) illustrates the recovery block is used to re-execute the faulty primary task T_5 and the recovery task T_5 is executed at f_{max}. The system reliability $R'(G)$ is further calculated as 99.999661%. We can see that system reliability under RAPM is enhanced compared with Fig. 3(a) and Fig. 3(b), while the energy consumption increases compared with Fig. 3(b).

To further reduce total energy consumption and enhance system reliability, we consider integrating PMM with reliability-aware DVFS scheduling in our method. In case that no fault occurs as illustrated in Fig. 3(e), the fault-free total energy consumption in the schedule is $106.38mJ$, which is an improvement of 26.19% over that of RAPM scheme. Fig. 3(f) shows the task T_5 is re-executed at f_{max} when the primary task T_5 fails. The system reliability $R'(G)$ is further calculated as 99.999933%, which exceeds that of RAPM scheme. This indicates that our method which integrates PMM with reliability-aware DVFS scheduling is more effective than the RAPM scheme.

B. ENERGY-EFFICIENT FAULT-TOLERANT SCHEDULING SCHEME

The reliability value of an application is the product of reliability value of each task as shown in (11). Assuming that R_i^{min} and R_i^{max} are the minimum and maximum reliability value of task T_i, respectively. They can be obtained by traversing through all processors when the task is executed at the minimum and maximum frequency supported by corresponding processor. Thus, the lower bound and upper bound of reliability value of an application G are represented as $R_{min}(G) = \prod_{i=0}^{N-1} R_i^{min}$ and $R_{max}(G) = \prod_{i=0}^{N-1} R_i^{max}$, respectively. We denote the reliability requirement of the application as $R_c(G)$ and assume that $R_c(G)$ belongs to the scope of $[R_{min}(G), R_{max}(G)]$.

To minimize energy consumption with reliability requirement, the constraint of $R_c(G)$ can be transferred to the constraint of reliability value for each task in G [12], [27]. Specifically, if the reliability value of each task exceeds $\sqrt[N]{R_c(G)}$, then the reliability value of the application must exceed its reliability requirement. For each task, the reliability value of the task is determined by its processing frequency if the task mapping is given. Thus, there must be a minimum reliable frequency that guarantees the reliability value of corresponding task. As long as processing frequency of each task is larger than or equal to its minimum reliable frequency, the reliability requirement of application G can always be satisfied. Assuming that task T_i is mapped on processor u_m, its' reliable frequency $f_{i,m}^{rel}$ can be obtained by using the relation between the reliability and frequency in (10). As f_m^{cri} is the critical frequency that consumes least energy, we set $f_{i,m}^{rel}$ to f_m^{cri} when $f_{i,m}^{rel} < f_m^{cri}$.

The fundamental idea of the existing RAPM scheme is that DVFS is adopted to scale down the execution of tasks for reducing energy consumption, and task recovery is used to utilize the available slack for maintaining system's original reliability [1], [4], [7]–[9], [11]. Assuming that a task T_i is dispatched at time t with WCET of W and needs to finish by $t + D$. As shown in Fig. 4(a), the RAPM scheme reserves a slack to schedule a recovery task T_r for the primary task T_i. The recovery task T_r will be dispatched at the maximum frequency f_{max} only if the fault is detected. In the scheduling, task T_i is executed at f_a with execution time w_a and active power P_{active}^a, and in $(t+w_a, t+D)$, the processor is idle after

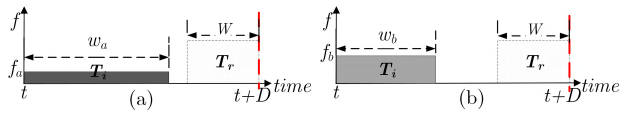


FIGURE 4. (a)RAPM scheme and (b)integrating PMM into the reliability-aware DVFS scheduling.

completing the task. The total fault-free energy consumption under RAPM is $E_a = P_{active}^a \times w_a + P_{idle} \times (D - w_a)$.

Considering that existing reliability-aware scheduling focuses on optimizing system energy consumption with DVFS only, the total energy consumption can be further reduced by jointly adopting DVFS and PMM techniques. As shown in Fig. 4(b), in the scheduling, task T_i is executed at $f_b (f_b > f_a)$ with execution time $w_b (W < w_b < w_a)$ and active power P_{active}^b, and in $(t + w_b, t + D)$, the processor enters sleep state. The total fault-free energy consumption in such scheduling is $E_b = P_{active}^b \times w_b + P_{sleep} \times (D - w_b) + E_{ms}$.

The reliability value of a task is positively related to its provided frequency from (10). Scaling down the frequency will lead to significantly increased transient fault rates. As $f_b > f_a$, the overall reliability value in Fig. 4(b) is larger than that of RAPM scheme in Fig. 4(a). In terms of energy consumption, if we schedule the task appropriately to satisfy the condition $E_a \geq E_b$, then, compared with RAPM, the scheduling from Fig. 4(b) not only enhance system reliability, but also reduce total energy consumption. This means that integrating PMM into the reliability-aware DVFS scheduling can alleviate the negative impact of DVFS on system reliability.

To obtain more efficient scheduling solutions, therefore, we propose an integrated scheme which is called Energy-efficient Fault-tolerant Scheduling (EFS). Shared-recovery is adopted to tolerate transient faults to enhance system reliability. Static schedule should reserve a slack for shared recovery tasks executed on the same processor in advance for any single recovery that may be needed at run-time. The slack reserved for shared-recovery is set to be the largest scaled task on its corresponding processor under DVFS settings. Unlike reliability-aware DVFS scheduling, PMM is introduced to recuperate the reliability loss due to DVFS, and DVFS and PMM are combined to reduce system total energy consumption globally. In the EFS scheme, task recovery jointly with energy optimization techniques are employed to provide reliable energy-efficient scheduling. However, a key issue is that DVFS and PMM aim to minimize total energy consumption that occupy available system slacks, while the shared-recovery require time redundancy that increases energy consumption. These techniques essentially compete for the available system slacks, therefore, the goal of saving system energy must be carefully weighted with the goal of enhancing system reliability. Moreover, the heterogeneity of the system increases the complexity of scheduling with the EFS scheme.

C. MILP FORMULATION WITH EFS SCHEME

To obtain optimal energy-efficient scheduling solutions for pre-mapped tasks with consideration of reliability requirement, we formally present a MILP model with the proposed EFS scheme based on the practical models defined in Section III. First, we define the following variables.

$s^l_{i,m}$: Binary variable, $s^l_{i,m} = 1$ if frequency level of task T_i on u_m is selected as f_l, and 0 otherwise.

$d\_idle_{i,m}$: Binary variable, for the i-th idle interval of each processor u_m, $d\_idle_{i,m} = 1$ if the processor should remain in idle mode, and $d\_idle_{i,m} = 0$ if the processor should enter into sleep mode.

$st_{i,m}$: Start time of task T_i on u_m.

Then, the constraints are listed as follows.

- The shared-recovery adopts single fault assumption and reserves an available slack on each processor. The recovery slack size is determined by the largest scaled task. Assuming $shrw_m$ is WCET of the largest scaled task, then the effective deadline of tasks on each processor u_m is calculated as

$$d_m = D - shrw_m. \tag{14}$$

- Frequency level selection constraints for each task T_i in G. We assume that L_m is the effective lowest frequency level. As we adopt inter-task DVFS, each task T_i can be assigned with only one frequency level in the scope of $[f_1, f_2, \ldots, f_{L_m}]$. The constraint is expressed as:

$$\sum_{l=1}^{L_m} s^l_{i,m} = 1 \tag{15}$$

Note that if T_i is not scaled ($s^1_{i,m} = 1$), which means that the task is assigned with the highest frequency level f_1 (or f_{max}), otherwise T_i is selected to be scaled ($s^1_{i,m} = 0$).

- Shared recovery block size on processor u_m is given as:

$$shrw_m = \max_{\forall T_i \; on \; u_m} ((1 - s^1_{i,m}) \times w^1_{i,m}) \tag{16}$$

- Deadline constraints (χ^m denotes time overheads for DVFS and task switch on the processor u_m). As the shared-recovery technique is used to maintain the system original reliability, the deadline constraint is expressed as:

$$st_{i,m} + \sum_{l=1}^{L_m} (s^l_{i,m} \times w^l_{i,m}) + \chi^m \leq d_m. \tag{17}$$

- Precedence constraints. The target task T_j can be started only after the source task T_i completes.

1) For tasks mapped to the same processor u_m (e.g, $st_{i,m}$ and $st_{j,m}$):

$$st_{i,m} + \sum_{l=1}^{L_m} (s^l_{i,m} \times w^l_{i,m}) + \chi^m \leq st_{j,m} \tag{18}$$

2) For tasks mapped to different processors (e.g, $st_{i,m}$ and $st_{j,n}$), inter-processor communication occurs from T_i to T_j.

$$st_{i,m} + \sum_{l=1}^{L_m} (s^l_{i,m} \times w^l_{i,m}) + \chi^m + c_{i,j} \leq st_{j,n} \tag{19}$$

Next, for given task-processor allocation and tasks' priority, the interval between any two adjacent tasks on each processor $u_m \in U$ can be modeled. Assuming that there are N_m tasks on processor u_m ($\sum_{m=0}^{M-1} N_m = N$), all these tasks mapped on u_m are stored in a task list $\{T_1, T_2, \ldots, T_k, \ldots, T_{N_m}\}(1 \leq k \leq N_m)$ where tasks are ordered in descending order of their priorities. For instance, task T_k is in front of task T_{k+1}, which means that T_{k+1} can start only after T_k finishes. Clearly, there are N_m idle intervals for N_m tasks running on processor u_m in a period H. Note that the first and last idle interval are merged into one, as tasks are executed periodically. The interval between any two adjacent tasks on each u_m can be directly presented as follows:

$$\begin{aligned} int_{1,m} &= st_{2,m} - ft_{1,m} \\ &\ldots \\ int_{k,m} &= st_{k+1,m} - ft_{k,m} \\ &\ldots \\ int_{N_m,m} &= [D - ft_{N_m,m}] + [st_{1,m} - 0] \end{aligned} \tag{20}$$

where $st_{k,m}$ and $ft_{k,m}$ represent the start time and finish time of task T_k on u_m, respectively. According to Definition 1, for each idle time interval $int_{k,m}$ on processor u_m,

$$d\_idle_{k,m} = \begin{cases} 1, & \text{if } int_{k,m} < T_{bet} \\ 0, & \text{if } int_{k,m} \geq T_{bet} \end{cases} \tag{21}$$

where $d\_idle_{k,m}$ is a binary variable used to denote the processor mode switch decision. Thus, the total idle and sleep interval for u_m can be calculated as follows:

$$t^m_{idle} = \sum_{k=1}^{N_m} (d\_idle_{k,m} \times int_{k,m}) \tag{22}$$

$$t^m_{sleep} = \sum_{k=1}^{N_m} ((1 - d\_idle_{k,m}) \times int_{k,m}). \tag{23}$$

The total energy overheads of mode switch for u_m can be calculated as follows:

$$E^m_{cov} = \sum_{k=1}^{N_m} ((1 - d\_idle_{k,m}) \times E_{ms}). \tag{24}$$

Lastly, the objective function (total system energy consumption in one period) can be calculated as:

$$\begin{aligned} E_{total} = &\sum_{m=0}^{M-1} (\sum_{l=1}^{L_m} (s^l_{i,m} \times w^l_{i,m}) \times P^m_{active}) \\ &+ \sum_{m=0}^{M-1} (t^m_{idle} \times P^m_{idle}) + \sum_{m=0}^{M-1} (t^m_{sleep} \times P^m_{sleep}) + \sum_{m=0}^{M-1} E^m_{cov}. \end{aligned} \tag{25}$$

Note that the step function introduced by $d\_idle_{k,m}$ in (21) and the multiplication of $int_{k,m}$ and binary variable $d\_idle_{k,m}$ in (22-23) are nonlinear equations. Solutions to similar nonlinear problems have been presented in [15] and [33].

Such problems can be solved by MILP solvers after linearization by using standard techniques. We now present the linearization process for our problem.

To linearize the multiplication of $d\_idle_{k,m} \times int_{k,m}$ in (22-23), we define an auxiliary variable $q_{k,m}$, such that $q_{k,m} = d\_idle_{k,m} \times int_{k,m}$. It is obvious that $int_{k,m} \leq D$. The multiplication can be linearized as the following constraints.

$$q_{k,m} - int_{k,m} \leq 0 \tag{26}$$

$$q_{k,m} - d\_idle_{k,m} \times D \leq 0 \tag{27}$$

$$int_{k,m} - q_{k,m} + d\_idle_{k,m} \times D \leq D \tag{28}$$

The step function introduced by $d\_idle_{k,m}$ in (21) can be transformed to the following constraint:

$$d\_idle_{k,m} \times (T_{bet} - int_{k,m}) + (1 - d\_idle_{k,m}) \times (int_{k,m} - T_{bet}) \geq 0 \tag{29}$$

In (29), the multiplication of $d\_idle_{k,m} \times int_{k,m}$ are linearized by using (26-28).

The MILP based design-time approaches incur high computational costs, especially for large task sets. To solve the MILP efficiently, the linear program with randomized rounding technique is used to generate task schedules by relaxing the integer constraints of the MILP. The technique is shown to yield the best approximation known by any polynomial time algorithm for many NP-hard problems [34], and has been extensively investigated in task scheduling [33] which is similar to us.

V. THE EFMS FRAMEWORK

The Section IV-C provides energy-efficient scheduling solutions for pre-mapped tasks with consideration of reliability requirement. However, two aspects need to be addressed cooperatively when designing energy-efficient fault-tolerant systems. First, the mapping, which determines task-processor allocation, should be able to create more potentials of exploiting energy saving and improving reliability for the scheduling process, where energy management and task recovery techniques are applied. Second, the scheduling, which determines slack allocation after mapping, should be able to maximize energy savings while preserving the system original reliability efficiently. This motivates us to jointly optimize mapping and scheduling for high reliability and low energy. In this section, we present the EFMS framework and give the implementation of the framework in detail.

A. OUTLINE OF THE EFMS FRAMEWORK

To obtain the solution of the problem, the designer has to determine where and when the tasks should be executed, depending on a set of constraints. We assume that the designer has already acquired information of hardware platform, target application and profiles. The proposed EFMS framework is based on the approach of iterative improvement mapping and scheduling, as shown in Fig. 5. In the framework, the target application is firstly parsed and transformed into a DAG. Then, three iterative steps are: a) Mapper: Generating task-processor allocation and execution order of tasks, b) Scheduler: Generating energy-efficient fault-tolerant frequency assignment and scheduling for the given task mapping from step a) through the EFS scheme, and c) Evaluation: Evaluating total energy consumption of current candidate solutions.

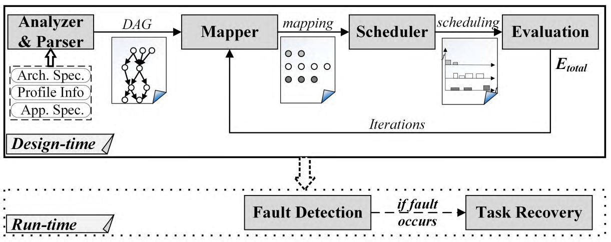


FIGURE 5. Energy-efficient fault-tolerant mapping and scheduling framework.

After the energy-efficient fault-tolerant mapping and scheduling solution is determined in the offline phase, the online operation is the same as that of shared-recovery. Specifically, as long as there are no errors, tasks are executed at the frequencies which are assigned in the offline phase. Once a soft error is detected during the execution of a task on a processor at runtime, the recovery of the faulty task is immediately dispatched in the form of re-execution and executed at f_{max}. At the same time, the system is notified to adopt the contingency schedule, where the concurrently running tasks on other processors continue their executions at the assigned frequency, but any newly dispatched tasks including the faulty task's successor tasks mapped on the same processor should run at f_{max} for reliability preservation until the end of period. With one designed recovery task block on each processor, the shared-recovery technique can guarantee that any faulty execution of the concurrently running tasks can be recovered in time without violating deadline constraints.

B. IMPLEMENTATION OF THE PROPOSED EFMS FRAMEWORK

Due to the NP hardness of mapping and scheduling, finding optimal solutions to minimize energy consumption with reliability management is not trivial. Moreover, scheduling applications on heterogeneous architecture increases the difficulty of the problem. The MILP based methods are able to obtain optimal solutions, but take large solving time. Heuristics are usually preferred due to its capability of exploiting good solutions in reasonable time. Among them, list-based scheduling heuristics are widely accepted [28] and they efficiently reduce the search space by means of greedy heuristics. However, due to the greedy nature, they cannot guarantee the solution quality for different applications. In attempts to obtain solutions of better quality, meta-heuristics have appeared to approach the mapping and scheduling problems by exploring design space and exploiting feedback from previous executions [35]–[38]. They have shown superiority to the one-shot heuristics, despite a longer elaboration time.

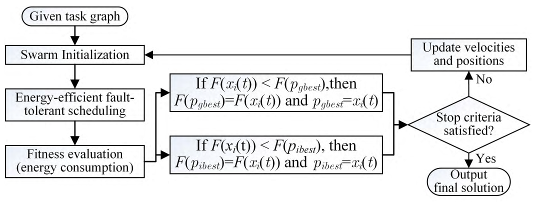

Given task graph
Swarm Initialization
Energy-efficient fault-tolerant scheduling
Fitness evaluation (energy consumption)
If $F(x_i(t)) < F(p_{gbest})$,then $F(p_{gbest})=F(x_i(t))$ and $p_{gbest}=x_i(t)$
If $F(x_i(t)) < F(p_{ibest})$, then $F(p_{ibest})=F(x_i(t))$ and $p_{ibest}=x_i(t)$
Update velocities and positions
No
Stop criteria satisfied?
Yes
Output final solution

FIGURE 6. Flowchart of the LBPSO-EFMS algorithm.

To combine the advantages and overcome the disadvantages of individual heuristics, we consider applying a meta-heuristic algorithm, based on the list scheduling heuristic to solve the problem efficiently. Among the meta-heuristics, PSO algorithm has been demonstrated successfully applied to a wide range of optimization problems. The attractiveness of using PSO is that the algorithm has the following features: natural metaphor, simplicity, stochastic move, adaptivity, positive feedback, and high quality solutions [38]. It is worth noting that existing works have applied PSO-based algorithms to task mapping and scheduling on multiprocessor systems [37], [38]. However, their works focus on deriving system performance or energy optimization only, thereby their approaches are not suitable for our problems. As the nature of the problem is discrete, we exploit the binary version of the PSO (i.e., BPSO), which has been applied to combinatorial problems, to fit our binary selections for processors and explore search space efficiently. The EFMS framework is implemented with the List-based BPSO Energy-efficient Fault-tolerant Mapping and Scheduling (LBPSO-EFMS) algorithm.

Fig. 6 shows the outline of the proposed LBPSO-EFMS algorithm. First, the algorithm generates initial particles. Based on the list scheduling method, we generate stochastic task-processor allocations and task priorities. The position of particles are randomly generated within the hypercube of the feasible space. Then, the algorithm performs energy-efficient scheduling under the EFS scheme for the given feasible particle solutions and evaluates the fitness value of current particles. Next, for each particle, its individual best position p_{ibest} is updated in each iteration when a better position for the particle is obtained, and the global best particle p_{gbest} is updated when a new best position within the whole swarm is found. In each iteration, each particle moves through the optimization space as a function of its velocity. The evolution process is repeated until the stopping criterion is satisfied.

1) INITIAL SWARM GENERATION

We generate the swarm based on a well-known list-based heuristic: Heterogeneous Earliest Finish Time (HEFT) [29]. The heuristic has already been used in reliability and energy-aware task scheduling on multiprocessor systems [9], [11], [12]. It includes two phases: arranging tasks based on descending order of priorities (priority assignment), and allocating task to processors (task-processor allocation). In the HEFT, task priority is determined by the assumption that the average WCET among various processors, and the communication cost between dependent tasks always exists even though they may be allocated to the same processor. However, there is a distinct difference between our approach and the HEFT. That is, we allocate tasks to processors before assigning task priorities. Meanwhile, we can get real computational time of the task on the specified processor and real communication cost between the dependent tasks, as we first allocate tasks to processor. We believe that our approach can present a more accurate estimation of the task priorities.

Algorithm 1 The Swarm Generation Algorithm

Input: A heterogeneous multiprocessor system, an application, task WCETs

Output: s feasible particles in the swarm

```
1:  Initialize the swarm size parameter $s$;
2:  $i \leftarrow 0$, $valid \leftarrow 0$;
3:  while ($i < s$) do
4:    while ($!valid$) do
5:      Generate a particle $x_i$ for the stochastic task-processor allocation;
6:      Task priority assignment based on the task-processor allocation;
7:      Pre-schedule to evaluate the scheduling length $l(x_i)$;

8:      if $l(x_i) > D$ then
9:        $valid \leftarrow 0$;
10:     else
11:       $valid \leftarrow 1; i++$;
12:     end if
13:   end while
14: end while
```

The pseudo-code of the swarm generation is shown in Alg. 1. In the algorithm, the swarm population contains predefined number of s particles. After generating a particle solution x_i, pre-scheduling is performed by completing tasks as soon as possible to check whether the particle is feasible or not. For a given DAG, every task is scheduled until there are no unscheduled tasks. The outer loop in the algorithm is repeated to find s feasible solutions of particles which satisfying precedence and deadline constraints. The two phases of stochastic task-processor allocation and task priority assignments in swarm generation are explained in detail as follows.

a: PARTICLE REPRESENTATION AND STOCHASTIC TASK-PROCESSOR ALLOCATION

In BPSO, each particle is a candidate solution represented by a fixed length binary vector. In this study, the decision for task-processor allocation solution of the underlying problem is transferred into the vector. Thus, each particle structure is implemented with a vector of N elements and each element is an integer value between 1 to M. The i-th integer of each vector represents the index of the processor to which the task T_i is allocated.

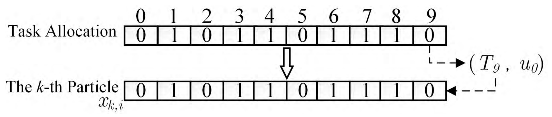


FIGURE 7. **An example of the *k*-th particle representation.**

TABLE 2. **upward rank value of each task in *G*.**

T_i	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
$urank$	106	75	85	79	51	59	49	43	39	21

Example 1: Fig. 7 shows an example for the encoding of a solution corresponding to the task-processor allocation that 10 tasks (indexed as T_0-T_9) are allocated to a dual-processor system (indexed as u_0-u_1). The k-th particle $x_{k,9} = 0$ indicates that task T_9 is allocated to processor u_0.

b: TASK PRIORITY ASSIGNMENTS

In HEFT, upward rank value is the priority assignment rule for DAG list scheduling. In this study, the upward rank value of a task T_i, $urank(T_i)$ is defined recursively as follows:

$$urank(T_i) = w_{i,alloc(T_i)} + \max_{\forall T_j \in succ(T_i)} \{c_{i,j} + urank(T_j)\} \tag{30}$$

where $alloc(T_i)$ represents the index of the processor to which the task T_i is allocated, $w_{i,alloc(T_i)}$ represents the execution time of task T_i on the processor $alloc(T_i)$. We calculate upward rank value of all tasks in the DAG and sort them in a list *ready_taskqueue* in descending order.

Example 2: Consider the case given in Fig. 2. Given mapping $\Theta_{u_0} = \{T_1, T_4, T_8, T_9\}$, $\Theta_{u_1} = \{T_2, T_5, T_7\}$ and $\Theta_{u_2} = \{T_0, T_3, T_6\}$, the *ready_taskqueue* is $\{T_0, T_2, T_3, T_1, T_5, T_4, T_6, T_7, T_8, T_9\}$ according to upward rank value of tasks which are shown in Table 2. Thus, the execution order of tasks on processor u_0, u_1 and u_2 are denoted as $\{T_1 \rightarrow T_4 \rightarrow T_8 \rightarrow T_9\}$, $\{T_2 \rightarrow T_5 \rightarrow T_7\}$ and $\{T_0 \rightarrow T_3 \rightarrow T_6\}$, respectively.

Our swarm generation helps improve the efficiency of the LBPSO-EFMS algorithm in contrast with existing PSO-based algorithms [37], [38] where invalid solutions may also be included in the evolving population. It is based on both global and local heuristics. On one hand, to ensure the exploration of search space, the BPSO is applied to evolve task-processor allocation solutions. On the other hand, to ensure the feasibility of every evolved particle solution, the HEFT heuristic is applied to decide task priorities.

2) ENERGY-EFFICIENT FAULT-TOLERANT SCHEDULING AND FITNESS EVALUATION

In this section, we implement the energy-efficient fault-tolerant scheduler which integrates energy optimization with task recovery techniques for real-time applications under given inputs from the mapping phase. The solutions of the MILP model in Section IV-C can provide energy-efficiency fault-tolerance driven slack allocation. We design the fitness function to evaluate each particle. Given deadline constraint, the fitness value, is defined by

$$F(x_i) = \begin{cases} 1/E_{total}(x_i), & \text{if } solvable \\ 0, & \text{else.} \end{cases} \tag{31}$$

In the fitness function, the total energy consumption E_{total} can be obtained from the output of the MILP model. The model may be unsolvable as the available slack size on each processor u_m may not be large enough to support shared task recovery for each x_i. The fitness value of the particle x_i is equal to zero since the solution is not feasible; otherwise, its fitness value increases when its total energy consumption decreases. Using this fitness function, we can select the particle with the highest fitness value to be the final solution.

3) POSITIONS AND VELOCITIES UPDATING

In each iteration of the loop body, optimized frequency assignment and scheduling is obtained by fitness function calculation. The classical BPSO only considers the possibility of change to 1, and does not consider the possibility of change to 0. In this paper, we apply update rules from the improved BPSO [39] to generate new velocity and position by taking the possibility of change to 0 into account. The readers interested in it can refer to [39].

Time complexity: The LBPSO-EFMS algorithm takes $O(I \times N_P \times c_i)$ time to get results, where I is the number of iterations which can be controlled by designers, N_P is the number of particles and c_i is the evaluation time of a particle.

VI. PERFORMANCE EVALUATION

In this section, we first present the experimental setup of our simulation. Then, we provide the detailed experimental results for the benchmark set. To evaluate and demonstrate the efficiency of our proposed approaches on both energy savings and system reliability, we implement a discrete-event simulator. The proposed LBPSO-EFMS algorithm is implemented in C language, and the simulation is performed on a machine with 3.6 GHz Intel 64-bit processors with 4 GB of RAM. We use the Gurobi v5.60 [40] with CVX 2.1 in Matlab to solve the MILP formulations.

A. EXPERIMENTAL SETUPS

We perform our experiments on both synthetic and realistic applications. The experiments are conducted on a benchmark set including 6 benchmarks represented by task graph TG1-TG6 as shown in Table 3. The number of tasks is varied roughly from 10 to 80. TG1 and TG2 are from [41]. TG1 is embedded consumer electronic applications including tasks such as JPEG compression, JPEG decompression, high pass gray-scale filter, RGB to CYMK conversion, RGB to YIQ conversion, etc. TG2 is automotive applications including tasks such as matrix arithmetic, road speed calculation, table lookup and interpolation, Fast Fourier Transform, etc. TG3 is a real robotic control application for Newton-Euler dynamic control calculation from "Standard Task Graph (STG) [42]."

TABLE 3. Benchmark set.

Benchmark	No. of tasks	No. of edges
TG1	12	12
TG2	20	18
TG3	88	131
TG4	49	59
TG5	33	36
TG6	61	113

Additionally, we use a well-known task-graph generator TGFF [43] to create 3 different periodic applications (TG4-TG6). These task graphs are from the original example input files (e.g., kbasic, kseries-paralle and robtst) that come from the software package. For each task, its WCET is randomly generated by the UUniFast algorithm [44] in the range of [10ms, 100ms]. The out degree of a task is varied from 2 to 7. Among them, TG6 is a slim graph which has very long critical path and a few independent tasks, TG5 is a fat graph which has short critical path and many independent tasks, and TG4 is a medium graph. Table 3 shows the detailed information of each benchmark.

We consider a four-processor architecture for our experiment based on the classical 70nm power model from [30], [31]. The power parameters of processors are consistent with those used in [12], [15], [24], [30], [31], and [45]. In these settings, the time overheads of processor mode switch and frequency change are collected and extracted from [46]. The break-even-time is computed according to formula (5) and the critical frequency $f_{cri} = 0.4$ [47]. The four processors support 5, 6, 4 and 3 normalized discrete frequency levels varying from $f_{min} = 0.4$ to $f_{max} = 1$, respectively. Furthermore, the power values under different lower frequency levels for each processor are normalized with respect to active power $P^{max}_{active} = 1$ under maximum frequency. We set $C_{ef} = 1$ and $m = 3$. The lowest fault rate at maximum processing frequency f_{max} for the four processors are 2e-6, 3e-6, 6e-6 and 6e-6, respectively. They are based on $\lambda_0 = $ 1e-6, which is realistic according to [5], [6], [9], and [12]. For transient faults that follow the Poisson distribution, the average fault arrival rate at the lowest processor speed is assumed to be $d = 3$.

B. EXPERIMENTAL RESULTS

In this section, we present our experimental results to evaluate the performance of our proposed approaches in terms of two main metrics: Average Power Consumption (APC) (defined as APC $= E_{total}/H$) and Probability of Failure (POF) (defined as POF=1-$R(G)$).

1) EVALUATION OF THE EFS SCHEME

We first evaluate and compare our EFS scheme with existing schemes under a given mapping. In the experiment, the task mappings are given by applying list scheduling method (e.g., HEFT) which is consistent with those used in literatures [9], [11], [12]. The scheduling schemes to be contrasted are as follows:

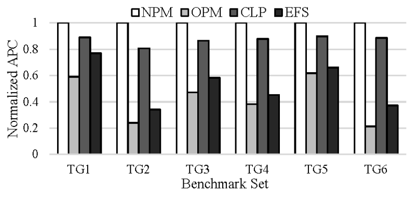
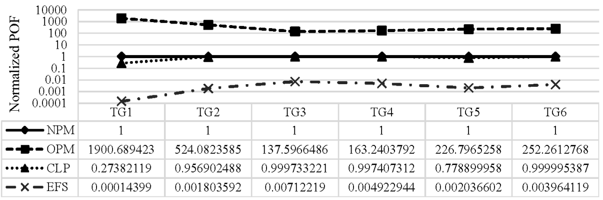


FIGURE 8. Normalized APC for TG1-TG6.



	TG1	TG2	TG3	TG4	TG5	TG6
NPM	1	1	1	1	1	1
OPM	1900.689423	524.0823585	137.5966486	163.2403792	226.7965258	252.2612768
CLP	0.27382119	0.956902488	0.999733221	0.997407312	0.778899958	0.999995387
EFS	0.00014399	0.001803592	0.00712219	0.004922944	0.002036602	0.003964119

FIGURE 9. Normalized POF for TG1-TG6.

- NPM: No Power Management. All tasks execute at f_{max} and processors are always on.
- OPM: Optimal Power Management. Total energy consumption is minimized by DVFS and PMM while reliability degradation is ignored. It is included here to show the maximum achievable energy savings.
- CLP: Constraint Logic Programming-based RAPM approach is used to produce energy-efficient scheduling and voltage/frequency scaling which takes fault tolerance into account in the context of heterogeneous distributed multiprocessor embedded systems [6].

We use NPM as the baseline scheme. Then, the normalized APC and POF are reported. Fig. 8 shows the APC for task graph TG1-TG6 under different static scheduling schemes. Fig. 9 shows the POF of different scheduling schemes. From the simulation results shown in Fig. 8, we can see that OPM has the largest reduction in power consumption, followed by EFS and CLP. The reason for the best results for OPM is that it does not reserve any slack for recovery such that all available slacks can be utilized for DVFS and PMM to reduce total system energy consumption. As can be seen from Fig. 9, OPM can lead to magnitude degradation for system reliability and cannot preserve system reliability, though it is the most efficient in saving energy.

Task recovery technique reserves slacks for system fault tolerance in both CLP-based scheduling and our EFS scheme. As shown in Fig. 8, comparing to CLP-based scheduling, our EFS scheme can on average achieve 39.4% (up to 57.9%) power savings. This is because that our EFS scheme integrates DVFS and PMM globally with fault-tolerant scheduling to optimize total energy consumption, while the CLP-based scheme only apply DVFS to optimize dynamic power consumption and consider the leakage power consumption of the system as a constant factor. Furthermore, one can observe that the POF of EFS and CLP are less than NPM

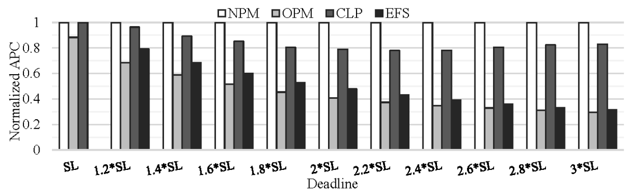
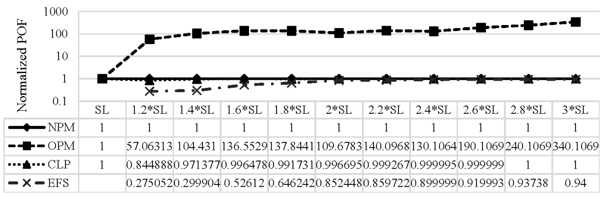


FIGURE 10. The impact of deadline on normalized APC.



	SL	1.2*SL	1.4*SL	1.6*SL	1.8*SL	2*SL	2.2*SL	2.4*SL	2.6*SL	2.8*SL	3*SL
NPM	1	1	1	1	1	1	1	1	1	1	1
OPM	1	57.06313	104.431	136.5529	137.8441	109.6783	140.0968	130.1064	190.1069	240.1069	340.1069
CLP	1	0.844888	0.971377	0.996478	0.991731	0.996695	0.999267	0.999995	0.999999	1	1
EFS		0.275052	0.299904	0.52612	0.646242	0.852448	0.859722	0.899999	0.919993	0.93738	0.94

FIGURE 11. The impact of deadline on normalized POF.

from Fig. 9, which means that they can preserve system's original reliability.

More interestingly, the POF of EFS scheme is significantly less than CLP in Fig. 9. The main reason comes from the fact that scaling down frequency can save energy while weaken reliability. For CLP-based scheme, the POF is sensitive to the frequency change. If we want to improve system reliability (or to reduce POF), then the frequency must be raised, but the energy consumption will certainly grows. While for our EFS scheme, increasing frequency to improve reliability will prolong total available slacks and increase the opportunity of entering low-power state for systems for leakage energy savings. In addition to enhancing the system reliability by increasing frequency, by applying PMM, the decreasing of static power consumption can compensate the growth of dynamic power consumption, or even reduce total system energy consumption. In other words, our EFS scheme which integrates DVFS and PMM with fault-tolerant scheduling can reduce the sensitivity between frequency change and reliability.

Next, we conduct the experiment to show the impact of deadline to the effectiveness of our approach. The deadline is represented by $D = \delta + SL$, where δ is the factor that varies the deadline with respect to the scheduling length SL. We vary the factor δ from 0 to $2 * SL$ with fixed step size $0.2 * SL$. Fig. 10 and Fig. 11 illustrate the normalized APC and POF for benchmark set TG3, respectively. Intuitively, the larger the deadline, the more opportunities for energy optimization techniques to achieve energy savings. Therefore, compared with the existing CLP-based approach, the EFS scheme can obtain significantly more energy savings. The energy improvement becomes more obvious as deadline increases. The reason is that fault-tolerance technique and low-power techniques are competing for slacks in scheduling. When deadline becomes looser, the slacks needed for task recovery can always be satisfied and more slacks can be utilized for DVFS and PMM. Thus, the performance of EFS becomes close to that of OPM, which represents the maximum total energy consumption that can be reduced. Note that when deadline is equal to SL, the EFS scheme cannot be applied since there is no slack for the shared recovery. Also, one can see that the APC of the CLP-based scheduling increases when the deadline is larger than $2.4 * SL$. This is because that the scheme does not consider applying PMM technique to decrease static power consumption which has become an increasingly nonnegligible part as deadline increases.

Again, from Fig. 11, both EFS and CLP-based scheme can preserve system's original reliability. The POF of EFS is smaller than that of CLP-based scheme when the deadline is tight, but it is slowly approaching to the CLP as the deadline constraint becomes loose. The reason is that more aggressive frequency scaling can be performed since more slacks are available as δ increases.

2) EVALUATION OF THE LBPSO-EFMS ALGORITHM

We evaluate of our LBPSO-EFMS algorithm in terms of the APC and POF. Existing algorithms for the energy-efficient fault-tolerant mapping and scheduling problem are as follows:

- SHREERM: Shared-recovery for Energy Efficiency and Reliability Maximization algorithm. The algorithm is first proposed in [11] to address the problem in heterogeneous multiprocessor systems. The algorithm only apply DVFS to minimize energy consumption. For fair comparison, to reduce leakage energy consumption, we modify the SHREERM by applying the PMM technique after the scheduling is generated.
- L-EFMS: List-based method for task mapping which is suggested in [9] and [12] for the problem. In the scheduling phase, EFS scheme is used in L-EFMS.

For the sake of fairness, we perform a Global MILP (denoted as GMILP) for the entire mapping and scheduling problem. We set the running time limit of the MILP model to two hours (i.e., 7200 seconds). We also modify and implement a Genetic Algorithm (denoted as GA-EFMS) because GA and BPSO are both population-based algorithms. Again, in the scheduling phase, EFS scheme is used in the GA-EFMS algorithm. The GA-EFMS uses the same fitness function as used by our HBPSO-EFMS. The parameter values used in both GA-EFMS and LBPSO-EFMS are optimally tuned by intensive preliminary experiments. Specifically, the parameter setting used by GA-EFMS is population size=30, crossover rate=70% and mutation rate=30%. For LBPSO-EFMS, the parameters for the number of particles=30, $\omega = 1$ and $c_1 = c_2 = 2$. Since each independent run of the same algorithm may yield a different result, we take the average of results for 10 independent runs for each problem instance.

We present the comparative performance of normalized APC with respect to SHREERM in Fig. 12. GMILP and LBPSO-EFMS reduce power consumption for the cases of TG1, TG2, TG4, TG5 by 28.98%, 18.26%, 41.98%, 14.57% and 31.27%, 19.01%, 42.04%, 17.36%, respectively,

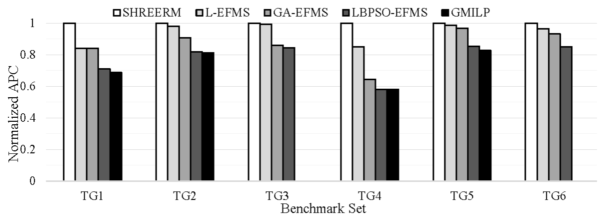
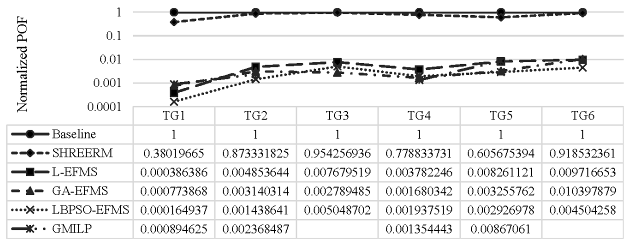


FIGURE 12. Normalized APC of different algorithms.



	TG1	TG2	TG3	TG4	TG5	TG6
Baseline	1	1	1	1	1	1
SHREERM	0.38019665	0.873331825	0.954256936	0.778833731	0.605675394	0.918532361
L-EFMS	0.000386386	0.004853644	0.007679519	0.003782246	0.008261121	0.009716653
GA-EFMS	0.000773868	0.003140314	0.002789485	0.001680342	0.003255762	0.010397879
LBPSO-EFMS	0.000164937	0.001438641	0.005048702	0.001937519	0.002926978	0.004504258
GMILP	0.000894625	0.002368487		0.001354443	0.00867061	

FIGURE 13. Normalized POF of different algorithms.

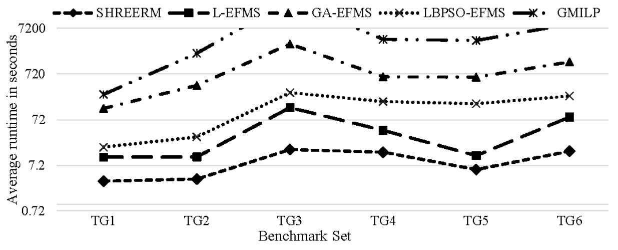


FIGURE 14. Average running time of different algorithms.

compared with SHREERM. For these cases, the average (maximum) deviation of LBPSO-EFMS from GMILP is 1.88% (3.27%). The results show LBPSO-EFMS can find high quality solutions and its performance is close to that of GMILP. However, for some task graphs in the benchmark set (i.e., TG3 and TG6 in the experiments), the GMILP cannot obtain optimal results within the predetermined limited time. The corresponding normalized APC of TG3 and TG6 is not included in Fig. 12 as the running time of GMILP is beyond 2 hours. This demonstrates that the GMILP approach fails to find the optimal solution within acceptable time when the problem size increases. On the other hand, from Fig. 12, the results of SHREERM and L-EFMS are higher than GA-EFMS and LBPSO-EFMS. The LBPSO-EFMS can on average achieve 22.43% and 17.39% energy savings with respect to SHREERM and L-EFMS, respectively. This is mainly due to the fact that the SHREERM and L-EFMS are list-based algorithms, where their mappings are guided by intuitive knowledge such as scheduling length minimization or some other objectives. Such heuristics cannot be effectively exploited to find the most favorable task mapping for the problem. The modified GA-EFMS and our proposed LBPSO-EFMS algorithms are multi-point random search algorithms that based on community. They find the best task mapping according to the navigation of the community iteratively. The differences between the two algorithms are that GA-EFMS uses genetic operators (crossover and mutation) to reproduce random solutions and applies local search strategy to position information directly, while LBPSO-EFMS uses list-based scheduling to generate feasible particles instead. Compared with GA-EFMS in Fig. 12, our proposed LBPSO-EFMS can on average achieve 9.65% power savings.

Again, we present reliability performance of SHREERM, L-EFMS, GA-EFMS, LBPSO-EFMS and GMILP with respect to system's original reliability (Baseline as shown) in Fig. 13. The normalized POF for TG3 and TG6 is not shown when GMILP runs longer than the tolerable time. It is shown that the normalized POF of LBPSO-EFMS is less than 1, which means that it can preserve system's original reliability. The POF of SHREERM is larger than that of L-EFMS, GA-EFMS, LBPSO-EFMS and GMILP, as reliability of SHREERM is more sensitive to frequency change than the algorithms which adopt EFS scheme. Furthermore, we can see that POF varies in L-EFMS, GA-EFMS, LBPSO-EFMS and GMILP. It's not easy to decide which algorithm is the best in terms of reliability. This is because mapping is a factor that affects reliability in addition to frequency. Different mappings obtained by these algorithms determine the diversity of their POF.

In the end, Fig. 14 shows the average running time of these algorithms which have been run for a specified number of evaluations. One can see that the running time of the GMILP is the longest, followed by the GA-EFMS, LBPSO-EFMS, L-EFMS and SHREERM. Although GMILP can obtain optimal solutions, the computation time increases exponentially with increasing size of task graphs. As shown in Fig. 12-Fig. 14, only the case for TG1, TG2, TG4 and TG5 can be solved by GMILP in limited time. While LBPSO-EFMS can always generate high quality solutions efficiently for all cases. The algorithm provides a good way for designers to search for energy-efficient fault-tolerant mapping and scheduling solution when the computation time is intolerable. In addition, compared with GA-EFMS, LBPSO-EFMS takes less computation time since it has a stronger ability of finding feasible solutions and no processes of reproduction, crossover and mutation operation. Obviously, due to the greedy nature, the computation time of list-based algorithms (SHREERM and L-EFMS) are much smaller than meta-heuristic based algorithms (GA-EFMS and LBPSO-EFMS). Nevertheless, they cannot ensure the solution quality while LBPSO-EFMS has the highest quality of solutions which has been illustrated in Fig. 12 and Fig. 13. As the optimization is performed at design-time in this paper and the computation time is acceptable, we can conclude that our LBPSO-EFMS algorithm achieves the best trade-off between solution quality and computation time and outperforms the existing algorithms.

VII. CONCLUSION

This paper has addressed the problem of mapping and scheduling real-time applications on heterogeneous multiprocessor systems for minimizing total energy consumption and preserving system fault tolerance. An energy-efficient fault-tolerant scheduling scheme called EFS which combines the PMM with reliability-aware DVFS has been proposed to mitigate the negative impact of DVFS on system reliability. In the scheme, total system energy is saved by DVFS and PMM, and fault tolerance is achieved by shared recovery technique. Based on the proposed scheme, we have built MILP formulations to obtain optimal energy-efficient reliability-guaranteed scheduling for pre-mapped applications. We have also developed an energy-efficient fault-tolerant mapping and scheduling framework called EFMS and implemented it by a List-based BPSO algorithm called LBPSO-EFMS to obtain high quality solutions. Experimental results have demonstrated that our proposed approaches are useful and promising for improving energy saving and system reliability over other approaches. As the future work, we are developing alternative heuristic methods with low complexity. Furthermore, we consider to extend our approaches to dynamic priority scheduling approaches to yield more effective reliability-guaranteed scheduling for energy savings, as the static approaches assume that tasks are scheduled with fixed priority and executed with WCET but in fact many tasks complete earlier than their WCETs at run-time.

APPENDIX

ABBREVIATIONS AND ACRONYMS

The following abbreviations and acronyms are used within this paper. They are collected together here to act as a reminder wherever the context is not clear.

DVFS: Dynamic Voltage Frequency Scaling

PMM: Power Mode Management

DAG: Directed Acyclic Graph

WCET: Worst Case Execution Time

NP: Non-deterministic Polynomial

MILP: Mixed Integer Linear Programming

HEFT: Heterogeneous Earliest Finish Time

PSO: Particle Swarm Optimization

BPSO: Binary Particle Swarm Optimization

EFS: Energy-efficient Fault-tolerant Scheduling

RAPM: Reliability Aware Power Management

EFMS: Energy-efficient Fault-tolerant Mapping and Scheduling

LBPSO: List-based Binary Particle Swarm Optimization

APC: Average Power Consumption

POF: Probability Of Failure

NPM: No Power Management

OPM: Optimal Power Management

CLP: Constraint Logic Programming

SHREERM: Shared Recovery for Energy Efficiency and Reliability Maximization

L-EFMS: List-based Energy-efficient Fault-tolerant Mapping and Scheduling

GA-EFMS: Genetic Algorithm based Energy-efficient Fault-tolerant Mapping and Scheduling

LBPSO-EFMS: LBPSO-based Energy-efficient Fault-tolerant Mapping and Scheduling

REFERENCES

[1] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2006, pp. 397–407.

[2] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. Comput. Syst.*, vol. 4, no. 3, pp. 214–237, 1986.

[3] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2586–2594, Dec. 2000.

[4] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2009, pp. 63–70.

[5] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2004, pp. 35–40.

[6] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *Proc. IEEE/ACM Int. Conf. Hardw./Softw. Code Syst. Synth.*, Sep./Oct. 2007, pp. 233–238.

[7] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Trans. Comput*, vol. 58, no. 10, pp. 1382–1397, Oct. 2009.

[8] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 2, 2013, Art. no. 23.

[9] Y. Guo, D. Zhu, and H. Aydin, "Reliability-aware power management for parallel real-time applications with precedence constraints," in *Proc. Int. Conf. Green Comput.*, 2011, pp. 1–8.

[10] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Proc. ACM/IEEE Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, vol. 40, Mar. 2003, p. 10918.

[11] L. Zhang, K. Li, K. Li, and Y. Xu, "Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems," *Int. J. Elect. Power Energy Syst.*, vol. 78, pp. 499–512, Jun. 2016.

[12] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, and K. Li, "Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Sustain. Comput.*, vol. 3, no. 3, pp. 167–181, Jul./Sep. 2017.

[13] K. Huang, L. Santinelli, J. J. Chen, L. Thiele, and G. C. Buttazzo, "Applying real-time interface and calculus for dynamic power management in hard real-time systems," *Real-Time Syst.*, vol. 47, no. 2, pp. 163–193, 2011.

[14] D. Bernstein, M. Rodeh, and I. Gertner, "On the complexity of scheduling problems for parallel/pipelined machines," *IEEE Trans. Comput.*, vol. 38, no. 9, pp. 1308–1313, Sep. 1989.

[15] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Trans. Embedded Comput. Syst.*, pp. 1–21, 2013.

[16] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in *Proc. ACM/IEEE Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2002, p. 514.

[17] D. Li and J. Wu, *Energy-Efficient Contention-Aware Application Mapping and Scheduling on NoC-Based MPSoCs*. New York, NY, USA: Academic, 2016.

[18] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, Jun. 2002, pp. 183–188.

[19] K. Nirmal, S. Bansal, and R. K. Bansal, "Energy conscious scheduling with controlled threshold for precedence-constrained tasks on heterogeneous clusters," *Concurrent Eng.*, vol. 25, no. 3, pp. 276–286, 2016.

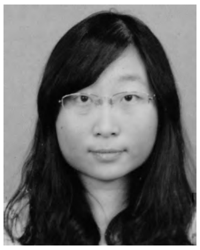
[20] N. Kaur, S. Bansal, and R. K. Bansal, "Energy efficient duplication-based scheduling for precedence constrained tasks on heterogeneous computing cluster," *Multiagent Grid Syst.*, vol. 12, no. 3, pp. 239–252, 2016.

[21] N. Kaur, S. Bansal, and R. K. Bansal, "Duplication-controlled static energy-efficient scheduling on multiprocessor computing system," *Concurrency Comput. Pract. Exper.*, vol. 29, no. 12, p. e4124, 2017.

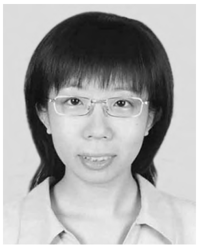
[22] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 31–44, Jan. 2012.
[23] M. E. T. Gerards and J. Kuper, "Optimal DPM and DVFS for frame-based real-time systems," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, 2013, Art. no. 41.
[24] M. A. Awan, P. M. Yomsi, G. Nelissen, and S. M. Petters, "Energy-aware task mapping onto heterogeneous platforms using DVFS and sleep states," *Real-Time Syst.*, vol. 52, no. 4, pp. 450–485, 2016.
[25] M. Fan, Q. Han, and X. Yang, "Energy minimization for on-line real-time scheduling with reliability awareness," *J. Syst. Softw.*, vol. 127, pp. 168–176, 2017.
[26] A. Abdi and H. R. Zarandi, "Hystery: A hybrid scheduling and mapping approach to optimize temperature, energy consumption and lifetime reliability of heterogeneous multiprocessor systems," *J. Supercomput.*, vol. 74, no. 5, pp. 2213–2238, 2018.
[27] M. Lin, Y. Pan, L. T. Yang, M. Guo, and N. Zheng, "Scheduling co-design for reliability and energy in cyber-physical systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 1, no. 2, pp. 353–365, Dec. 2013.
[28] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.
[29] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[30] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, Jul. 2004, pp. 275–280.
[31] W. Wang and P. Mishra, "Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems," in *Proc. Int. Conf. VLSI Design*, 2010, pp. 357–362.
[32] D. Pradhan, *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.
[33] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 9, pp. 1127–1140, Sep. 2008.
[34] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer, 2003.
[35] A. S. Wu, H. Yu, S. Jin, K. C. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 824–834, Sep. 2004.
[36] P.-C. Chang, I-W. Wu, J.-J. Shann, and C.-P. Chung, "ETAHM: An energy-aware task allocation algorithm for heterogeneous multiprocessor," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, Jun. 2008, pp. 776–779.
[37] Q. Kang and H. He, "A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems," *Microprocess. Microsyst.*, vol. 35, no. 1, pp. 10–17, 2011.
[38] P.-Y. Yin, S.-S. Yu, P.-P. Wang, and Y.-T. Wang, "A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems," *Comput. Standards Interfaces*, vol. 28, no. 4, pp. 441–450, 2006.
[39] M. A. Khanesar, M. Teshnehlab, and M. A. Shoorehdeli, "A novel binary particle swarm optimization," in *Proc. Int. Conf. Control Automat.*, 2008, pp. 1–6.
[40] *Gurobi Optimizer*. [Online]. Available: http://www.gurobi.com/
[41] *Embedded System Synthesis Benchmarks Suite (E3S)*. [Online]. Available: http://ziyang.eecs.umich.edu/dickrp/e3s/
[42] *Standard Task Graph Set*. [Online]. Available: http://www.kasahara.elec.waseda.ac.jp/schedule/
[43] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. Int. Workshop Hardw./Softw. Codesign*, 1998, vol. 15, no. 4, pp. 97–101.
[44] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, 2005.
[45] G. Zeng, T. Yokoyama, H. Tomiyama, and H. Takada, "Practical energy-aware scheduling for real-time multiprocessor systems," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2009, pp. 383–392.
[46] (2015). *MPC8536E PowerQUICC III Integrated Processor Hardware Specifications*. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MPC8536EEC.pdf?fsrch=1&sr=1
[47] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2002, pp. 721–725.

KAI HUANG received the B.S.E.E. degree from Nanchang University, Nanchang, China, in 2002, and the Ph.D. degree in engineering circuit and system from Zhejiang University, Hangzhou, China, in 2008. From 2009 to 2011, he was a Post-Doctoral Research Assistant with the Institute of VLSI Design, Zhejiang University. In 2010, he was a Collaborative Expert with the VERIMAG Laboratory, Grenoble, France. Since 2012, he has been an Associate Professor with the Department of Information Science and Electronic Engineering, Zhejiang University. His current research interests include embedded processors and SoC system-level design methodology and platforms.

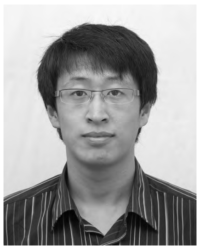
XIAOWEN JIANG is currently pursuing the Ph.D. degree with the Institute of VLSI Design, Zhejiang University, Hangzhou, China. His current research interests include multiprocessor software exploration, real-time systems, energy-efficient scheduling, and fault tolerance.



XIAOMENG ZHANG is currently pursuing the Ph.D. degree with the Institute of VLSI Design, Zhejiang University, China. Her current research interests include multiprocessor software exploration and multi-thread code generation.



RONGJIE YAN received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2007. She was with VERIMAG, Grenoble, France, for two years, where she focused on compositional and incremental verification methodology and correctness-by-construction of component-based systems. She is currently an Assistant Researcher with the Institute of Software, Chinese Academy of Sciences. She is recently involved in extra-function analysis of embedded systems. Her current research interests include modeling and formal verification of embedded systems.



KE WANG is currently pursuing the Ph.D. degree with the Institute of VLSI Design, Zhejiang University, China. His current research interests include energy-efficient task mapping and scheduling algorithm on multiprocessor Soc and low-power IC design methodology.

DONGLIANG XIONG received the B.S. degree from the College of Electrical Engineering, Zhejiang University, Hangzhou, Zhejiang, China, in 2012, and the Ph.D. degree in circuit and system from the Institute of VLSI Design, Zhejiang University, in 2017. His current research interests include QoS memory system design, architecture design, and hardware implementation for deep learning accelerators.

XIAOLANG YAN received the B.S.E.E. and M.S.E.E. degrees from Zhejiang University, Hangzhou, China, in 1968 and 1981, respectively. From 1993 to 1994, he was a Visiting Scholar with Stanford University, Palo Alto, CA, USA. From 1994 to 1999, he was a Professor and the Dean of the Hangzhou Institute of Electronic Engineering, Hangzhou, China. Since 1999, he has been a Professor, the Dean of the Information Science and Engineering College, and the Director of the Institute of VLSI Design, Zhejiang University. His current research interests include embedded CPU design, SoC design methodology, and design for manufacturability.