Hardware Cost Design Optimization for Functional Safety-Critical Parallel Applications on Heterogeneous Distributed Embedded Systems

Guoqi Xie, *Member, IEEE*, Yuekun Chen, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Industrial embedded systems are cost sensitive, and hardware cost of industrial production should be reduced for high profit. The functional safety requirement must be satisfied according to industrial functional safety standards. This study proposes three hardware cost optimization algorithms for functional safety-critical parallel applications on heterogeneous distributed embedded systems during the design phase. The explorative hardware cost optimization (EHCO), enhanced EHCO (EEHCO), and simplified EEHCO (SEEHCO) algorithms are proposed step by step. Experimental results reveal that EEHCO can obtain minimum hardware cost, whereas SEEHCO is efficient for large-scale parallel applications compared with the existing algorithms.**

***Index Terms*—Functional safety, hardware cost, heterogeneous distributed embedded systems, real time, reliability.**

I. Introduction

A. Motivation

HIGH-END industrial embedded systems, such as automotive, avionic, and real-time control systems, are typical heterogeneous distributed embedded systems, where several heterogeneous processors with difference performance are distributed on the same communication bus, such as controller area network (CAN), to form a distributed architecture [1]. The communication among processors is performed through message passing over the bus. In heterogeneous distributed systems, applications (functions) are increasingly parallel, and tasks in an application exhibit evident data dependencies and precedence constraints [1]–[3]. Examples of parallel applications are Gaussian elimination and fast Fourier transform [1], [3]. These two applications have also been implemented in embedded systems [4], [5]. A parallel application with precedence-constrained tasks at a functional level is described by a directed acyclic graph (DAG) [1]–[3], where nodes represent tasks, and edges represent communication messages between tasks. However, the development of industrial embedded systems is usually cost sensitive because they are mass-produced industrial products. Reducing the hardware cost can greatly save the cost of industrial production for high profit. Although the distributed architecture has drastically reduced the hardware cost of wiring harness, the number of processors required for application execution still generates significant hardware cost. In addition, with the application scale continuing to grow, the required processor hardware cost is correspondingly increased. Currently, the unit price of processors for CAN interfaces are from \$25 to \$110 [6]. In fact, some unnecessary processors can be removed as long as such operation does not affect the correct execution of the application. Therefore, it is important to reduce hardware cost by minimizing the number of processors [7].

Meanwhile, functional safety has become the preferential direction of industrial embedded system development, and it refers to the absence of unreasonable risk caused by systematic failures and random hardware failures [8]. A safety-critical embedded application must comply with related functional safety standards, such as ISO 26262 for automotive systems [9], DO-178B for avionics systems, and IEC 61508 for all kinds of industrial embedded systems [10]. Abnormal execution and missed deadline are considered typical random hardware and systematic failures, respectively. That is, besides normal real-time requirement (also called response time requirement, timing constraint, and deadline constraint), reliability requirement (RR) (also called

Manuscript received May 8, 2017; revised October 9, 2017; accepted October 26, 2017. Date of publication November 2, 2017; date of current version June 1, 2018. This work was supported in part by the National Key Research and Development Plan of China under Grant 2016YFB0200405, in part by the National Natural Science Foundation of China under Grant 61702172, Grant 61672217, Grant 61173036, Grant 61379115, Grant 61402170, Grant 61370097, Grant 61502162, and Grant 61502405, in part by the CCF-Venustech Open Research Fund under Grant CCF-VenustechRP2017012, in part by the CERNET Innovation Project under Grant NGII20161003, and in part by the China Postdoctoral Science Foundation under Grant 2016M592422. Paper no. TII-17-0986. *(Corresponding author: Renfa Li.)*

G. Xie, Y. Chen, and R. Li are with the College of Computer Science and Electronic Engineering and the Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, Changsha 410082, China (e-mail: xgqman@hnu.edu.cn; chenyuekun@126.com; lirenfa@hnu.edu.cn).

K. Li with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz NY 12561 USA (e-mail: lik@newpaltz.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2017.2768075

reliability goal, reliability assurance, and reliability constraint) must also be satisfied to ensure application's functional safety according to related functional safety standards. Therefore, it is necessary to reduce hardware costs while satisfying the functional safety requirement, which includes the real-time requirement and the RR together.

B. Our Contributions

A development life cycle of safety-critical embedded systems usually involves analysis, design, implementation, and testing phases [1]. The current study focuses on the design phase. The main contribution of this study is to reduce hardware cost while satisfying the functional safety requirement of a DAG-based parallel application on heterogeneous distributed embedded systems. The details are as follows.

1) An explorative hardware cost optimization (EHCO) algorithm is proposed by iteratively removing the processors, without which the minimum hardware costs can be generated while satisfying application's functional safety requirement.
2) Considering that partial EHCO-generated results can satisfy the real-time requirement, but may not satisfy the RR, a reliability enhancement (RE) algorithm is proposed to enhance application's reliability value without violating the precedence constraints among tasks and real-time requirements of the application, and thereby, we propose an enhanced EHCO (EEHCO) algorithm to improve the possibility that functional safety requirement is satisfied.
3) Considering that both EHCO and EEHCO have high time complexity and thereby require large computation effort for large-scale parallel applications, a simplified EEHCO (SEEHCO) algorithm is proposed to adapt large-scale parallel application's hardware cost design optimization.

II. Related Work

Depending on the application, an industrial embedded system development must comply with one or more standards like IEC 61499, IEC 61508, IEC 62061, IEC 61511, and ISO 26262 [8], [11]. This study mainly reviews related research on functional safety and cost optimization of a DAG-based parallel application.

Response time minimization and reliability maximization are conflicting, and optimizing them is a bicriteria minimization problem [12], [13]. In [14] and [15], the authors presented the maximum reliability (MaxRe) and least resources to satisfy the RR algorithms to reduce the resource cost for a parallel application by minimizing the task redundancy on heterogeneous distributed systems using fault tolerance. In [12], the authors presented a bicriteria scheduling heuristic to generate an approximate Pareto curve of nondominated solutions, among which the designers can verify the functional safety requirement by finding the points that satisfy the RR and the real-time requirement simultaneously. In [13], the authors also studied the same problem as MaxRe and RR and proposed the heuristic replication for redundancy minimization method, which exhibited a significant improvement in resource cost reduction. Considering the limited resources of embedded systems, fault tolerance may be unsuitable [1]. In [1], resource cost is presented to satisfy the RR by transferring the RR of the application to each task without using fault tolerance.

Besides resource cost, the development cost and hardware cost design optimization for safety-critical parallel applications were studied in [7], [16], and [17]. In [16] and [17], the authors presented development cost minimization to satisfy the real-time requirement for parallel applications by presenting genetic algorithm-based and tabu-search-based metaheuristics, respectively. In [7], the authors presented hardware cost minimization to satisfy the real-time and security requirements for a parallel application by presenting integer linear programming and heuristics, respectively. Despite the introduction of hardware cost in [7], it focuses on the security requirement rather than the RR. However, functional safety requirement simultaneously includes the RR and the real-time requirement, and they must be simultaneously satisfied according to functional safety standards.

TABLE I
NOTATIONS IN THIS STUDY

Notation	Definition
$w_{i,k}$	WCET of the task n_i on the processor p_k
$c_{i,j}$	WCRT between the tasks n_i and n_j
$\text{rank}_\text{u}(n_i)$	Upward rank value of the task n_i
$\|X\|$	Size of the set X
price_k	Unit price the processor p_k
λ_k	Failure rate of the processor p_k
$R(n_i, p_k)$	Reliability of the task n_i on the processor p_k
$R_\text{max}(n_i)$	Maximum reliability of the task n_i
$R_\text{min}(n_i)$	Minimum reliability of the task n_i
$R(n_i)$	Reliability of the task n_i
$p_{pr(i)}$	Assigned processor of the task n_i
$\text{EST}(n_i, p_k)$	Earliest start time of the task n_i on the processor p_k
$\text{EFT}(n_i, p_k)$	Earliest finish time of the task n_i on the processor p_k
$\text{LFT}(n_i, p_k)$	Latest finish time of the task n_i on the processor p_k
$\text{AST}(n_i)$	Actual start time of the task n_i
$\text{AFT}(n_i)$	Actual finish time of the task n_i
$\text{HC}(G)$	Hardware cost of the application G
$R(G)$	Reliability of the application G
$R_\text{max}(G)$	Maximum reliability of the application G
$R_\text{min}(G)$	Minimum reliability of the application G
$R_\text{req}(G)$	Reliability requirement of the application G
$\text{LB}(G)$	Lower bound of the application G
$\text{RT}(G)$	Response time of the application G
$\text{RT}_\text{req}(G)$	Real time of the application G

III. Models

Table I lists the notations and their definitions that are used in this study.

A. System Architecture

This study considers a distributed architecture where several processors are mounted on the same CAN bus [1]. Each processor contains a central processing unit (CPU), random access memory and nonvolatile memory, and a network interface card [17]. A task executed completely in one processor sends messages to all its successor tasks, which may be located in the

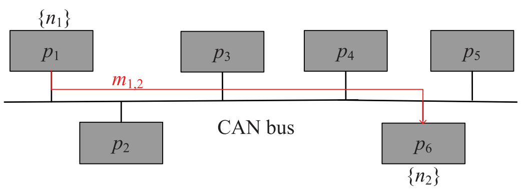

Fig. 1. Heterogeneous distributed embedded system architecture.

different processors of different buses. For example, task n_1 is executed on processor p_1 of CAN$_1$. It then sends a message $m_{1,2}$ to its successor task n_2 located in p_6 of CAN$_3$ (see Fig. 1). $P = \{p_1, p_2, ..., p_{|P|}\}$ represents a set of heterogeneous processors, where $|P|$ represents the size of set P. For any set X, this study uses $|X|$ to denote its size.

B. Application Model

A parallel application is represented by a DAG $G = (N, W, M, C)$ [1], [2], [13], [16], [17].

1) N represents a set of tasks in G, and $n_i \in N$ represents the ith task of G. pred(n_i) represents the set of the immediate predecessor tasks of n_i, whereas succ(n_i) represents the set of the immediate successor tasks of n_i. The task with no predecessor task is denoted by n_{entry}, whereas the task with no successor task is denoted by n_{exit}. If a DAG-based application has multiple n_{entry} or multiple n_{exit} tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph. Each task $n_i \in N$ has different worse-case execution time (WCET) values on different processors due to the heterogeneity of processors. W is an $|N| \times |P|$ cube, where $w_{i,k}$ denotes the WCET of n_i on the processor p_k. All the WCETs of the tasks are determined through the WCET analysis method during the analysis phase [1].
2) The communication between tasks mapped to different processors is performed through message passing over the bus. Hence, M is a set of communication edges, and each edge $m_{i,j} \in M$ represents the communication message from n_i to n_j. Accordingly, $c_{i,j} \in C$ represents the worst-case response time (WCRT) of $m_{i,j}$ [1]. All the WCRTs of the messages are also determined through the WCRT analysis method during the analysis phase [1].

The scheduling can be either static or dynamic and preemptive or nonpreemptive [18]. The real-time application can be soft or hard [18]. This study considers the nonpreemptive static scheduling for a hard real-time application during the design phase. Fig. 2 shows a motivating parallel application with tasks and messages [1]. The example shows ten tasks executed on three processors $\{p_1, p_2, p_3\}$. The weight 18 of the edge between n_1 and n_2 represents WCRT, denoted by $c_{1,2}$ if n_1 and n_2 are not assigned to the same processor. Table II is the WCET matrix $|N| \times |P|$ of tasks on different processors of the motivating parallel application. For example, the weight 14 of n_1 and p_1 in Table II represents WCET of n_1 on p_1, denoted by $w_{1,1} = 14$. It can be seen that the same task has different WCETs on different processors due to the heterogeneity of the processors. The motivating example will be used to explain the proposed algorithms

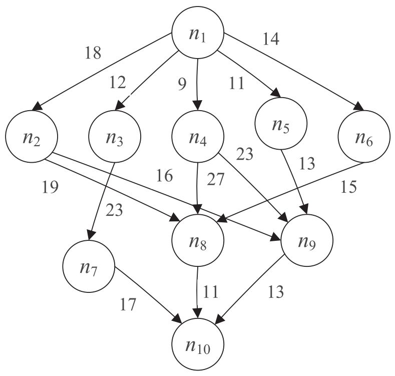

Fig. 2. Motivating example of a DAG-based parallel application with ten tasks [1], [3], [19].

TABLE II
WCETs of Tasks on Different Processors of the Motivating Parallel Application [1], [3], [19]

Task	p_1	p_2	p_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

TABLE III
Processor Parameters for the Motivating Parallel Application

Parameters/processors	p_1	p_2	p_3
price$_k$	20	10	30
λ_k	0.00055	0.0005	0.0004

in this paper. For simplicity, all the units of all parameters are ignored in the example.

C. Hardware Cost Model

As heterogeneous processors are used in this study, all processors have individual unit prices. Hence, let $\{\text{price}_1, \text{price}_2, ..., \text{price}_{|P|}\}$ represent the set of unit prices of heterogeneous processors. Table III shows that the hardware costs of the processors p_1, p_2, and p_3 are 20, 10, and 30, respectively.

The hardware cost of the application is the sum of those of all active processors, and it is calculated by

$$\text{HC}(G) = \sum_{p_k \in P_{\text{active}}} \text{price}_k.$$

D. Reliability Model and RR Assessment

There are two major temporal types of failures, namely, the transient failure (i.e., random hardware failures) and the permanent failure [1], [12], [13]. This study only considers the transient failure of processors because some functional safety standards, such as ISO 26262 and IEC 61508, only combine the random hardware failures and reliability together [1], [13]. As pointed out in ISO 26262, random hardware failures occur unpredictably during the life cycle of a hardware but follow a probability distribution [1]. In general, transient failures for a task in a DAG-based parallel application follow the Poisson distribution [1], [12], [13]. Let λ_k represent the constant failure rate per time unit of the processor p_k, and the reliability of n_i executed on p_k and its WCET is donated by

$$R(n_i, p_k) = e^{-\lambda_k w_{i,k}}. \tag{1}$$

Similar to the WCRTs of messages and WCETs of tasks, this study assumes that the failure rates are known and have been obtained in the analysis phase. The reliability of the DAG-based parallel application is calculated by [1], [12], [13]

$$R(G) = \prod_{n_i \in N} R(n_i) = \prod_{n_i \in N} R(n_i, p_{\text{pr}(n_i)}) \tag{2}$$

where $p_{\text{pr}(n_i)}$ represents the assigned processor of n_i. As the CAN bus has high fault-tolerance capacity, this study only considers processor failure and does not include the communication failure into the problem (i.e., communication is reliable in this study).

As the WCET of each task on each processor has been determined by the WCET analysis method during the analysis phase, the MaxRe value of task n_i can be obtained by traversing all the processors, and it is calculated by

$$R_{\max}(n_i) = \max_{p_k \in P} R(n_i, p_k). \tag{3}$$

As the reliability of application G is the product of reliability values of all the tasks [see (2)], the MaxRe value of application G is calculated by

$$R_{\max}(G) = \prod_{n_i \in N} R_{\max}(n_i). \tag{4}$$

The RR $R_{\text{req}}(G)$ must be less than or equal to $R_{\max}(G)$; otherwise, $R_{\text{req}}(G)$ cannot always be satisfied. Hence, $R_{\text{req}}(G)$ must have the following constraint:

$$R_{\text{req}}(G) \leqslant R_{\max}(G). \tag{5}$$

Otherwise, the RR assessment cannot be passed. Table III shows that the failure rates of processors p_1, p_2, and p_3 are $\lambda_1 = 0.00055$, $\lambda_2 = 0.0005$, and $\lambda_3 = 0.0004$, respectively. The MaxRe value is calculated as $R_{\max}(G) = 0.956476$. Let RR $R_{\min}(G) = 0.95$, which can pass the assessment, because

$$0.95 \leqslant 0.956476$$

according to (5) of the RR assessment.

E. Lower Bound and Real-Time Requirement Assessment

Besides RR assessment, lower bound and real-time requirement must also be assessed. The lower bound refers to the minimum response time of a parallel application when all tasks are executed on the processors using a well-studied DAG scheduling algorithm. Considering that scheduling task with quality-of-service (QoS) requirement for optimality on multiprocessors is known to be an NP-hard optimization problem [20], obtaining a lower bound of a parallel application is an NP-hard optimization problem [19]. Therefore, a convincing and well-studied algorithm must be employed to obtain the lower bound. The heterogeneous earliest finish time (HEFT) algorithm presented in [19] is a well-studied scheduling algorithm to obtain the lower bound and has been applied to parallel application [3], [21]. This study also uses the HEFT algorithm to assess the lower bound. The concrete process is as follows.

TABLE IV
UPWARD RANK VALUES FOR TASKS OF THE MOTIVATING PARALLEL APPLICATION

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$\text{rank}_\text{u}(n_i)$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7

First, the HEFT algorithm uses the upward rank value (rank_u) of a task given by (6) as the common task priority standard. In this case, the tasks are ordered according to the decreasing order of rank_u. Table IV shows the upward rank values of all the tasks in the motivating example, which are obtained with (6):

$$\text{rank}_\text{u}(n_i) = \overline{w_i} + \max_{n_j \in \text{succ}(n_i)} \{c_{i,j} + \text{rank}_\text{u}(n_j)\}. \tag{6}$$

Note that only if all the predecessor tasks of n_i have been assigned to the processors, n_i will prepare to be assigned. Assume that two tasks n_i and n_j satisfy $\text{rank}_\text{u}(n_i) > \text{rank}_\text{u}(n_j)$; if no precedence constraint exists between n_i and n_j, then n_i may not have higher priority than n_j. Finally, the task priority in G is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

Second, the attributes $\text{EST}(n_i, p_k)$ and $\text{EFT}(n_i, p_k)$ represent the earliest start time (EST) and the earliest finish time (EFT), respectively, of task n_i on processor p_k. $\text{EFT}(n_i, p_k)$ is considered as the common task assignment criterion because it can meet the local optimal of each precedence-constrained task by using the greedy policy. The aforementioned attributes are calculated as follows:

$$\begin{cases} \text{EST}(n_{\text{entry}}, p_k) = 0 \\ \text{EST}(n_i, p_k) = \max\left(\text{avail}[p_k], \max_{n_h \in \text{pred}(n_i)} \{\text{AFT}(n_h) + c'_{h,i}\}\right) \end{cases} \tag{7}$$

and

$$\text{EFT}(n_i, p_k) = \text{EST}(n_i, p_k) + w_{i,k}. \tag{8}$$

$\text{avail}[p_k]$ is the earliest available time when processor p_k is ready for task execution. $\text{AFT}(n_h)$ is the actual finish time of task n_h. n_i is assigned to the processor with the minimum EFT by using the insertion-based scheduling policy that n_i can be inserted into the slack with the minimum EFT.

Fig. 3 shows the Gantt chart of parallel application G (see Fig. 2) using the HEFT algorithm. The lower bound is obtained as $\text{LB}(G) = 80$. The arrows in Fig. 3 represent the generated communication between tasks.

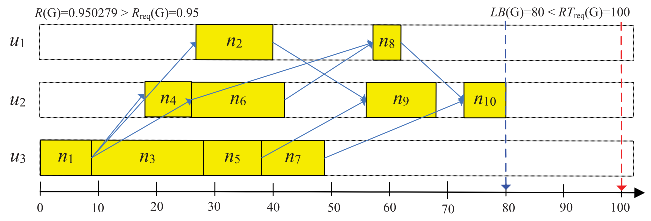

Fig. 3. Lower bound calculation using HEFT.

A known real-time requirement $\mathrm{RT}_{\mathrm{req}}(G)$ (i.e., deadline) is then provided for the application on the basis of the actual physical time requirement after hazard analysis and risk assessment by certificate authority (CA) (see Fig. 3). The concrete hazard analysis and risk assessment are not discussed in this paper because this study mainly focuses on hardware cost optimization. $\mathrm{RT}_{\mathrm{req}}(G)$ must be larger than or equal to $\mathrm{LB}(G)$; otherwise, $\mathrm{RT}_{\mathrm{req}}(G)$ cannot always be satisfied based on the HEFT assessment algorithm. Hence, $\mathrm{RT}_{\mathrm{req}}(G)$ must derive the following constraint:

$$\mathrm{LB}(G) \leqslant \mathrm{RT}_{\mathrm{req}}(G). \tag{9}$$

Otherwise, the real-time requirement assessment cannot be passed. For this motivating example, let the real-time requirement be $\mathrm{RT}_{\mathrm{req}}(G) = 100$, which can pass the assessment, because

$$80 \leqslant 100$$

according to (9) of the RR assessment.

F. Problem Statement

The problem to be addressed is to find the processor assignments of all tasks to minimize the hardware cost:

$$\mathrm{HC}(G) = \sum_{p_k \in P_{\text{active}}} \text{price}_k$$

while satisfying the real-time requirement

$$\mathrm{RT}(G) = \mathrm{AFT}(n_{\text{exit}}) \leqslant \mathrm{RT}_{\mathrm{req}}(G) \tag{10}$$

and the RR:

$$R(G) \geqslant R_{\mathrm{req}}(G). \tag{11}$$

As stated in Section III-E, scheduling tasks with QoS requirement for optimality on multiprocessors is known to be an NP-hard optimization problem [20].

IV. Exploratory Hardware Cost Optimization

A. Iterative Hardware Cost Optimization (IHCO) Algorithm

As this study aims to minimize the total hardware cost of a parallel application, reducing the use of the processor is an effective way. Therefore, we should remove as many as processors and assign the tasks of the parallel application to the remaining processors while still satisfying the functional safety requirement.

Removing processor has been employed in energy consumption reduction. In [22], the proposed method iteratively removes as many processors with a small number of assigned tasks to reduce the energy consumption until the real-time requirement cannot be satisfied. The idea of [22] can also be applied to this study as long as the method iteratively removes as many processors with a high hardware cost until application's real-time requirement cannot be satisfied. Such an approach is named as IHCO in this study. Considering the motivating example that processor p_3 has the highest cost of 30 among $\{p_1, p_2, p_3\}$, as shown in Fig. 4, we must remove p_3 from $\{p_1, p_2, p_3\}$ and assign all the tasks of the application to p_1 and p_2. Of course, we can let p_3 to be a sleep state rather than really removing it from the system during the design phase. Fig. 4 shows the HEFT-generated task mapping on p_1 and p_2 when p_3 is in the sleep state. We can obtain the response time of 82, which can satisfy the real-time requirement of 100. However, the obtained reliability value calculated by (2) is 0.946816, which cannot satisfy the RR of 0.95. In other words, removing p_3 from the processor set is infeasible, and we cannot obtain a valid hardware cost value for the application in this case. Therefore, iteratively removing as many processors with a high hardware cost is not optimized design and we need to propose better methods.

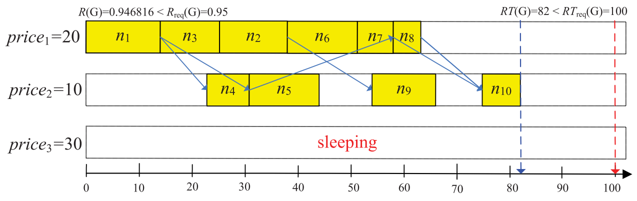

Fig. 4. HEFT-generated task mapping on p_1 and p_2 when p_3 is in the sleep state.

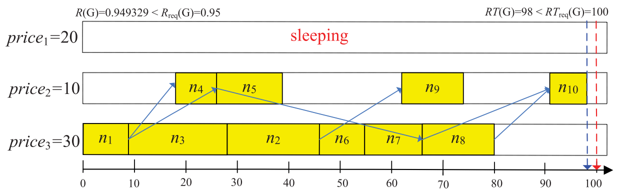

Fig. 5. HEFT-generated task mapping on p_2 and p_3 when p_1 is in the sleep state.

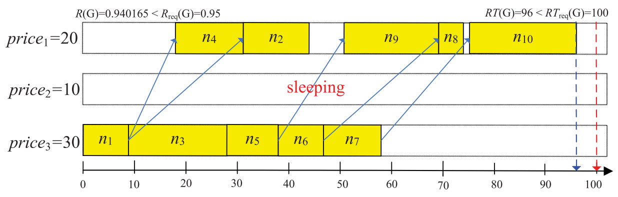

Fig. 6. HEFT-generated task mapping on p_1 and p_3 when p_2 is in the sleep state.

B. EHCO Algorithm

In view of the limitations of the IHCO, this subsection proposes an exploratory method. Consider the same motivating example as in Fig. 4; if p_1 or p_2 is in the sleep state instead of the p_3, then individual task mappings are shown in Figs. 5 and 6, respectively.

Algorithm 1: EHCO Algorithm.

```
Input: $P = \{p_1, p_2, ..., p_{|P|}\}$, $G$, and $\mathrm{RT}_{\mathrm{req}}(G)$, $R_{\mathrm{req}}(G)$
Output: $\mathrm{RT}(G)$, $R(G)$, and $\mathrm{HC}(G)$
 1: $P_{\mathrm{active}} \leftarrow P$;
 2: Invoke the HEFT algorithm on $P_{\mathrm{active}}$ to obtain the
    initial $\mathrm{RT}(G) \leftarrow \mathrm{LB}(G)$, $R(G)$, and $\mathrm{HC}(G)$;
 3: if ($\mathrm{RT}(G) > \mathrm{RT}_{\mathrm{req}}(G) || R(G) < R_{\mathrm{req}}(G)$) then
 4:     return false;
 5: end if
 6: while ($P_{\mathrm{active}}$ is not null) do
 7:     for (each active processor $p_k \in P_{\mathrm{active}}$) do
 8:         Let $p_k$ be the sleep state;
 9:         Invoke the HEFT algorithm on $P_{\mathrm{active}} - \{p_k\}$ to
            obtain $\mathrm{RT}_k(G)$, $R_k(G)$, and $\mathrm{HC}_k(G)$;
10:     end for
11:     if (no result satisfies $\mathrm{RT}_k(G) \leqslant \mathrm{RT}_{\mathrm{req}}(G)$ &&
        $R_k(G) \geqslant R_{\mathrm{req}}(G)$) then
12:         return true;
13:     else
14:         Obtain the processor $p_{\min}$ that has $\mathrm{HC}_{\min}(G) = \min_{p_k \in P_{\mathrm{active}}, \mathrm{RT}_k(G) \leqslant \mathrm{RT}_{\mathrm{req}}(G), R(G) \geqslant R_{\mathrm{req}}(G)} \{\mathrm{HC}_k(G)\}$
            using (12);
15:         $P_{\mathrm{active}} \leftarrow (P_{\mathrm{active}} - \{p_{\min}\})$;
16:         $\mathrm{RT}(G) \leftarrow \mathrm{RT}_{\min}(G)$, $R(G) \leftarrow R_{\min}(G)$, and
            $\mathrm{HC}(G) \leftarrow \mathrm{HC}_{\min}(G)$.
17:     end if
18: end while
```

We can see that Figs. 5 and 6 obtain higher reliability values than Fig. 4 while satisfying the real-time requirement. Even Figs. 5 and 6 still cannot satisfy the RR of 0.95, their results indicate that removing the processor with high hardware cost does not necessarily lead to the reliability enhance of the application.

Inspired by the aforementioned analysis, we present the exploratory method called EHCO described in Algorithm 1.

The main idea of the EHCO algorithm is that it iteratively removes the processors, without which the minimum hardware cost is generated while satisfying application's functional safety requirement from active processors P_{active} until application's functional safety requirement cannot be satisfied. The details of EHCO are explained as follows.

1) In Line 1, all processors are active, namely, $P_{\mathrm{active}} \leftarrow P$.
2) In Line 2, EHCO invokes the HEFT algorithm on all the processors to obtain the initial response time $\mathrm{RT}(G) \leftarrow \mathrm{LB}(G)$, reliability value $R(G)$, and hardware cost $\mathrm{HC}(G)$. If the functional safety requirement cannot be satisfied in this case, then EHCO directly returns false (i.e., hardware cost optimization is failed) (lines 3–5). In the following, EHCO executes the iterative explorative process in lines 6–18.
3) In lines 7–10, EHCO makes an explorative process for each active processor by invoking the HEFT algorithm on $P_{\mathrm{active}} - \{p_k\}$ to obtain the response time $\mathrm{RT}_k(G)$, reliability value $R_k(G)$, and hardware cost $\mathrm{HC}_k(G)$ of the application G on the active processors when the processor p_k is the sleep state.
4) If no result of the above explorative process satisfies application's functional safety requirement, then EHCO directly returns true and the hardware cost optimization is end (lines 11–13); otherwise, EHCO removes the processor $p_{\min}$, without which the minimum hardware cost $\mathrm{HC}_{\min}(G)$ is generated while satisfying application's functional safety requirement (lines 14 and 15). $\mathrm{HC}_{\min}(G)$ can be found by

$$\mathrm{HC}_{\min}(G) = \min_{p_k \in P_{\mathrm{active}}, \mathrm{RT}_k(G) \leqslant \mathrm{RT}_{\mathrm{req}}(G), R(G) \geqslant R_{\mathrm{req}}(G)} \times \{\mathrm{HC}_k(G)\} \tag{12}$$

where P_{active} represents the active processor set excluding the already removed processors as mentioned earlier.
5) EHCO updates application's response time $\mathrm{RT}(G)$, reliability value $R(G)$, and hardware cost $\mathrm{HC}(G)$ (Line 16).

The time complexity of the EHCO algorithm is $O(|N|^2 \times |P|^3)$. The details are as follows.

1) The maximum number of removed processors is $O(|P|)$ time (lines 6–18).
2) Traversing all processors can be done in $O(|P|)$ time (lines 7–10).
3) Invoking the HEFT algorithm must be done in $O(|N|^2 \times |P|)$ time (Line 9).

TABLE V
EHCO-GENERATED SCHEDULE RESULTS

Sleeping processor	Response time $\mathrm{RT}(G)$	Reliability value $R(G)$	Satisfying functional safety requirement?	Hardware cost $\mathrm{HC}(G)$
p_1	98	0.949329	No	–
p_2	96	0.940165	No	–
p_3	82	0.946816	No	–

C. Example Results Using EHCO

Table V shows the EHCO-generated schedule results when each processor is in the sleep state. It can be seen that each case can satisfy the real-time requirement because their individual response time values are less than 100, but cannot satisfy the RR because their individual reliability values are less than 0.95. Therefore, for motivating example, regardless of which processor is removed, the application's functional security requirement cannot be satisfied.

V. Enhanced Exploratory Hardware Cost Optimization

A. EEHCO Algorithm

By observing the reliability values in Table V, we find that they are only small distances to the RR of 0.95. If we can enhance the reliability value without violating the real-time requirements of the application, it is possible to satisfy the RR and remove

Algorithm 2: EEHCO Algorithm.

```
Input: $P = \{p_1, p_2, ..., p_{|P|}\}$, $G$, and $\mathrm{RT_{req}}(G)$, $R_{\mathrm{req}}(G)$
Output: $\mathrm{RT}(G)$, $R(G)$, and $\mathrm{HC}(G)$
 1: $P_{\mathrm{active}} \leftarrow P$;
 2: Invoke the HEFT algorithm on $P_{\mathrm{active}}$ to obtain the
    initial $\mathrm{RT}(G) \leftarrow \mathrm{LB}(G)$, $R(G)$, and $\mathrm{HC}(G)$;
 3: if ($\mathrm{RT}(G) > \mathrm{RT_{req}}(G) || R(G) < R_{\mathrm{req}}(G)$) then
 4:     return false;
 5: end if
 6: while ($P_{\mathrm{active}}$ is not null) do
 7:     for (each active processor $p_k \in P_{\mathrm{active}}$) do
 8:         Let $p_k$ be the sleep state;
 9:         Invoke the HEFT algorithm on $P_{\mathrm{active}} - \{p_k\}$ to
            obtain $\mathrm{RT}_k(G)$, $R_k(G)$, and $\mathrm{HC}_k(G)$;
10:         if $\left(\mathrm{RT}(G) \leqslant \mathrm{RT_{req}}(G) \&\& R_k(G) \leqslant R_{\mathrm{req}}(G)\right)$
            then
11:             Invoke the RE algorithm (Algorithm 3) to
                enhance the reliability value $R_k(G)$;
12:             if $\left(R_k(G) \geqslant R_{\mathrm{req}}(G)\right)$ then
13:                 Recalculate the hardware cost $\mathrm{HC}_k(G)$;
14:             end if
15:         end if
16:     end for
17:     if (no result satisfies $\mathrm{RT}_k(G) \leqslant \mathrm{RT_{req}}(G)$ &&
        $R_k(G) \geqslant R_{\mathrm{req}}(G)$) then
18:         return false;
19:     else
20:         Obtain the processor $p_{\min}$ using (12);
21:         $P_{\mathrm{active}} \leftarrow (P_{\mathrm{active}} - \{p_{\min}\})$;
22:         $\mathrm{RT}(G) \leftarrow \mathrm{RT_{min}}(G)$, $R(G) \leftarrow R_{\mathrm{min}}(G)$, and
            $\mathrm{HC}(G) \leftarrow \mathrm{HC_{min}}(G)$.
23:     end if
24: end while
```

processors. In this subsection, the enhanced algorithm called EEHCO is described in Algorithm 2.

Compared to EHCO, EEHCO's improvement is that it introduces the statements of lines 10–15, as shown in Algorithm 2.

1) EEHCO invokes the RE algorithm (see Algorithm 3) to enhance the reliability value $R_k(G)$ after the HEFT algorithm is invoked if the HEFT-generated reliability value $R_k(G)$ is less than the RR $R_{\mathrm{req}}(G)$ (line 11).
2) EEHCO recalculates the hardware cost $R_k(G)$ after the RE algorithm is invoked if the RE-generated reliability value $R_k(G)$ is larger than or equal to the RR $R_{\mathrm{req}}(G)$.

The time complexity of the EEHCO algorithm is the same as that of the EHCO algorithm (i.e., $O(|N|^2 \times |P|^3)$). The details are analyzed as follows.

1) The maximum number of removed processors is $|P|$ by invoking the while loop, such that it needs $O(|P|)$ time (lines 6–24).
2) Traversing all processors can be done in $O(|P|)$ time (lines 7–16).
3) Invoking the RE and HEFT algorithms can be done in $O(|N|^2 \times |P|)$ time (lines 9 and 11) because they are not nested.

Therefore, the time complexity of the EEHCO algorithm is $O(|N|^2 \times |P|^3)$.

Algorithm 3: RE Algorithm.

```
Input: $P_{\mathrm{active}} - \{p_k\}$, $G$, given task assignments
Output: $R_k(G)$, $\mathrm{RT}_k(G)$
 1: Sort the tasks in a list task_list by ascending order of
    $\mathrm{rank_u}$ values.
 2: while (there are tasks in task_list) do
 3:     $n_i \leftarrow$ task_list.out();
 4:     for (each processor $p_v \in (P_{\mathrm{active}} - \{p_{\min}\})$) do
 5:         Calculate $\mathrm{EST}(n_i, p_v)$ using (14);
 6:         Calculate $\mathrm{LFT}(n_i, p_v)$ using (15);
 7:         if ($\mathrm{LFT}(n_i, p_v) - \mathrm{EST}(n_i, p_v) < w_{i,v}$) then
 8:             continue;
 9:         end if
10:         Calculate $R(n_i, p_v)$ using (1);
11:     end for
12:     Select the processor $p_{pr(i)}$ with the MaxRe value
        for $n_i$;
13:     $\mathrm{AFT}(n_i) \leftarrow \mathrm{RT_{req}}(n_i, p_{\mathrm{pr}(i)})$;
14:     $\mathrm{AST}(n_i) \leftarrow \left(\mathrm{RT_{req}}(n_i, p_{\mathrm{pr}(i)}) - w_{i,\mathrm{pr}(i)}\right)$;
15: end while
16: Calculate the reliability $R(G)$ using (2);
```

B. RE Algorithm

The concrete RE algorithm is described in Algorithm 3.

The main idea of RE is that it migrates each task from the originally assigned processor to the possible new processor that can enhance the application's reliability as long as such migration does not violate the precedence constraints among tasks and real-time requirement of the application. The reason for this migration is that there are slacks on processors that can accommodate tasks.

The time complexity of the RE algorithm is analyzed as follows.

1) Assigning all tasks by traversing all tasks needs $O(|N|)$ time (lines 2–15).
2) Traversing all processors can be done in $O(|P|)$ time (lines 4–11).
3) Calculating the EST and latest finish time (LFT) of each should traverse all its precedence and successor tasks and all processors and be done in $O(|N| \times |P|)$ time (lines 5 and 6).

Therefore, the time complexity of the RE algorithm is $O(|N|^2 \times |P|)$, which is equal to that of the HEFT algorithm. RE also implements low-time complexity reliability enhancing.

EEHCO invokes the RE algorithm to enhance the reliability and does not sacrifice much computational efficiency due to the following reasons.

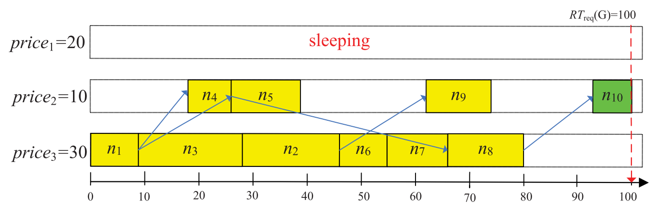

Fig. 7. Reassignment of the task n_{10} of the motivating application.

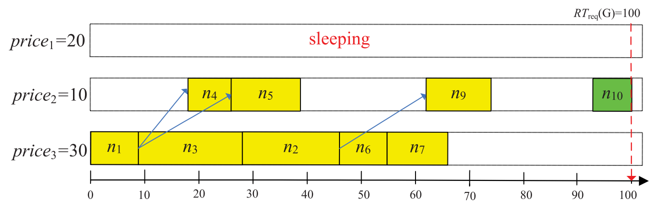

Fig. 8. Remove n_8 from Fig. 7 of the motivating application.

1) EEHCO has invoked the HEFT algorithm in line 9 (see Algorithm 2), and HEFT has the same time complexity as RE.
2) HEFT and RE are not nested in EEHCO, such that adding RE into EEHCO does not increase the time complexity of EEHCO.
3) RE could skip some processors that tasks cannot be inserted into (lines 7–9), such that it has higher computational efficiency than HEFT.

In contrast to HEFT, where the tasks are arranged in the descending order of rank_u values (i.e., from entry to exit tasks by downward optimization), this section sorts the tasks in the ascending order of rank_u values (i.e., from exit to entry tasks by upward optimization). The reason is that slacks exist between the $\text{RT}(G)$ and $\text{RT}_\text{req}(G)$, and the exit task with the minimum rank_u can first change its AFT to $\text{RT}_\text{req}(G)$, followed by the remaining tasks.

Considering that the example of p_1 is in the sleep state, the slacks in Fig. 5 are between $\text{RT}(G) = 98$ and $\text{RT}_\text{req}(G) = 100$. n_{10} will be optimized first, followed by n_8, n_7, n_9, n_6, n_5, n_2, n_4, n_3, and n_1. In the following discussion, we explain how the real-time requirements of tasks are calculated.

C. Real-Time Requirements of Tasks

We determine the real-time requirements of tasks in this subsection. We assume that n_{10} has been reassigned in p_2 (denoted with green color) using RE shown in Fig. 7, where we have $\text{AST}(n_{10}) = 93$ and $\text{AFT}(n_{10}) = 100$. We then consider the second task to be reassigned n_8. In the following discussion, n_8 is used as the example to explain the process.

We first removed the current task n_8 from Fig. 7, shown in Fig. 8. We then reassign n_8 into Fig. 8 with a higher reliability value without violating its real-time requirement. n_8's real-time requirement is calculated as follows.

1) Considering that n_8's successor task n_{10} has been reassigned by RE and cannot be changed, the LFT of the current task n_i is restricted by its successor tasks because of the precedence constraints among them. LFT is calculated as follows:

$$\begin{cases} \text{LFT}(n_\text{exit}, p_v) = \text{RT}_\text{req}(G) \\ \text{LFT}(n_i, p_v) = \min\limits_{n_j \in \text{succ}(n_i)} \left\{ \text{AST}(n_j) - c'_{i,j} \right\}. \end{cases}$$

2) Considering that n_8's predecessor tasks n_2, n_4, and n_6 have been assigned by the HEFT and cannot be changed, the EST of the current task n_i is also restricted by its predecessor tasks owing to the precedence constraints among them. The EST is calculated as follows:

$$\begin{cases} \text{EST}(n_\text{entry}, p_v) = 0 \\ \text{EST}(n_i, p_v) = \max\limits_{n_h \in \text{pred}(n_i)} \left\{ \text{AFT}(n_j) - c'_{h,i} \right\}. \end{cases}$$

For example, the ESTs and LFTs of n_8 on the processors $(P_\text{active} - \{p_\text{min}\})$ in Fig. 8 can be obtained as

$$\begin{cases} \text{EST}(n_8, u_2) = 70 \\ \text{EST}(n_8, u_3) = 55 \end{cases} \quad \begin{cases} \text{LFT}(n_8, u_2) = 93 \\ \text{LFT}(n_8, u_3) = 82. \end{cases}$$

3) Even when the EST and the LFT have been obtained on each available processor for the current task, task reassignment must be further constrained because the processors in $P_\text{active} - \{p_k\}$ are not always available because other tasks have already taken parts of the processors and only some slacks remain in the processors, as shown in Fig. 8. Therefore, task reassignment is actually task insertion. The slack set on processor p_v for n_i is defined as follows:

$$S_{i,v} = \{S_{i,v,1}, S_{i,v,2}, \ldots\}$$

where $S_{i,v,1}$ represents the first slack on p_v for n_i. Each slack has a start time (ST) and end time (ET). The vth slack $S_{i,v,q}$ is defined as

$$S_{i,v,q} = [t_\text{s}(S_{i,v,q}), t_\text{e}(S_{i,v,q})]$$

where $t_\text{s}(S_{i,v,q})$ and $t_\text{e}(S_{i,v,q})$ represent corresponding ST and ET of n_i on p_v, respectively. For example, when reassigning the task n_8 in Fig. 8, the slacks on p_2 and p_3 for n_8 are

$$\begin{cases} S_{8,2} = \{[0, 18], [39, 62], [74, 93]\} \\ S_{8,3} = \{[66, 100]\}. \end{cases} \tag{13}$$

4) To avoid violating the precedence constraints among tasks, the current task n_i should be assigned to the slacks that satisfy the new EST and LFT constraints as follows:

$$\text{EST}(n_i, p_v) = \max\left\{\text{EST}(n_i, p_v), t_\text{s}(S_{i,v,t})\right\} \tag{14}$$

and

$$\text{LFT}(n_i, p_v) = \min\left\{\text{LFT}(n_i, p_v), t_\text{e}(S_{i,v,t})\right\}. \tag{15}$$

For example, the new EST and LFT values of n_8 on all processors shown in Fig. 8 are as follows:

$$\begin{cases} \text{EST}(n_8, u_2) = 74 \\ \text{EST}(n_8, u_3) = 66 \end{cases} \quad \begin{cases} \text{LFT}(n_8, u_2) = 93 \\ \text{LFT}(n_8, u_3) = 82. \end{cases}$$

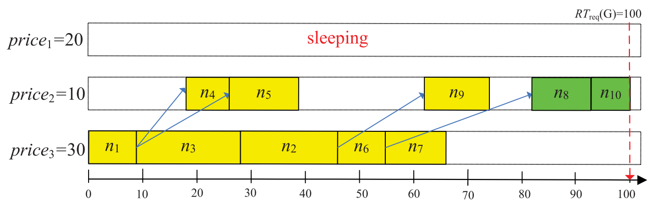

Fig. 9. n_8 is inserted into u_2 of Fig. 8 of the motivating application.

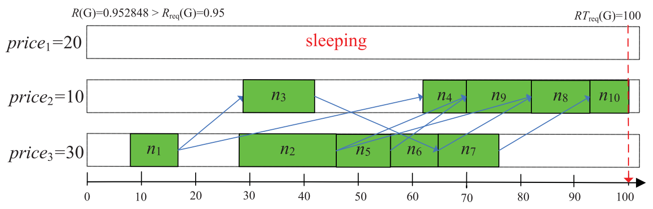

Fig. 10. RE-generated motivating application's task mapping of the on p_2 and p_3 when p_1 is in the sleep state.

5) Considering that the ESTs and LFTs for n_i have been obtained, we can decide which processor can accept n_i's insertion by judging whether (16) is found:

$$\mathrm{LFT}(n_i, p_v) - \mathrm{EST}(n_i, p_v) \geqslant w_{i,v}. \tag{16}$$

For example, n_8 can be inserted into u_1 and u_2 because

$$\begin{cases} \mathrm{LFT}(n_6, p_2) - \mathrm{AST}(n_8, p_2) = 93 - 74 = 19 \\ \quad \geqslant w_{8,2} = 11 \\ \mathrm{LFT}(n_8, p_3) - \mathrm{AST}(n_8, p_3) = 82 - 66 = 16 \\ \quad \geqslant w_{8,3} = 14. \end{cases}$$

D. REs of Tasks

After the functional safely requirements of the current task have been obtained in the previous subsections, the current task can be inserted into the processors. The strategy of enhancing reliability is as follows: we simply reassign the current task n_i to the insertable processor with the MaxRe value while stratifying the precedence constraints among tasks. That is, the assigned processor $p_{\mathrm{pr}(i)}$ is determined by

$$R(n_i, p_{\mathrm{pr}(i)}) = \max_{p_v \in (P_{\mathrm{active}} - \{p_k\}), \mathrm{LFT}(n_i, p_v) - \mathrm{EST}(n_i, p_v) \geqslant w_{i,v}} \times \{R(n_i, p_v)\}. \tag{17}$$

We assign n_i to $p_{\mathrm{pr}(i)}$ according to n_i's AFT and AST, which are calculated by

$$\mathrm{AFT}(n_i) = \mathrm{LFT}(n_i, p_{\mathrm{pr}(i)}) \tag{18}$$

and

$$\mathrm{AST}(n_i) = \mathrm{AFT}(n_i, p_{\mathrm{pr}(i)}) - w_{i,\mathrm{pr}(i)} \tag{19}$$

respectively. For instance, n_8 is inserted into p_2 of Fig. 8 because it has the MaxRe value of 0.994515, which is larger than 0.994416, where n_8 is inserted into p_3 (see Fig. 9). The AST and AFT calculated by (19) and (18) are 82 and 93, respectively.

The reliability values of remaining tasks $n_7, n_9, n_6, n_5, n_2, n_4, n_3$, and n_1 are then enhanced using the RE algorithm. Finally, RE-generated motivating application's task mapping on p_2 and p_3 is shown in Fig 10, where application's reliability value is enhanced to 0.952848.

E. Example Results Using EEHCO

Table VI shows the EEHCO-generated schedule results when each processor is in the sleep state in the first while loop. It can be seen that p_1's sleep satisfies not only the real-time requirement of 100, but also the RR of 0.05, whereas p_2's or p_3's sleep merely satisfies the real-time requirement of 100 and cannot satisfy the RR of 0.95. Therefore, the application's functional safety requirement can still be satisfied after removing p_1.

TABLE VI
EEHCO-Generated Schedule Results When Each Processor Is in the Sleep State in the First While Loop

Sleeping processor	Response time RT(G)	Reliability value $R(G)$	Satisfying functional safety requirement?	Hardware cost HC(G)
p_1	92	0.952848	Yes	40
p_2	96	0.940165	No	–
p_3	82	0.946816	No	–

TABLE VII
EEHCO-Generated Schedule Results When Each Processor Is in the Sleep State in the Second While Loop

Sleeping processor	Response time RT(G)	Reliability value $R(G)$	Satisfying functional safety requirement?	Hardware cost HC(G)
p_2	143	0.944405	No	–
p_3	130	0.937067	No	–

After the first while loop, Table VII shows the EEHCO-generated schedule results when each processor is in the sleep state in the second while loop. It can be seen that either p_2's or p_3's sleep cannot satisfy the real-time requirement of 100 and the RR of 0.95. Therefore, merely p_1's sleep in the first while loop is valid, and the task mapping is shown in Fig. 10.

F. SEEHCO Algorithm

Although EEHCO can sufficiently remove the reasonable processors to reduce hardware cost, it has high time complexity and thereby requires large computation effort for large-scale parallel application. For this reason, a simplified algorithm called SEEHCO is further proposed in this study. We elaborate the description of the SEEHCO algorithm in Algorithm 4.

SEEHCO's simplification is that it determines the processor removing order by attempting to remove each active processor in lines 5–15, which is similar to EEHCO. However, SEEHCO no longer redetermines the processor removing order in the while loop (lines 16–29), but it just directly selects the order

Algorithm 4: SEEHCO Algorithm.

```
Input: $P = \{p_1, p_2, ..., p_{|P|}\}$, $G$, and $\text{RT}_{\text{req}}(G)$, $R_{\text{req}}(G)$
Output: $\text{RT}(G)$, $R(G)$, and $\text{HC}(G)$
 1: Invoke the HEFT algorithm on $P_{\text{active}}$ to obtain the
    initial $\text{RT}(G) \leftarrow \text{LB}(G)$, $R(G)$, and $\text{HC}(G)$;
 2: if ($\text{RT}(G) > \text{RT}_{\text{req}}(G) || R(G) < R_{\text{req}}(G)$) then
 3:     return false;
 4: end if
 5: for (each processor $p_k \in P$) do
 6:     Let $p_k$ be the sleep state;
 7:     Invoke the HEFT algorithm on $P - \{p_k\}$ to obtain
        $\text{RT}_k(G)$, $R_k(G)$, and $\text{HC}_k(G)$;
 8:     if $\left(\text{RT}(G) \leqslant \text{RT}_{\text{req}}(G) \&\& R_k(G) \leqslant R_{\text{req}}(G)\right)$ then
 9:         Invoke the RE algorithm (Algorithm 3) to enhance
    the
            reliability value $R_k(G)$;
10:         if $\left(R_k(G) \geqslant R_{\text{req}}(G)\right)$ then
11:             Recalculate the hardware cost $\text{HC}_k(G)$;
12:         end if
13:     end if
14: end for
15: Put all the processors into $P_{\text{active}}$ according to an
    ascending of $\text{HC}_k(G)$;
16: while ($P_{\text{active}}$ is not null) do
17:     Let $p_k$ be the sleep state;
18:     Invoke the HEFT algorithm on $P_{\text{active}} - \{p_k\}$ to
        obtain $\text{RT}_k(G)$, $R_k(G)$, and $\text{HC}_k(G)$;
19:     if $\left(\text{RT}(G) \leqslant \text{RT}_{\text{req}}(G) \&\& R_k(G) \leqslant R_{\text{req}}(G)\right)$ then
20:         Invoke the RE algorithm (Algorithm 3) to enhance
            the reliability value $R_k(G)$;
21:         if $\left(R_k(G) \geqslant R_{\text{req}}(G)\right)$ then
22:             Recalculate the hardware cost $\text{HC}_k(G)$;
23:             $P_{\text{active}} \leftarrow (P_{\text{active}} - \{p_k\})$;
24:             $\text{RT}(G) \leftarrow \text{RT}_k(G)$, $R(G) \leftarrow R_k(G)$, and
                $\text{HC}(G) \leftarrow \text{HC}_k(G)$.
25:         else
26:             return false;
27:         end if
28:     end if
29: end while
```

determined in lines 5–15. By this treatment, SEEHCO's time complexity is reduced to $O(|N|^2 \times |P|^2)$ (analyzed in detail in the following), but such a simplification may result in larger hardware cost than the EEHCO algorithm. Given that the motivating example only has one valid while loop to select removed processor, the final hardware cost using the SEEHCO algorithm is also 129.6059 (see Fig. 10).

The time complexity of the SEEHCO algorithm is analyzed in detail as follows.

1) The maximum number of removed processors is $|P|$ by invoking the while loop, such that it needs $O(|P|)$ time (lines 16–29).
2) Traversing all processors can be done in $O(|P|)$ time (lines 5–14).
3) Invoking the RE and HEFT algorithms can be done in $O(|N|^2 \times |P|)$ time (lines 9 and 11) because they are not nested.

Considering that lines 5–14 and lines 16–29 are not nested, the time complexity of the EEHCO algorithm is $O(|N|^2 \times |P|^2)$.

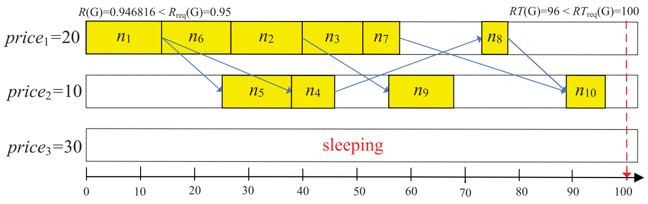

Fig. 11. Task mapping on p_1 and p_2 with the MaxRe when p_3 is in a sleep state.

G. Optimal Solutions of the Motivating Application

Considering that both EHCO and EEHCO are greedy algorithms and cannot guarantee an optimal solution, it is better to provide optimality of the small-scale motivating application for the proposed heuristic algorithms.

1) Removing two processors is infeasible because the sum of the WCETs of all tasks on each processor in Table II is larger than the real-time requirement of 100.
2) Considering that p_3 has the maximum hardware cost of 30, if we can find a feasible task assignment for the motivating application (i.e., $R(G) \geqslant 0.95$ and $\text{RT}(G) \leqslant 100$) by removing p_3, then we can obtain an optimal hardware cost minimization solution.
3) For the motivating application, we calculate that there are at most 40 320 task priority orders, but only 1680 task priority orders are valid by excluding orders that do not meet the precedence constraints among tasks. n_1 and n_{10} are the highest and lowest task priorities in all the orders. There are 2^{10} task assignment schemes because each task may be assigned to p_1 or p_2. Therefore, there are 1 720 320 (i.e., 1680×2^{10}) combinations to determine the optimal salutation. However, we traverse all the 1 720 320 combinations and find that the MaxRe is 0.946816, which cannot satisfy the RR of 0.95. Fig. 11 shows the task mapping on p_1 and p_2 with the MaxRe when p_3 is in a sleep state, where the task priority order is $\{n_1, n_6, n_2, n_5, n_4, n_9, n_3, n_7, n_8, n_{10}\}$ and the response time of the application is 96. Therefore, the minimum hardware cost minimization of 30 is unfeasible, and it must be increased to 40, which has been obtained by EEHCO and SEEHCO.

VI. Experiments

The performance metrics for comparison are the actual response time value $\text{RT}(G)$, the actual reliability value $R(G)$, and the total hardware cost $\text{HC}(G)$ of application. The compared algorithm with the proposed EHCO, EEHCO, and SEEHCO is the IHCO algorithm, which is involved from [22]. Processor and application parameters taken from [23] and [24] are as follows: 10 ms $\leqslant w_{i,k} \leqslant$ 100 ms, 10 ms $\leqslant c_{i,j} \leqslant$ 100 ms, and 0.000001/ms $\leqslant \lambda_k \leqslant$ 0.000009/ms. The embedded processor's

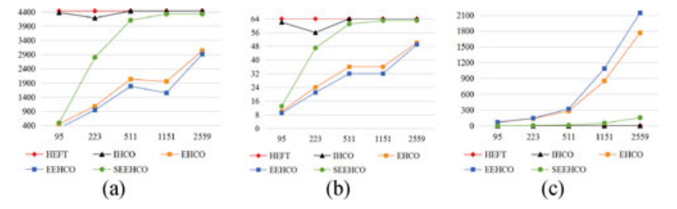

Fig. 12. Results of fast Fourier transform applications for varying task numbers. (a) Hardware cost (unit: \$). (b) Active processor numbers. (c) Computation time (unit: s).

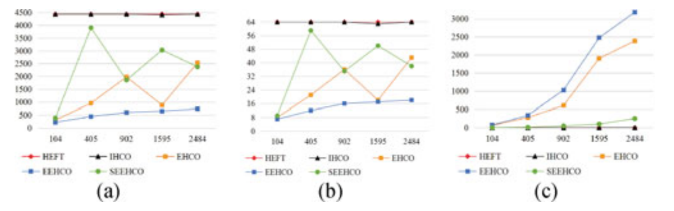

Fig. 13. Results of Gaussian eliminations for varying task numbers. (a) Hardware cost (unit: \$). (b) Active processor numbers. (c) Computation time (unit: s).

prices are in the scope of \$25 $\leqslant \text{price}_k \leqslant$ \$110 [6], and the total hardware cost of the system is \$4439. Considering that this study focuses on the design phase, the parallel applications are executed on a simulated heterogeneous embedded system based on the above processor and application parameter values. The simulated system is configured with 64 processors implemented by Java on a standard desktop computer (2.6-GHz Intel CPU and 4 GB memory). We use fast Fourier transform and Gaussian elimination as experimental objects because they are two typical parallel applications with high and low parallelism, respectively. The details of these two applications can refer to [1].

Experiment 1: This experiment is conducted to compare hardware costs, active processor numbers, and computation time values of fast Fourier transform applications for varying numbers of tasks. We let the real-time requirement and the RR be $\text{RT}_{\text{req}}(G) = \text{LB}(G)$ and $R_{\text{req}}(G) = R_{\text{heft}}(G)$, respectively. $R_{\text{heft}}(G)$ represents the HEFT-generated reliability value. Therefore, all the HEFT, IHCO, EHCO, EEHCO, and SEEHCO algorithms can satisfy given functional safety requirements because HEFT-generated values are the initial values of all the algorithms. For this reason, we no longer provide the actual response time and reliability values and directly observe the hardware costs, active processor numbers, and computation time values of applications. The task numbers are changed from 95 to 2559.

1) The results in Fig. 12(a) show that EEHCO generates the minimum hardware costs followed by EHCO, SEEHCO, IHCO, and HEFT in all the cases. We can see that the HEFT always generates the maximum hardware cost of \$4439 because it occupies all the 64 processors. The reduced hardware costs using IHCO are limited, because its hardware costs are from \$4384 to \$4439 and are very close to \$4439. In general, the hardware costs increase with the increment of task numbers using EHCO, EEHCO, and SEEHCO. The results also show that EEHCO is an enhanced version of EHCO, and it outperforms EHCO by 12.4–32%, that is, using the RE algorithm to enhance the reliability is effective. SEEHCO can reduce hardware costs compared with IHCO. However, SEEHCO's effect is gradually reduced with the increase in the application's scales because it merely explores one loop to select removed processors.
2) Fig. 12(b) shows the active processor numbers using all the algorithms. We find that Fig. 12(b) shows surprising consistency with the curve changes in Fig. 12(a). The reason is that consumed hardware costs of applications are very relevant to the numbers of processes, even if the prices of processors are different.
3) Fig. 12(c) shows the computation time values using all the algorithms. As we expect, both EHCO and EEHCO are time consuming, especially on large-scale applications. For example, when the task number is 2559, the computation time values reach 30 and 36 min for EHCO and EEHCO, respectively. For this reason, SEEHCO can reduce the computation time to 3 s to 2.5 min by reducing the loop number of EEHCO. However, the negative effect of SEEHCO is that the cost reduction ratio is also significantly reduced.

Experiment 2: The application scales, parameter values, and objective of this experiment are approximately the same as Experiment 1 for Gaussian elimination applications. We still let the real-time requirement and the RR be $\text{RT}_{\text{req}}(G) = \text{LB}(G)$ and $R_{\text{req}}(G) = R_{\text{heft}}(G)$, respectively. The task numbers are changed from 104 to 2484.

1) Compared with Fig. 12(a), the main difference in Fig. 13(a) is that EEHCO is more effective than EHCO in terms of reducing the hardware cost. In addition, SEEHCO also shows good performance, and generated hardware costs are less than those of EHCO when the task numbers are 902 and 2484.
2) Similar to Experiment 1, Fig. 13(b) also shows consistency with the curve changes in Fig. 13(a). The results further indicate that the consumed hardware costs of the application are directly relevant to the numbers of processes.
3) The computation time values using all the algorithms in Fig. 13(c) are also similar to those of Fig. 12(c). That is, EHCO and EEHCO are time consuming, whereas SEEHCO is fast, but it has certain negative effect in hardware cost reduction.

Experiment 3: Besides aforementioned real applications, we also consider randomly generated parallel applications by the task graph generator, which provides convenient configuration for application's parameters [25]. The parameters are set as follows: the communication-to-computation ratio is 1, the shape parameter is 1, and the heterogeneity factor is 0.5. The shape parameter means the density of the graph values are in the 0.35–$\sqrt{|N|/3}$ scope in the task graph generator, where 0.35 and $\sqrt{|N|/3}$ are the lowest and highest parallelism factors, respectively. As we set the shape parameter as 1 in the experiment, the randomly generated applications can be considered as high-

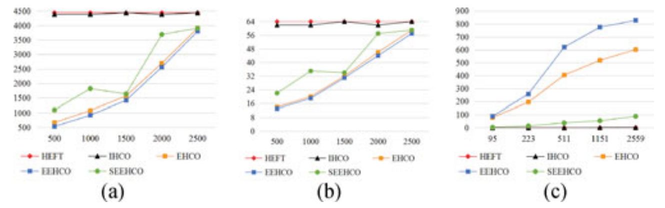

Fig. 14. Results of randomly generated applications for varying task numbers. (a) Hardware cost (unit: \$). (b) Active processor numbers. (c) Computation time (unit: s).

parallelism applications. The heterogeneity factor values are in the 0–1 scope in the task graph generator, where 0.1 and 1 are the lowest and highest heterogeneity factors, respectively. We still let the real-time requirement and the RR be $\mathrm{RT}_{\text{req}}(G) = \mathrm{LB}(G)$ and $R_{\text{req}}(G) = R_{\text{heft}}(G)$, respectively. The task numbers are changed from 500 to 2500. In general, the results in Fig. 14 (a)–(c) show the same rule patterns as Experiments 1 and 2. Due to space limitation, we no longer discuss similar details in this experiment. In a word, experimental results with real parallel or random generated applications reveal that EEHCO can obtain minimum hardware cost, whereas SEEHCO is efficient for large-scale parallel applications compared with the existing algorithms.

From the experiments, we have the following observations. For fast Fourier transform application and randomly generated applications, EEHCO has very similar results compared to EHCO, as shown in Figs. 12 and 14. But for Gaussian elimination application, obvious difference can be observed, as shown in Fig. 13. We know that the fast Fourier transform application and randomly generated applications in this study are high-parallelism applications, whereas Gaussian elimination application is low-parallelism application. A low-parallelism application has longer response time than a high-parallelism application in the approximately equal scale. That is, the tasks in low-parallelism application are not evenly assigned to the processor. This nonuniform assignment allows the RE algorithm to be fully employed. Therefore, EEHCO has obvious difference compared to EHCO for low-parallelism Gaussian elimination application.

Experiment 4: To calculate how many processors can be removed and how much cost can be saved in a real industrial example, we use the real automotive application shown in Fig. 15 adopted from [1] and [16] to show the results. This application consists of six application blocks: engine controller with seven tasks (n_1–n_7), automatic gear box with four tasks (n_8–n_{11}), antilocking brake system with six tasks (n_{12}–n_{17}), wheel angle sensor with two tasks (n_{18}–n_{19}), suspension controller with five tasks (n_{20}–n_{24}), and body work with seven tasks (n_{25}–n_{31}). Processor and application parameters take from [1] are as follows: $100\,\mu$ s$\leqslant w_{i,k} \leqslant 400\,\mu$ s, $100\,\mu$ s $\leqslant c_{i,j} \leqslant 400\,\mu$ s, $0.000001/\mu$ s $\leqslant \lambda_k \leqslant 0.000009/\mu$ s. The application is executed on 16 electronic control units (ECUs), and its real-time requirement is 1000 μ s based on the hazard analysis and risk assessment.

We adopt reliability-related values from the automotive functional safety standard ISO 26262. ISO 26262 provides the duration/probability of exposure in [9, Table B.2, Annex B of

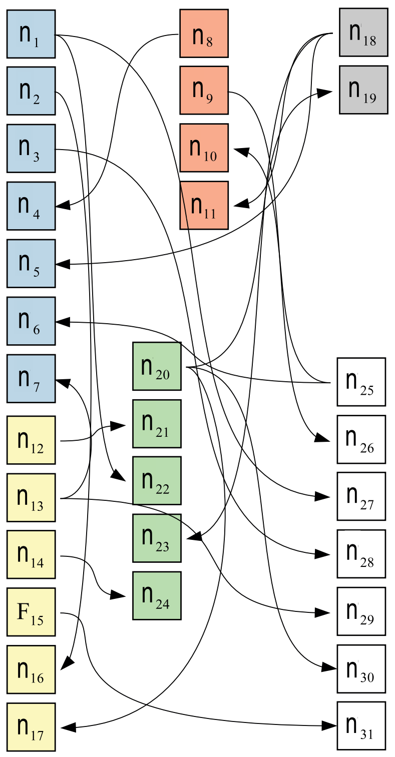

Fig. 15. Real automotive application [1], [16].

TABLE VIII
CLASSES OF PROBABILITY OF EXPOSURE REGARDING DURATION/PROBABILITY OF EXPOSURE IN ISO 26262 [9]

Exposure level		Probability of exposure	Reliability requirement
E1	Very low probability	Not specified	At least exceeds 0.99
E2	Low probability	$< 1\%$	0.99
E3	Medium probability	[1%, 10%]	> 0.9
E4	High probability	$> 10\%$	$<=0.9$

Part 3], as shown in Table VIII. For example, the probability of exposure E4 is larger than 10% of average operating time. ISO 26262 does not define the concept of RR, but we can deduce the corresponding RRs for given exposure levels. For example, the probability of exposure E2 is less than 1% of average operating time, that is, the lowest probability of a occurrence hazardous event is close to 0.01; to ensure safety, the actual reliability must be larger than or equal to $1 - 0.01 = 0.99$, which is considered as the RR in this case. The RRs for other exposures can also be obtained according to the above rule. Finally, the RRs for exposures are shown in Table VIII.

In this experiment, the RR is changed from 0.9 to 0.99 with a 0.01 increment because values of RRs fall in the range of exposures E3 and E2 (see Table VIII). Meanwhile, the MaxRe requirement for the application is set as 0.999999, which belongs to the RR in E1. The total hardware cost of the system including 32 ECUs is \$1097.

Tables IX and X show the removed ECU numbers and saved hardware costs, respectively, of the real automotive application for varying RRs.

TABLE IX
REMOVED ECU NUMBERS OF THE REAL AUTOMOTIVE APPLICATION FOR VARYING RRS

$R_{\text{req}}(G)$	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
IHCO	7	7	7	7	7	7	7	7	0	0
EHCO	10	10	10	10	10	10	10	10	0	0
EEHCO	10	10	10	10	10	10	10	10	8	0
SEEHCO	8	8	8	8	8	8	8	8	5	0

TABLE X
SAVED HARDWARE COSTS (UNIT: $) OF THE REAL AUTOMOTIVE APPLICATION FOR VARYING RRS

$R_{\text{req}}(G)$	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
IHCO	325	325	325	325	325	325	325	325	0	0
EHCO	869	869	869	869	869	869	869	869	0	0
EEHCO	869	869	869	869	869	869	869	869	709	0
SEEHCO	750	750	750	750	750	750	750	750	500	0

1) Table IX shows that IHCO, EHCO, EEHCO, and SEEHCO can remove seven, ten, ten, and eight ECUs, respectively; Table X shows that IHCO, EHCO, EEHCO, and SEEHCO can save the hardware costs of \$325, \$869, \$869, and \$750, respectively, when the RR is from 0.9 to 0.97. The removed ECU numbers do not increase with the increment of RRs from 0.9 to 0.97. The results also indicate that IHCO can also remove redundant ECUs and save hardware costs, that is, IHCO is feasible when the RR is not very high. In the feasible range of IHCO, we can calculate that EHCO, EEHCO, SEEHCO can, respectively, improve 62.5%, 62.5%, and 56.7% hardware costs compared with IHCO, as shown in Table X.
2) When the RR reaches 0.98, the removed ECU numbers using IHCO and EHCO drastically reduce to 0, whereas EEHCO can still remove eight ECUs and save the hardware cost of \$709 by including the RE algorithm to enhance the reliability, and SEEHCO can also remove five ECUs and save the hardware cost of \$500 by simplifying the EEHCO algorithm. The results indicate that IHCO has a shortcoming, and it may be infeasible when the RR is high.
3) When the RR reaches 0.99, all the algorithms cannot remove any ECU and save hardware cost, as shown in Tables IX and X. The results reflect that we need to consume a considerable cost to satisfy its functional safety requirement when the RR is very high.

VII. CONCLUSION

In this study, three hardware cost design optimization algorithms EHCO, EEHCO, and SEEHCO were proposed for a functional safety-critical DAG-based parallel application on heterogeneous distributed embedded systems. EHCO iteratively removes the processors, without which the minimum hardware costs can be generated while satisfying application's functional safety requirement. EEHCO improves the possibility that the functional safety requirement is satisfied by invoking the RE algorithm to enhance application's reliability value without violating the precedence constraint among tasks and real-time requirements of the application. SEEHCO adapts large-scale parallel application's hardware cost design optimization by simplifying EEHCO's loop number. Experiments also show the effectiveness of the proposed algorithms.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the associate editor and three anonymous reviewers for their constructive comments which have helped to improve the quality of the paper.

REFERENCES

[1] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, and K. Li, "Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1629–1640, Aug. 2017.
[2] M. Hu, J. Luo, Y. Wang, M. Lukasiewycz, and Z. Zeng, "Holistic scheduling of real-time applications in time-triggered in-vehicle networks," *IEEE Trans. Ind. Informat.*, vol. 10, no. 3, pp. 1817–1828, Aug. 2014.
[3] G. Xie, J. Jiang, Y. Liu, R. Li, and K. Li, "Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1068–1078, Jun. 2017.
[4] J. Hascoet, J.-F. Nezan, A. Ensor, and B. D. de Dinechin, "Implementation of a fast fourier transform algorithm onto a manycore processor," in *Proc. Conf. Design Archit. Signal Image Process.*, 2015, pp. 1–7.
[5] T. Mladenov, S. Nooshabadi, and K. Kim, "Implementation and evaluation of raptor codes on embedded systems," *IEEE Trans. Comput.*, vol. 60, no. 12, pp. 1678–1691, Dec. 2011.
[6] 2017. [Online]. Available: http://copperhilltech.com/embedded-can-interfaces/
[7] Z. Gu, G. Han, H. Zeng, and Q. Zhao, "Security-aware mapping and scheduling with hardware co-processors for flexray-based distributed embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 3044–3057, Oct. 2016.
[8] Z. E. Bhatti, P. S. Roop, and R. Sinha, "Unified functional safety assessment of industrial automation systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 1, pp. 17–26, Feb. 2017.
[9] *Road Vehicles-Functional Safety*, ISO 26262, 2011.
[10] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1234–1249, Aug. 2013.
[11] V. N. Dubinin and V. Vyatkin, "Semantics-robust design patterns for IEC 61499," *IEEE Trans. Ind. Informat.*, vol. 8, no. 2, pp. 279–290, May 2012.
[12] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 241–254, Oct.–Dec. 2009.
[13] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, and K. Li, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2017.2665552.
[14] L. Zhao, Y. Ren, Y. Xiang, and K. Sakurai, "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems," in *Proc. 12th IEEE Int. Conf. High Perform. Comput. Commun.*, 2010, pp. 434–441.
[15] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Comput.*, vol. 39, no. 10, pp. 567–585, Jul. 2013.
[16] J. Gan, P. Pop, and J. Madsen, "Tradeoff analysis for dependable real-time embedded systems during the early design phases," Ph.D. dissertation, Dept. Informat. Math. Model., Tech. Univ. Denmark, Kongens Lyngby, Denmark, 2014.
[17] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, May 2015, Art. no. 50.

[18] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surveys*, vol. 43, no. 4, Oct. 2011, Art. no. 35.

[19] 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.

[20] J. D. Ullman, "Np-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.

[21] G. Xie, G. Zeng, Z. Li, R. Li, and K. Li, "Adaptive dynamic scheduling on multi-functional mixed-criticality automotive cyber-physical systems," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 6676–6692, Aug. 2017.

[22] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment," *J. Grid Comput.*, vol. 14, no. 1, pp. 55–74, Mar. 2016.

[23] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 316–328, Aug. 2010.

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

[25] T. graph generato. 2015. [Online]. Available: https://sourceforge.net/projects/taskgraphgen/

Guoqi Xie (M'15) received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2014.

He is currently an Associate Professor with Hunan University. He was a Postdoctoral Researcher with Nagoya University, Japan, from 2014 to 2015, and with Hunan University, from 2015 to 2017. His major research interests include embedded and cyber-physical systems, parallel and distributed systems, software engineering, and methodology. Dr. Xie has received the Best Paper Award from international symposium on parallel and distributed processing with applications 2016. He is a member of the Association for Computing Machinery and the China Computer Federation.

Yuekun Chen is currently working toward the Ph.D. degree with Hunan University, Changsha, China.

Her research interests include embedded systems design, distributed systems design, and software engineering.

Renfa Li (M'05–SM'10) received the Ph.D. degree in electronic engineering from Huazhong University of Science and Technology, Wuhan, China, in 2002.

He is a Professor of computer science and electronic engineering with Hunan University, Changsha, China. He is the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. He is also an expert committee member of National Supercomputing Center, Changsha. His major interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of things.

Dr. Li is a member of the council of the China Computer Federation and a senior member of the Association for Computing Machinery.

Keqin Li (M'90–SM'96–F'15) received the Ph.D. degree in computer science from the University of Houston, Houston, TX, USA, in 1990.

He is a SUNY Distinguished Professor of computer science. He has authored or coauthored more than 500 journal articles, book chapters, and refereed conference papers. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things, and cyber-physical systems.

Dr. Li has received several best paper awards. He is serving or has served on the Editorial Boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing.