Resource Consumption Cost Minimization of Reliable Parallel Applications on Heterogeneous Embedded Systems

Guoqi Xie, *Member, IEEE*, Yuekun Chen, Yan Liu, Yehua Wei, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Heterogeneous processors are increasingly being used in embedded systems where parallel applications with precedence-constrained tasks widely exist. Reliability is an important functional safety requirement and reliability goal should be satisfied for safety-critical parallel applications; meanwhile, resource is limited in embedded systems and it should be minimized. This study solves the problem of resource consumption cost minimization of a reliable parallel application on heterogeneous embedded systems without using fault tolerance. The problem is decomposed into two subproblems, namely, satisfying reliability goal and minimizing resource consumption cost. The first subproblem is solved by transferring the reliability goal of the application to that of each task, and the second subproblem is solved by heuristically assigning each task to the processor with the minimum resource consumption cost while satisfying its reliability goal. Experiments with real parallel applications verify that the proposed algorithm obtains minimum resource consumption costs compared with the state-of-the-art algorithms.**

***Index Terms*—Heterogeneous embedded systems, parallel applications, reliability goal, resource consumption cost.**

Manuscript received July 25, 2016; revised November 5, 2016 and December 4, 2016; accepted December 15, 2016. Date of publication December 21, 2016; date of current version August 1, 2017. This work was supported in part by the National Key Research and Development Plan of China under Grant 2016YFB0200405 and Grant 2012AA01A301-01, in part by the Natural Science Foundation of China under Grant 61672217, Grant 61173036, Grant 61432005, Grant 61370095, Grant 61300037, Grant 61370097, Grant 61502405, Grant 61402170, and Grant 61502162, and in part by the China Postdoctoral Science Foundation under Grant 2016M592422. Paper no. TII-16-0750.R2. *(Corresponding author: Renfa Li.)*

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

Y. Wei is with the College of Physics and Information Science, Hunan Normal University, Changsha 410082, China (e-mail: yehuahn@163.com).

K. Li is with the Key Laboratory for Embedded and Network Computing of Hunan Province, 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.2016.2641473

I. Introduction

A. Background

Multiprocessors are increasingly used in the implementation of high-performance embedded systems, such as image recognition, automotive control, and human body interaction plus gesture control. To satisfy the higher functional and nonfunctional requirements, heterogeneous embedded systems have emerged [1]–[3]. Although the increasing processing capacity of processors has been developed to achieve high performance, the frequency of transient failures has increased dramatically, caused by electromagnetic interference, radiations, high temperature, etc. [4]. Reliability is defined as the probability the schedule is successful (i.e., succeeds to complete its execution) for an application [5]–[7]. Any application cannot be 100% reliable in practice. If an application satisfies its certificated reliability goal, then it is considered to be reliable in safety-critical embedded systems. Reliability goal has been used in some industrial functional safety standards, such as the ISO 26262 standard for automotive software systems, the DO-178C standard for avionics software, and the IEC 61508 standard for all kinds of industrial software systems [8]. If the reliability goal of the application cannot be satisfied, then disastrous consequences could happen. Therefore, reliability is an important functional safety property and reliability goal should be satisfied for a safety-critical embedded application.

B. Motivation

As multiprocessors continue to scale, increasingly distributed and parallel applications with precedence-constrained tasks widely exist in embedded systems [1], [9]. A parallel application with precedence-constrained tasks is represented by a directed acyclic graph (DAG), where the nodes represent the tasks and the edges represent the communication messages between the tasks [1], [9]. Scheduling tasks of a DAG-based parallel application for fastest execution is a well-known NP-hard optimization problem, and many heuristic list scheduling algorithms have been proposed to generate near-optimal solutions to the problem in heterogeneous systems [10]–[12]. In general, fault-tolerant techniques are the effective approaches to satisfy the reliability goal of a parallel application, and replication approaches

are important software fault-tolerant techniques. Considering the limited resources of embedded systems, fault-tolerant techniques may be not suitable [13]–[15]. Therefore, an effective task assignment strategy to satisfy the reliability goal without using replication should be presented.

Resource is often limited in embedded systems, and should be optimized during the design phase [16], [17]. Resource consumption cost minimization, modeling, analysis, and verification have been widely studied from embedded systems [16], [17] to cluster systems [7], [18]. Intuitively, larger reliability could cause a longer schedule length and consume more resources for a parallel application [13]–[15]. In safety-critical embedded systems, if the reliability goal of an application can be satisfied, then this application is reliable according to the aforementioned functional safety standards. The recent investigation presented the MaxRe [6] and the RR (i.e., least resources to meet the reliability requirement) [7] algorithms to minimize resource consumption while satisfying the reliability goal of a parallel application on heterogeneous cluster systems, respectively. Although the MaxRe and RR algorithms can be compatible in a nonfault-tolerant manner, they can be further improved in calculating reliability goal of each task and assigning each task to the processor (refer to Section IV-D for more details).

C. Main Contributions

This study aims to solve the problem of minimizing the resource consumption cost of reliable parallel applications on heterogeneous embedded systems without using fault tolerance, and the problem is decomposed into two subproblems, namely, satisfying reliability goal and minimizing resource consumption cost. The main contributions are summarized as follows.

1) The first subproblem is solved by presupposing that the unassigned tasks are assigned to the processor with the minimum reliability value, thereby transferring the reliability goal of the application to the reliability goal of each task.
2) The second subproblem is solved by heuristically scheduling each task with low time complexity, thereby assigning the task to the processor with the minimum resource consumption cost.
3) Experiments with real parallel applications verify that the proposed algorithm obtains minimum resource consumption costs compared with the state-of-the-art algorithms.

The rest of this paper is organized as follows. Section II reviews related studies. Section III builds related models. Section IV presents related preliminaries. Section V solves the presented problem. Section VI verifies the performance of the proposed algorithm. Section VII concludes this study.

II. Related Work

Considering that this study focuses on resource consumption cost and reliability of a DAG-based parallel application, this study mainly reviews the related research on cost and reliability of DAG-based parallel applications.

Resource optimization is one of the important topics in embedded systems. Qiu and Sha [16] solved the cost minimization while satisfying hard or soft timing constraints for heterogeneous embedded systems. Ovatman *et al.* [17] built the resource consumption cost model and analyzed and verified the resource consumption cost scenarios for embedded systems using priced timed automata. The widely accepted reliability model for an operation was presented in [19], where the failure rate of each hardware is characterized by a constant λ. Generally, the reliability of a parallel application is represented as the product of the reliability values of all the tasks [6], [7], [20], [21]. In general, fault-tolerant techniques are the effective ways to satisfy the reliability goal of a parallel application, and replication approaches are important software fault-tolerant techniques. Considering the limited resource of embedded systems, fault tolerance may not be suitable as pointed out earlier.

Intuitively, larger reliability could cause longer schedule length of a parallel application and the problem of optimizing schedule length and reliability is considered as a typical bicriteria optima or Pareto optima problem [13]–[15]. Doğan and Özgüner [13] investigated the bicriteria optima problem between reliability and scheduled length by merging the bicriteria of scheduled length and reliability into a single objective function for joint optimization of them. The method in [13] has better reliability but longer scheduled length than that in [14]. Dongarra *et al.* [15] implemented the bio-objective optimization of maximizing reliability and minimizing scheduled length. Tang *et al.* [22], [23] considered the reliability goal for a parallel application on heterogeneous systems without using fault tolerance; however, these studies do not aim to satisfy the reliability goal of the application. Zhao *et al.* [6], [7] presented the MaxRe and RR algorithms to minimize the resource consumption while satisfying the reliability goal of a parallel application in heterogeneous cluster systems by using fault tolerance. The main limitations of [6] and [7] have been summarized in Section I-B.

TABLE I
Important Notation in This Paper

Notation	Definition
$c_{i,j}$	WCTT between the tasks n_i and n_j
$w_{i,k}$	WCET of the task n_i on the processor u_k
$\overline{w_i}$	Average execution time of the task n_i
$\text{rank}_u(n_i)$	Upward rank value of the task n_i
$\|X\|$	Size of the set X
λ_k	Constant failure rate per time unit of the processor u_k
γ_k	Resource consumption cost rate per time unit of the processor u_k
γ_{comm}	communication cost rate γ_{comm} per time unit
$R(n_i, u_k)$	Reliability of the task n_i on the processor u_k
$R(G)$	Reliability of the application G
$\text{cost}(n_i, u_k)$	Resource consumption cost of the task n_i on the processor u_k
$\text{cost}(G)$	Resource consumption cost of the application G
$u_{\text{proc}(i)}$	Assigned processor of the task n_i
$R_{\text{min}}(G)$	Minimum reliability of the application G
$R_{\text{max}}(G)$	Maximum reliability of the application G
$R_{\text{goal}}(G)$	Reliability goal of the application G
$R_{\text{goal}}(n_i)$	Reliability goal of the task n_i

III. Models

Table I lists important notations and their definitions used in this study.

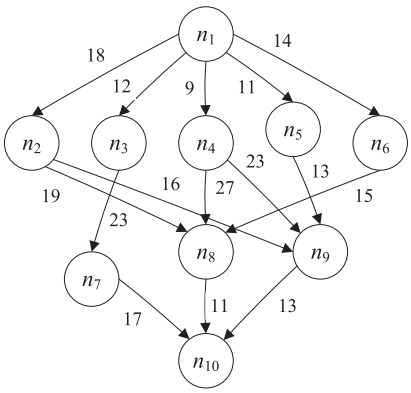


Fig. 1. Motivating example of a DAG-based parallel application [10]–[12].

TABLE II
WCETs OF TASKS ON DIFFERENT PROCESSORS OF THE MOTIVATING PARALLEL APPLICATION [10]–[12]

Task	u_1	u_2	u_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
CLASSES OF PROBABILITY OF EXPOSURE REGARDING DURATION/PROBABILITY OF EXPOSURE IN ISO 26262 [24]

	Exposure level	Probability of exposure	Reliability goal
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	$>10\%$	$<=0.99$

A. Application Model

This study considers a system platform with heterogeneous multiprocessors. As controller area network (CAN) networks themselves provide redundant links for high fault tolerance, similar to most studies [6], [14], [21], this study only considers the processor failure, and does not include the communication failure into the problem (i.e., the communication is assumed to be reliable in this study), but the communication time between processors should be considered because of the distributed characteristic of parallel applications and the system [1], [9].

Let $U = \{u_1, u_2, \ldots, u_{|U|}\}$ represent a set of heterogeneous processors, where $|U|$ represents the size of set U. Note that for any set X, this study uses $|X|$ to denote its size. A parallel application is represented by a DAG $G = (N, M, C, W)$ [9]–[12]. N represents a set of tasks in G, and each task $n_i \in N$ has different worst case execution times (WCETs) on different processors. 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 a worst case transmission time (WCTT) of $m_{i,j}$, if n_i and n_j are not assigned to the same processor [12]. pred(n_i) represents the set of the immediate predecessor tasks of n_i. succ(n_i) represents the set of the immediate successor tasks of n_i. The task that has no predecessor task is denoted by n_{entry}, and the task that has no successor task is denoted by n_{exit}. W is an $|N| \times |U|$ matrix, where $w_{i,k}$ denotes the WCET of n_i running on u_k. All tasks are scheduled with the global nonpreemptive policy.

Fig. 1 shows a motivating example of a DAG-based parallel application with tasks and messages. Table II is a matrix of WCETs in Fig. 1. The example shows ten tasks executed on three processors $\{u_1, u_2, u_3\}$. Weight 14 of n_1 and u_1 in Table II represents the WCET, denoted by $w_{1,1} = 14$. As can be seen that the same task has different WCETs on different processors due to the heterogeneity of the processors. Weight 18 of edge between n_1 and n_2 represents the communication time, denoted by $c_{1,2}$ if n_1 and n_2 are not assigned to the same processor.

B. Reliability Model

There are two major temporal type of failures, namely, the transient failure (i.e., random hardware failures) and the permanent failure. This study only considers the transient failure of processors because some functional safety standards (e.g., ISO 26262) only combine the random hardware failures and reliability together [24]. ISO 26262 defines the conception of exposure for random hardware failures. Exposure means the relative expected frequency of the operational conditions in which the injury may happen [24]. ISO 26262 also gives the duration/probability of exposure in [24, Table B.2, Annex B of Part 3]. According to ISO 26262, for a given exposure level, the corresponding reliability goal can be found in Table III. The relationships in Table III show that when the exposure is high, the corresponding reliability goal is low, which means that the risk of occurrence is high.

As pointed out in ISO 26262, random hardware failures occur unpredictably during the life time of a hardware but follow a probability distribution [24]. In general, the occurrence of transient fault for a task in a DAG-based application follows the Poisson distribution [6], [7], [14], [19]–[21]. The reliability of an event in unit time t is denoted by $R(t) = e^{-\lambda t}$, where λ is the *constant failure rate per time unit* for a processor. Let λ_k represent the constant failure rate per time unit of the processor u_k, and the reliability of n_i executed on u_k in its execution time is donated by

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

A development life cycle of safety-critical systems usually contains analysis, design, implementation, and testing phases. In this study, we aim to minimize the resource consumption cost of a parallel application with reliability goal during the design phase. Similar to WCTT of messages and WCET of tasks, we assume that the failure rates have been known and obtained in the analysis phase. Similar to [6], [20], and [21], the

reliability of the parallel application with precedence-constrained tasks should be

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

where $u_{\text{proc}(n_i)}$ represents the assigned processor of n_i.

As some communication networks themselves provide redundant links for high fault tolerance in embedded systems (e.g., FlexRay), similar to most studies [6], [14], [21], this study only involves the processor failure, and does not include the communication failure into the problem (i.e., the communication is assumed to be reliable in this study).

C. Resource Consumption Cost Model

The priced timed automata defines the computation resource cost rate γ per time unit for task execution, which means that the computation resource cost grows constantly with γ in a processor (location) and the computation resource grows constantly with computation resource rate in a processor to access data stored in memory [17]. In other words, for a task with execution time w, the cost should be $\gamma \times w$.

As heterogeneous processors are used in this study, all processors should have individual computation cost rates because different processors are configured with different memory components and access operations. Hence, let $U = \{\gamma_1, \gamma_2, \ldots, \gamma_{|U|}\}$ represent a set of rates on heterogeneous processors.

The priced timed automata defines fixed cost for message transitions (transmission). However, when a message is transmitted on a distributed embedded system, different sizes of messages should consume different resource costs. The reason is that each message should be packed before it is sent and should be unpacked after it is received, even though the packing and unpacking times depend on the sizes of messages. As homogeneous communication links are employed in this study, the communication cost rate γ_{comm} per time unit is the same for each message transmission.

When a task is finished in one processor, this task sends messages to all its successor tasks that may be located in different processors. Therefore, let $\text{cost}(n_i, u_k)$ represent the generated resource consumption cost of the task n_i on the processor u_k and is calculated by

$$\text{cost}(n_i, u_k) = w_{i,k} \times \gamma_k + \sum_{n_x \in \text{pred}(n_i)} c'_{x,i} \times \gamma_{\text{comm}} \tag{3}$$

where $c'_{x,i}$ represents the actual communication time between n_x and n_i. If n_x and n_i are assigned to the same processor, then $c'_{x,i} = 0$; otherwise, $c'_{x,i} = c_{x,i}$. That is, the $\text{cost}(n_i, u_k)$ includes the processor resource consumption cost that n_i is executed on u_k and the communication resource consumption cost that messages are transmitted between n_i and its predecessors when they are not assigned the same processor. Note that the current task just considers its predecessors, and its successors will consider the current task in calculating communication resource consumption cost.

Finally, the total resource consumption cost of G is the sum of that of all the tasks, namely

$$\text{cost}(G) = \sum_{n_i \in N} \text{cost}\left(n_i, u_{\text{proc}(i)}\right). \tag{4}$$

IV. Preliminaries

A. Reliability Goal

As the WCET of each task on each processor is known, the minimum and maximum reliability values can be obtained by traversing all the processors. Both of them are calculated by

$$R_{\min}(n_i) = \min_{u_k \in U} R(n_i, u_k) \tag{5}$$

and

$$R_{\max}(n_i) = \max_{u_k \in U} R(n_i, u_k) \tag{6}$$

respectively.

As the reliability of the application G is the product of reliability values of all the tasks (2), the minimum and maximum reliability values of G are calculated by

$$R_{\min}(G) = \prod_{n_i \in N} R_{\min}(n_i) \tag{7}$$

and

$$R_{\max}(G) = \prod_{n_i \in N} R_{\max}(n_i). \tag{8}$$

As mentioned earlier, if the reliability goal $R_{\text{goal}}(G)$ can be satisfied, then the application is reliable. Note that $R_{\text{goal}}(G)$ should be larger than or equal to $R_{\min}(G)$; otherwise, $R_{\text{goal}}(G)$ is always satisfied. Meanwhile, $R_{\text{goal}}(G)$ should be less than or equal to $R_{\max}(G)$; otherwise, $R_{\text{goal}}(G)$ cannot always be satisfied. Hence, this study assumes that $R_{\text{goal}}(G)$ belongs to the scope $R_{\min}(G)$ and $R_{\max}(G)$, namely

$$R_{\min}(G) \leqslant R_{\text{goal}}(G) \leqslant R_{\max}(G). \tag{9}$$

B. Problem Description

Given a parallel application G with known reliability goal $R_{\text{goal}}(G)$ that would be executed on a heterogeneous multiprocessor set U, the problem to be addressed in this study is to assign an available processor to each task while reducing the resource consumption cost of G and satisfying its reliability goal $R_{\text{goal}}(G)$. The formal description is finding the processor assignments of all the tasks to minimize the resource consumption cost of the application G

$$\text{cost}(G) = \sum_{n_i \in N} \text{cost}\left(n_i, u_{\text{proc}(i)}\right)$$

subject to

$$R(G) = \prod_{n_i \in N} R\left(n_i, u_{\text{proc}(i)}\right) \geqslant R_{\text{goal}}(G)$$

for $\forall i : 1 \leqslant i \leqslant |N|, \quad u_{\text{proc}(i)} \in U$.

TABLE IV
UPWARD RANK VALUES OF THE TASKS OF THE PARALLEL APPLICATION IN FIG. 1 [10]

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

C. Task Prioritizing

The remaining work of this study is to assign each task to an appropriate processor after the reliability goal and problem description are determined. The first work is to determine the assignment order of tasks. Similar to [1], [6], [7], and [10], this study employs the upward rank value (rank_u) of a task given by (10) as the common task priority standard

$$\text{rank}_\text{u}(n_i) = \overline{w_i} + \max_{n_j \in \text{succ}(n_i)} \{c_{i,j} + \text{rank}_\text{u}(n_j)\} \tag{10}$$

where $\overline{w_i}$ represents the average WCET of task n_i and calculated by

$$\overline{w_i} = \left(\sum_{k=1}^{|U|} w_{i,k} \right) / |U|.$$

All the tasks are ordered according to the decreasing order of rank_u. Table IV shows the upward rank values of all the tasks in Fig. 1. Note that only if all the predecessors of n_i have been assigned to the processors, n_i will be prepared 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 assignment order in G is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

D. MaxRe and RR Algorithms

The state-of-the-art MaxRe and RR algorithms were presented to minimize resource consumption while satisfying the reliability goal of a parallel application on heterogeneous systems using fault tolerance. Calculating reliability goals of tasks and assigning tasks to processors are two important steps.

1) In calculating the reliability goal, both MaxRe and RR calculate the reliability goal of the entry task as $R_\text{goal}(n_1) = \sqrt[|N|]{R_\text{goal}(G)}$, where $R_\text{goal}(G)$ and $R_\text{goal}(n_1)$ represent the reliability goals of the application and the entry task, respectively, and $|N|$ represents the number of tasks; for the rest of tasks (i.e., nonentry tasks), the reliability goal of each task using MaxRe is still $R_\text{goal}(n_1) = \sqrt[|N|]{R_\text{goal}(G)}$, whereas that using RR is calculated by

$$R_\text{goal}(n_{\text{seq}(j)}) = \sqrt[|N|-j+1]{\frac{R_\text{goal}(G)}{\prod_{x=1}^{j-1} R(n_x)}}$$

where $R_\text{goal}(n_{\text{seq}(j)})$ represents the reliability goal of the jth task; however, the reliability goals for tasks are still too high and need to pay for more resource to satisfy.

2) In assigning each task to processor, both MaxRe and RR assign the task to the processor with the maximum reliability value $R(n_i, u_k) = e^{-\lambda_k w_{i,k}}$ (1); however, the maximum reliability value does not mean minimum resource consumption costs in heterogeneous systems because different processors have different failure and cost rates.

V. PROPOSED ALGORITHM

The problem of minimizing the resource consumption cost of a reliable parallel application on heterogeneous embedded systems comes down to two subproblems, namely, satisfying reliability goal and minimizing resource consumption cost. This section first solves the subproblem and then solves the second subproblem with a proposed algorithm.

A. Satisfying Reliability Goal

The strategy of satisfying reliability goal of the parallel application G is as follows. Assume that the task to be assigned is $n_{\text{seq}(j)}$, where $\text{seq}(j)$ represents the jth assigned task (sequence number), then $\{n_{\text{seq}(1)}, n_{\text{seq}(2)}, \ldots, n_{\text{seq}(j-1)}\}$ represents the task set where the tasks have been assigned, and $\{n_{\text{seq}(j+1)}, n_{\text{seq}(j+2)}, \ldots, n_{\text{seq}(|N|)}\}$ represents the task set where the tasks have not been assigned. To ensure that the reliability of the application is satisfied at each task assignment, we presuppose that each unassigned task in $\{n_{\text{seq}(j+1)}, n_{\text{seq}(j+2)}, \ldots, n_{\text{seq}(|N|)}\}$ is assigned to the processor with the maximum reliability value. Hence, when assigning $n_{\text{seq}(j)}$, the reliability of G is calculated by

$$R_{\text{seq}(j)}(G) = \prod_{x=1}^{j-1} R\left(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}\right) \times R\left(n_{\text{seq}(j)}, u_{\text{proc}(\text{seq}(j))}\right) \times \prod_{y=j+1}^{|N|} R_{\max}\left(n_{\text{seq}(y)}\right).$$

For any task $n_{\text{seq}(j)}$, only if $R_{\text{seq}(j)}(G) \geqslant R_\text{goal}(G)$, the actual reliability $R(G) = \prod_{n_i \in N} R\left(n_i, u_{\text{proc}(i)}\right)$ will be larger than or equal to $R_\text{goal}(G)$. The correctness of this strategy is proved in the Theorem 1.

Theorem 1: Each task $n_{\text{seq}(j)}$ in the parallel application G can always find a processor to be assigned to satisfy

$$R_{\text{seq}(j)}(G) = \prod_{x=1}^{j-1} R\left(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}\right) \times R\left(n_{\text{seq}(j)}, u_{\text{proc}(\text{seq}(j))}\right) \times \prod_{y=j+1}^{|N|} R_{\max}\left(n_{\text{seq}(y)}\right) \geqslant R_\text{goal}(G). \tag{11}$$

The theorem is proved in the Appendix.

B. Minimizing Resource Consumption Cost

First, the reliability goal of each task is given before the algorithm is proposed. According to Theorem 1, it should have

$$R(n_{\text{seq}(j)}, u_k) \geqslant R_{\text{goal}}(G) \Bigg/ \left(\prod_{x=1}^{j-1} R(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}) \times \prod_{y=j+1}^{|N|} R_{\max}(n_{\text{seq}(y)}) \right).$$

Hence, let the reliability goal of the task $n_{\text{seq}(y)}$ be

$$R_{\text{goal}}(n_{\text{seq}(j)}) = R_{\text{goal}}(G) \Bigg/ \left(\prod_{x=1}^{j-1} R(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}) \times \prod_{y=j+1}^{|N|} R_{\max}(n_{\text{seq}(y)}) \right). \quad (12)$$

Then, the reliability goal of the application can be transferred to the reliability goal of each task. That is, just let $n_{\text{seq}(j)}$ satisfy the following constraint:

$$R(n_{\text{seq}(j)}, u_k) \geqslant R_{\text{goal}}(n_{\text{seq}(j)}).$$

Hence, when assigning the task $n_{\text{seq}(j)}$, the reliability goal $R_{\text{goal}}(n_{\text{seq}(j)})$ of $n_{\text{seq}(j)}$ can be directly considered and the reliability goal of application G need not be concerned about. In this way, a low time complexity heuristic algorithm can be achieved. As the minimum reliability goal of $n_{\text{seq}(j)}$ is $R_{\min}(n_{\text{seq}(j)})$, $R(n_{\text{seq}(j)})$ should be required to satisfy the following constraint:

$$R(n_{\text{seq}(j)}, u_k) \geqslant \max\{R_{\text{goal}}(n_{\text{seq}(j)}), R_{\min}(n_{\text{seq}(j)})\}. \quad (13)$$

In the following, the algorithm called minimizing resource consumption cost with reliability goal (MRCRG) for a reliable parallel application on heterogeneous embedded systems is proposed, as shown in Algorithm 1.

The core idea of MRCRG is that the reliability goal of the application is transferred to the reliability goal of each task. Each task just selects the processor with the minimum resource consumption cost while satisfying its reliability goal. The main details are explained as follows.

1) MRCRG has obtained the reliability goal of each task before it prepares to be assigned (line 5).
2) MRCRG skips the processors that do not satisfy the reliability goal (lines 9–11). That is, MRCRG does not need to calculate the total reliability of the application and determine whether it satisfies the given reliability goal in each task assignment by traversing all the tasks in the parallel application.
3) MRCRG selects the processor with the minimum resource consumption cost for each task while satisfying the condition $\text{RC}(n_i, u_k) < \max\{R_{\text{goal}}(n_i), R_{\min}(n_i)\}$ (lines 9–17).

The time complexity of the MRCRG algorithm is analyzed as follows. Scheduling all tasks must traverse all tasks, which can be done in O($|N|$) time. Calculating the minimum resource consumption value of each task can be done in O($|N| \times |U|$) time. Hence, the time complexity of the MRCRG algorithm is $O(|N|^2 \times |U|)$, which is equal to that of the heterogeneous earliest finish time (HEFT) algorithm [10]. In other words, MRCRG implements effective resource consumption cost minimization without sacrificing the time complexity.

Algorithm 1: The MRCRG Algorithm.

Input: $U = \{u_1, u_2, ..., u_{|U|}\}$, G
Output: $R(G)$, $cost(G)$
1: Sort the tasks in a list $downward\_task\_list$ by descending order of $rank_u$ values.
2: **while** (there are tasks in $downward\_task\_list$) **do**
3: $n_i = downward\_task\_list.out()$;
4: Calculate $R_{\min}(n_i)$, $R_{\max}(n_i)$ using Eqs. (5) and (6), respectively;
5: Calculate $R_{\text{goal}}(n_i)$ using Eq. (12); // 1) calculate the reliability goal of n_i before it prepares to be assigned.
6: var $u_{proc(i)} \leftarrow NULL$, $cost(n_i, u_{proc(i)}) \leftarrow \infty$, $R(n_i, u_{proc(i)}) \leftarrow 0$;
7: **for** (each processor $u_k \in U$) **do**
8: Calculate $R(n_i, u_k)$ using Eq. (1);
9: **if** ($R(n_i, u_k) < \max\{R_{\text{goal}}(n_i), R_{\min}(n_i)\}$) **then**
10: continue;// 2) skip the processors that do not satisfy the reliability goal of n_i.
11: **end if**
12: Calculate $cost(n_i, u_k)$ using Eq. (3);
13: **if** ($cost(n_i, u_k) < cost(n_i, u_{proc(i)})$) **then**
14: $proc(i) \leftarrow k$;
15: $R(n_i, u_{proc(i)}) \leftarrow R(n_i, u_k)$;
16: $cost(n_i, u_{proc(i)}) \leftarrow cost(n_i, u_k)$; // 3) select the processor with the minimum resource consumption cost $cost(n_i, u_{proc(i)})$.
17: **end if**
18: **end for**
19: **end while**
20: Calculate the actual reliability $R(G)$ using Eq. (2);
21: Calculate the final resource consumption cost $cost(G)$ using using Eq.(4).

TABLE V
FAILURE AND COST RATES OF PROCESSORS $\{u_1, u_2, u_3\}$

Parameter	u_1	u_2	u_3
λ_k	0.0005	0.0002	0.0009
γ_k	5	9	2

C. Example of the MRCRG Algorithm

The process and results of the motivating example using the MRCRG algorithm are illustrated in this section. The failure and resource consumption cost rates of processors are shown in Table V and the rate of the link is $\gamma_{\text{comm}} = 1$. For simplicity, all the units of all parameters are ignored in the example.

The minimum and maximum reliability values are $R_{\min}(G) = 0.8792$ and $R_{\max}(G) = 0.9743$ according to (7) and (8), respectively. The reliability goal of G is set to $R_{\text{goal}}(G) = 0.95$. Table VI shows the task assignments of the motivating example using MRCRG, where each row represents a task assignment. The values in red text mean that the processor is selected with the minimum resource consumption cost. The values denoted with "–" mean that assigning task to the processor cannot satisfy the reliability goal of the task. The final reliability is $R(G) = 0.9502 > R_{\text{goal}}(G) = 0.95$, and the obtained total resource consumption cost is $\text{cost}(G) = 910$.

Fig. 2 also shows the task scheduling of the motivating parallel application G based on after generated by MRCRG, where the schedule length (makespan) is 113. Note that the arrows in Fig. 2 represent generated communications between tasks. The later

TABLE VI
TASK ASSIGNMENT OF THE MOTIVATING PARALLEL APPLICATION GENERATED BY MRCRG

n_i	$R_{goal}(n_i)$	$cost(n_i, u_1)$	$cost(n_i, u_2)$	$cost(n_i, u_3)$	$R(n_i)$
n_1	0.9919	70	144	18	0.9919
n_3	0.9830	55	117	38	0.9830
n_4	0.9925	74	81	-	0.9935
n_2	0.9952	-	189	-	0.9962
n_5	0.9964	-	117	-	0.9974
n_6	0.9958	-	158	-	0.9968
n_9	0.9966	-	131	-	0.9976
n_7	0.9960	35	135	-	0.9965
n_8	0.9973	59	133	-	0.9975
n_{10}	0.9984	-	91	-	0.9986
$cost(G) = 910$, $R(G) = 0.9502 > R_{\text{goal}}(G) = 0.95$					

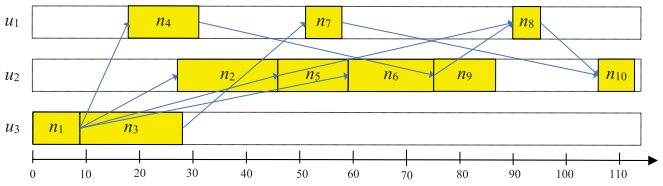


Fig. 2. Task scheduling of the motivating parallel application generated by MRCRG.

implementation phase can implement the application according to the assigned processor, start time, and end time of each task generated by MRCRG during the design phase.

VI. EXPERIMENTS

A. Experimental Metrics

The performance metrics selected for comparison are the actual reliability value $R(G)$ (2) and the final resource consumption cost $\text{cost}(G)$ (4) of the application. The compared algorithms with the presented MRCRG algorithm are of course the sate-of-the-art MaxRe [6] and RR algorithms [7]. Processor and application parameters refer to [20], [21]: 10 ms $\leqslant w_{i,k} \leqslant$ 100 ms, 10 ms $\leqslant c_{i,j} \leqslant$ 100 ms, 0.000001/ms $\leqslant \lambda_k \leqslant$ 0.000009/ms, 0.1 Kb/ms $\leqslant \gamma_k \leqslant$ 0.9 Kb/ms, and $\gamma_{\text{comm}} = 0.2$ Kb/ms. All parallel applications will be executed in a simulated heterogeneous multiprocessor platform with 32 processors.

B. Real Parallel Applications

Parallel applications with precedence-constrained tasks widely exist in some embedded applications, such as the Gaussian elimination [25] and fast Fourier transform [26] applications. The Gaussian elimination and Fourier transform application are the two typical parallel applications with low and high parallelism, respectively. To verify the effectiveness and validity of the proposed algorithm, these two types of real parallel applications are used to compare the results of all the algorithms. A new parameter ρ is used as the matrix size of the Gaussian elimination application and the total number of tasks should be $|N| = \frac{\rho^2+\rho-2}{2}$ [10]. Fig. 3 shows an example of the Gaussian elimination application with $\rho = 5$. A new parameter ρ is used as the size of the fast Fourier transform application and the total number of tasks should be $|N| = (2 \times \rho - 1) + \rho \times \log_2 \rho$, where assume that $\rho = 2^y$ for some integer y. Fig. 4 shows an example of the fast Fourier transform application with $\rho = 8$. Note that ρ exit tasks exist in the fast Fourier transform application with the size ρ. To adapt the application model of this study, a virtual exit task is added and the last ρ tasks are set as the immediate predecessor tasks of the virtual task. In this situation, the total task number should be $|N| = (2 \times \rho - 1) + \rho \times \log_2 \rho + 1$.

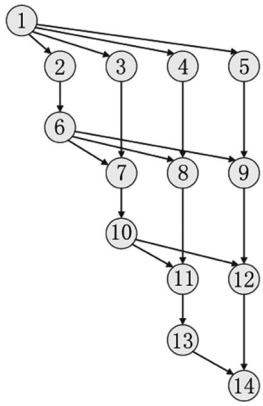


Fig. 3. Example of the Gaussian elimination application with $\rho = 5$.

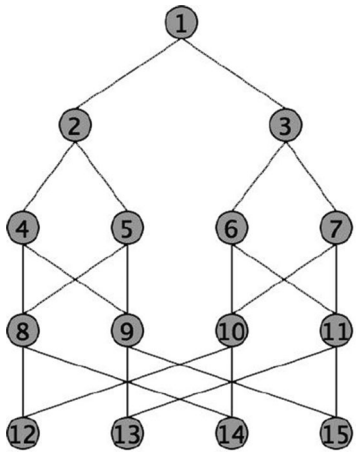


Fig. 4. Example of the fast Fourier transform application with $\rho = 4$.

Experiment 1: This experiment is conducted to compare the actual reliability values and the total resource consumption costs of Gaussian elimination applications for varying number of tasks. We limit $R_{\text{goal}}(G) = 0.93$, because it is contained in the interval of the reliability goals of the exposure E3 (i.e., [0.9,0.99)), which is primarily used in actual system design. ρ is changed from 14 to 70, i.e., the task number is changed from 104 (small scale) to 2484 (large scale). The results are shown in Fig. 5.

Fig. 5(a) shows the actual reliability values of Gaussian elimination applications on different task scales. As can be seen, all the algorithms can satisfy the reliability goals of applications in all cases and the actual reliability values decrease with the increased numbers of tasks. In small scales ($|N| = 104$, $|N| = 405$, and $|N| = 902$), the three algorithms obtain the same reliability values. With the increasing numbers of tasks,

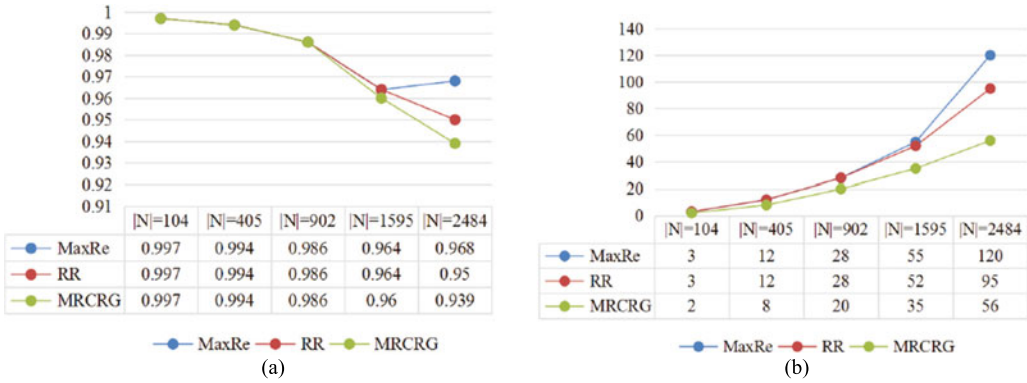


Fig. 5. Results of Gaussian elimination applications for varying numbers of tasks. (a) Actual reliability values. (b) Total resource consumption costs (Mb).

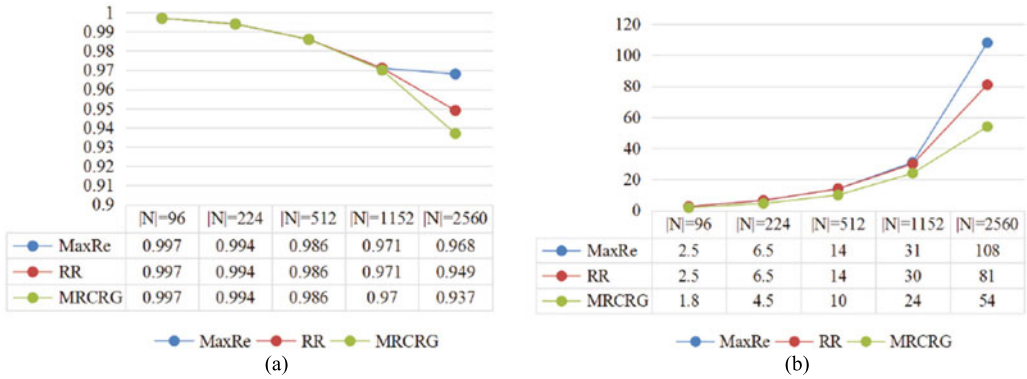


Fig. 6. Results of fast Fourier transform applications for varying numbers of tasks. (a) Actual reliability values. (b) Total resource consumption costs (Mb).

MRCRG generates least reliability values followed by RR and MaxRe. For example, when $|N| = 2484$, the reliability value using MRCRG is merely 0.939, which is very close to the reliability goal value; however, the obtained reliability values using RR and MaxRe reach to 0.95 and 0.968, respectively.

Fig. 5(b) shows the total resource consumption costs of Gaussian elimination applications on different task scales. It is pleasing that MRCRG generates the least resource consumption costs followed by RR and MaxRe. In small scales ($|N| = 104$, $|N| = 405$, and $|N| = 902$), RR and MaxRe generates the same resource consumption costs. The reason for the same resource consumption costs is that RR and MaxRe only need one replicas for each task and they do not require fault tolerance; in addition, considering that MaxRe and RR assign the task to the processor with the maximum reliability value $R(n_i, u_k) = e^{-\lambda_k w_{i,k}}$ (1), the actual resource consumption costs should be equal. In small scales ($|N| = 104$, $|N| = 405$, and $|N| = 902$), the resource consumption costs using MRCRG are merely 66.7% (2/3) of those of using RR and MaxRe. When $|N| = 2484$, the resource consumption cost using MRCRG is merely 46.6% (56/120) and 58.9% (56/95) of those of using RR and MaxRe, respectively. The reason for the least resource consumption costs using MRCRG is that it improves the strategy of satisfying reliability goal (Theorem 1); moreover, MRCRG assigns the tasks to the processors with the minimum resource consumption cost rather than minimum reliability values of RR and MaxRe because maximum reliability value does not mean minimum resource consumption cost in heterogeneous systems.

Experiment 2: This experiment is conducted to compare the actual reliability values and the total resource consumption costs of fast Fourier transform applications for varying number of tasks. Limit $R_{\text{goal}}(G) = 0.93$. ρ is changed from 16 to 256, i.e., the task number is changed from 96 (small scale) to 2560 (large scale). The results are shown in Fig. 6.

Fig. 6(a) shows the actual reliability values of fast Fourier transform applications on different task scales. Compared with the results of Gaussian elimination applications in Fig. 5(a), the same regular pattern is shown in Fig. 6(a) in different cases. Such results indicate that both the low and high parallel applications have not obvious differences in obtaining the actual reliability values.

Fig. 6(b) shows the total resource consumption costs of fast Fourier transform applications on different task scales. Similar to the results of Fig. 5(b), MRCRG generates the least resource consumption costs followed by RR and MaxRe in all cases, and RR and MaxRe generate the same resource consumption costs in small scales ($|N| = 96$, $|N| = 224$, and $|N| = 512$). The main difference is that the advantage of the resource consumption costs using MaxRe is slightly weakened to some extent. For example, in small scales ($|N| = 96$, $|N| = 224$, and $|N| = 512$), the resource consumption costs using MRCRG are merely 72% of those of using RR and MaxRe. When $|N| = 2560$, the

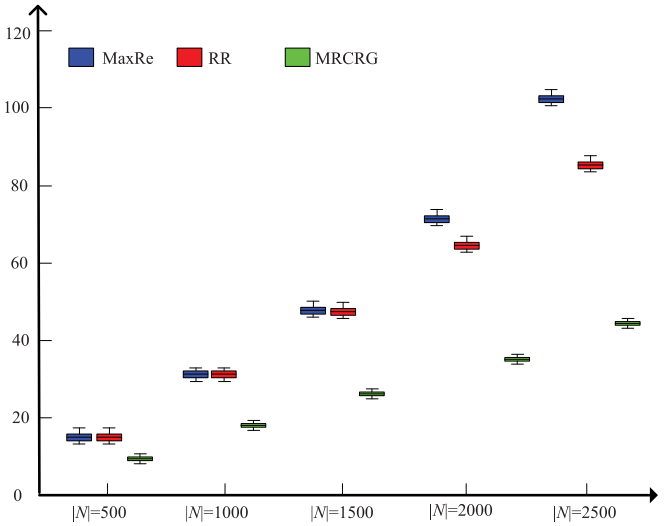


Fig. 7. Resource consumption costs (unit: Mb) of randomly generated parallel applications for varying numbers of tasks.

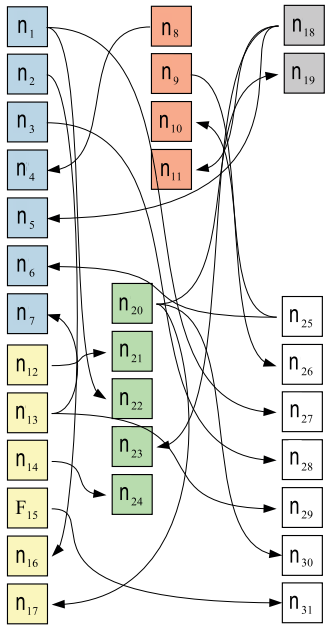


Fig. 8. Real-life automotive application [28].

resource consumption cost using MRCRG is about 50% (54/108) and 66.7% (54/81) of those of using RR and MaxRe, respectively. Note that the resource consumption costs using MRCRG are not reduced in large scale for different applications, but RR and MaxRe have generate less resource consumption costs in high parallel applications than those in in low parallel applications. Anyway, MRCRG still produces less resource consumption costs than RR and MaxRe in all cases and the advantage is significant for both low and high parallel applications.

C. Randomly Generated Parallel Applications

To fully demonstrate the benefits of the proposed algorithm, we consider parallel application samples, which are randomly generated depending on the following parameters: shape parameter set $\alpha = \{0.5, 1.0, 2.0\}$, range percentage of computation time set $\eta = \{0.1, 0.5, 1.0, 2.0, 4.0\}$, and communication to computation ratio CCR $= \{0.1, 0.5, 1.0, 5.0, 10.0\}$. Random DAGs have been generated by task graphs for free 3.5 [27], and programmed by Java to compare the results. Statistical experimental results are shown with "box and whiskers graphs," which can show the concrete values with maximum point, minimum point, middle point (the median), and the middle points of the two halves (submedians) clearly [12].

Experiment 3: This experiment is conducted to compare the total resource consumption costs of randomly generated parallel applications for varying number of tasks. Let $R_{\text{goal}}(G)$ still be 0.93. The task number is changed from 500 (small scale) to 2500 (large scale) with 500 increments. Statistical experimental results are shown with "box and whiskers graphs," which can show the concrete values with maximum point, minimum point, middle point (the median), and the middle points of the two halves (submedians) clearly [12]. The results of total resource consumption costs are shown in Fig. 7.

As can be seen in Fig. 7, MRCRG generates much more resource consumption costs than MaxRe and RR in all the same scales. It is easy to see that the resource consumption costs with 2500 tasks using MRCRG is even less than those with 1500 tasks using MaxRe and RR.

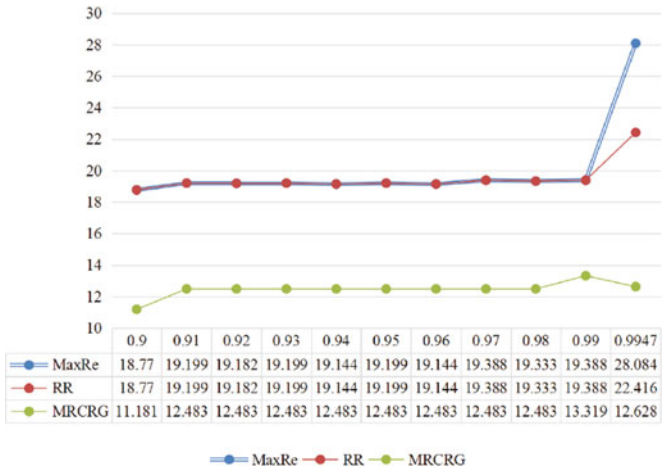
D. Real-Life Automotive Application

In order to adapt to ISO 26262, we use the real-life automotive application shown in Fig. 8 adopted from [28] 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 are as follows: 100 μs $\leqslant w_{i,k} \leqslant$ 400 μs, 100 μs $\leqslant c_{i,j} \leqslant$ 400 μs, 0.000001/μs $\leqslant \lambda_k \leqslant$ 0.000009/μs, 0.5 kb/ms $\leqslant \gamma_k \leqslant$ 1.5 kb/ms, and $\gamma_{\text{comm}} = 0.5$ kb/ms.

Experiment 4: This experiment is conducted to compare the final resource consumption costs of the real-life automotive application for varying reliability goals. The reliability goal is changed from 0.9 to 0.99 with a 0.01 increment because reliability goal values fall in the range of exposure E3 and E2 in ISO 26262. Meanwhile, the maximum reliability goal for the application is calculated as 0.9947, which falls in the range of exposure E1 in ISO 26262.

Fig. 9 shows the actual reliability values of the real-life application for varying reliability goals. The following observations are made.

1) MRCRG still generates least resource consumption costs followed by MaxRe and RR. MRCRG outperforms MaxRe and RR by 55.03% and 43.66%, respectively, in the best case.
2) The maximum resource consumption costs using MaxRe and RR are obtained when $R_{\text{goal}}(G) = 0.9947$, whereas that using MRCRG is obtained when $R_{\text{goal}}(G) = 0.99$. Such results indicate that higher reliability goals do not lead to lower resource consumption costs using MRCRG.



	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99	0.9947
MaxRe	18.77	19.199	19.182	19.199	19.144	19.199	19.144	19.388	19.333	19.388	28.084
RR	18.77	19.199	19.182	19.199	19.144	19.199	19.144	19.388	19.333	19.388	22.416
MRCRG	11.181	12.483	12.483	12.483	12.483	12.483	12.483	12.483	12.483	13.319	12.628

Fig. 9. Resource consumption costs (unit: Mb) of real-life automotive application for varying reliability goals.

3) MRCRG generates relatively stable resource consumption cost of 12.483 Mb in different reliability goals of 0.91–0.98. In other words, a reliability goal that in this interval can be accepted for the real-life automotive application.
4) The maximum reliability goal of 0.9947 shows that it is unrealistic to achieve very high and unreachable reliability goal value (e.g., 0.999). In this case, a relatively low but safe reliability goal (e.g., 0.93) can be achieved.

In summary, combined with the results of real and randomly generated applications, the proposed MRCRG algorithm is very effective in resource consumption cost minimization while satisfying the given reliability goal. It is expected that MRCRG can effectively explore a part of the design space and achieve better solutions during the design phase aiming at reducing the resource consumption cost.

VII. Conclusion

This study develops an effective and low time complexity resource consumption cost minimization algorithm MRCRG for a parallel application on heterogeneous embedded systems. MRCRG is implemented by transferring the reliability goal of the parallel application to the reliability goal of each task. In each task assignment, MRCRG always selects the processor with the minimum resource consumption cost while satisfying the reliability goal of the task. First, MRCRG can always satisfy the reliability goal of a parallel application and is verified with various kinds of experiments. Second, MRCRG implements the more effective resource consumption cost minimization while satisfying the reliability goal compared with the state-of-the-art algorithms. MRCRG can be used for different types of real parallel applications in heterogeneous embedded systems. It is expected that MRCRG can effectively explore a part of the design space aiming at minimizing the resource consumption cost of reliable parallel applications under different conditions. In our future work, we will extend our method to the energy consumption optimization of heterogeneous embedded systems.

Appendix

Proof of Theorem 1: The mathematical induction is used to prove the theorem. First, the entry task $n_1 = n_{\text{seq}(1)}$ is considered. In this case, all the tasks are not assigned to processors and the application G needs to satisfy its reliability goal

$$R_{\text{seq}(1)}(G) = R(n_{\text{seq}(1)}, u_k) \times \prod_{y=2}^{|N|} R_{\max}(n_{\text{seq}(y)}) \geqslant R_{\text{goal}}(G) \tag{14}$$

namely, $n_{\text{seq}(1)}$ is required to be satisfied

$$R(n_{\text{seq}(1)}, u_k) \geqslant R_{\text{goal}}(G) \Big/ \prod_{y=2}^{|N|} R_{\max}(n_{\text{seq}(y)}). \tag{15}$$

As

$$R_{\max}(G) = R_{\max}(n_{\text{seq}(1)}) \times \prod_{y=2}^{|N|} R_{\max}(n_{\text{seq}(y)}) \geqslant R_{\text{goal}}(G) \tag{16}$$

according to (8) and (9), then

$$R_{\max}(n_{\text{seq}(1)}) \geqslant R_{\text{goal}}(G) \Big/ \prod_{y=2}^{|N|} R_{\max}\left(n_{\text{seq}(y)}\right). \tag{17}$$

As the maximum value of $R(n_{\text{seq}(1)}, u_k)$ is $R_{\max}(n_{\text{seq}(1)})$, then $n_{\text{seq}(1)}$ is able to find an assigned processor to satisfy the (14), namely

$$R_{\text{seq}(1)}(G) = R(n_{\text{seq}(1)}, u_k) \times \prod_{y=2}^{|N|} R_{\max}\left(n_{\text{seq}(y)}\right) \geqslant R_{\text{goal}}(G). \tag{18}$$

Assume that the jth task $n_{\text{seq}(j)}$ can find an assigned processor $u_{\text{proc}(seq(j))}$ to satisfy the $R_{\text{goal}}(G)$, then

$$R_{\text{seq}(j)}(G) = \sum_{x=1}^{j-1} R(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}) \times R(n_{\text{seq}(j)}, u_{\text{proc}(\text{seq}(j))}) \times \prod_{y=j+1}^{|N|} R_{\max}\left(n_{\text{seq}(y)}\right) \geqslant R_{\text{goal}}(G) \tag{19}$$

namely

$$R_{\text{seq}(j)}(G) = \prod_{x=1}^{j} R(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}) \times \prod_{y=j+1}^{|N|} R_{\max}(n_{\text{seq}(y)}) \geqslant R_{\text{goal}}(G). \tag{20}$$

Hence, it should have

$$\prod_{x=1}^{i} R(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}) \geqslant R_{\text{goal}}(G) \Big/ \prod_{y=j+1}^{|N|} R_{\max}(n_{\text{seq}(y)}). \tag{21}$$

For the $(j+1)$th task $n_{\text{seq}(j+1)}$, the reliability of the application is

$$R_{\text{seq}(j+1)}(G) = \prod_{x=1}^{j} R(n_{\text{seq}(x)}, u_{\text{proc}(\text{seq}(x))}) \times R(n_{\text{seq}(j+1)}, u_k) \times \prod_{y=j+2}^{|N|} R(n_{\text{seq}(y)}, u_{\text{proc}(\text{seq}(y))}). \quad (22)$$

Placing (21) into (22) gives the following:

$$\begin{aligned} R_{\text{seq}(j+1)}(G) &\geqslant \left(R_{\text{goal}}(G) \Big/ \prod_{y=j+1}^{|N|} R_{\max}(n_{\text{seq}(y)} \right) \\ &\times R(n_{\text{seq}(j+1)}, u_k) \times \prod_{y=j+2}^{|N|} R_{\max}(n_{\text{seq}(y)})) \\ &= \left(R_{\text{goal}}(G) \Big/ (R_{\max}(n_{\text{seq}(j+1)}) \right) \times R(n_{\text{seq}(j+1)}, u_k). \end{aligned} \quad (23)$$

As the maximum value of $R(n_{\text{seq}(j+1)}, u_k)$ is $R_{\max}(n_{\text{seq}(j+1)})$, when $R(n_{\text{seq}(j+1)}, u_k) = R_{\max}(n_{\text{seq}(j+1)})$, then it should have

$$R_{\text{seq}(j+1)}(G) \geqslant R_{\text{goal}}(G) \quad (24)$$

according to (23). That is, $n_{\text{seq}(j+1)}$ can also find an assigned processor to satisfy $R_{\text{goal}}(G)$.

As all the tasks can find individual assigned processors to satisfy $R_{\text{goal}}(G)$, Theorem 1 is satisfied. ■

ACKNOWLEDGMENT

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

REFERENCES

[1] G. Xie, G. Zeng, L. Liu, R. Li, and K. Li, "Mixed real-time scheduling of multiple dags-based applications on heterogeneous multi-core processors," *Microprocess. Microsyst.*, vol. 47, pp. 93–103, Nov. 2016.

[2] A. Schranzhofer, J. J. Chen, and L. Thiele, "Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms," *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 692–707, Nov. 2010.

[3] J. Castrillon, R. Leupers, and G. Ascheid, "Maps: Mapping concurrent dataflow applications to heterogeneous MPSoCs," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 527–545, Feb. 2013.

[4] J. Henkel *et al.*, "Design and architectures for dependable embedded systems," in *Proc. 9th Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2011, pp. 69–78.

[5] M. Chtepen, F. H. Claeys, B. Dhoedt, F. De Turck, P. Demeester, and P. A. Vanrolleghem, "Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 180–190, May 2009.

[6] 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.

[7] 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.

[8] J. Machrouh *et al.*, "Cross domain comparison of system assurance," in *Proc. Embedded Real Time Softw. Syst.*, Toulouse, France, 2012, pp. 1–3.

[9] 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.

[10] 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, Aug. 2002.

[11] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Comput.*, vol. 38, no. 4, pp. 175–193, Apr. 2012.

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

[13] A. Doğan and F. Özgüner, "Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems," *Comput. J.*, vol. 48, no. 3, pp. 300–314, Mar. 2005.

[14] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Depend. Sec. Comput.*, vol. 6, no. 4, pp. 241–254, Oct.–Dec. 2009.

[15] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proc. 19th Annu. ACM Symp. Parallel Algorithms Archit.*, 2007, pp. 280–288.

[16] M. Qiu and E. H.-M. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, Mar. 2009, Art. no. 25.

[17] T. Ovatman, A. W. Brekling, and M. R. Hansen, "Cost analysis for embedded systems: Experiments with priced timed automata," *Electron. Notes Theor. Comput. Sci.*, vol. 238, no. 6, pp. 81–95, 2010.

[18] M. W. Convolbo and J. Chou, "Cost-aware dag scheduling algorithms for minimizing execution cost on cloud resources," *J. Supercomput.*, vol. 72, no. 3, pp. 985–1012, Jan. 2016.

[19] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Rel.*, vol. 38, no. 1, pp. 16–27, Apr. 1989.

[20] 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.

[21] 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, pp. 99–109, Mar. 2013.

[22] X. Tang, K. Li, R. Li, and B. Veeravalli, "Reliability-aware scheduling strategy for heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 941–952, Sep. 2010.

[23] X. Tang, K. Li, M. Qiu, and E. H.-M. Sha, "A hierarchical reliability-driven scheduling algorithm in grid systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 4, pp. 525–535, 2012.

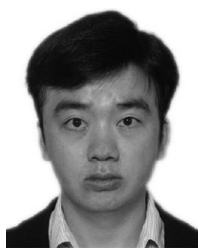
[24] *Road Vehicles-Functional Safety*, ISO 26262, 2011.

[25] 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.

[26] 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. 2015 Conf. Des. Archit. Signal Image Process.*, 2015, pp. 1–7.

[27] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. 6th Int. Workshop Hardware/Softw. Codes.*, 1998, pp. 97–101.

[28] 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.



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

From 2014 to 2015, he was a Postdoctoral Researcher at Nagoya University, Nagoya, Japan. Since 2015, he has been working as a Postdoctoral Researcher at Hunan University. His research interests include embedded and real-time systems, parallel and distributed systems, and software engineering and methodology.

Dr. Xie received the Best Paper Award from IEEE International Symposium on Parallel and Distributed Processing with Applications 2016. He is a member of ACM and China Computer Federation.

Yuekun Chen is currently working toward the Ph.D. degree in computer science and engineering at Hunan University, Changsha, China.

Her research interests include service computing, fault-tolerance computing, and software engineering.

Yan Liu received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2010. He is an Assistant Professor in the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China.

His research interests include computer architectures and embedded computing systems.

Prof. Liu is a member of the China Computer Federation.

Yehua Wei received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2009. He is an Associate Professor in the College of Physics and Information Science, Hunan Normal University, Changsha, China.

His research interests include cyber-physical systems and Internet of Things.

Prof. Wei is a member of the China Computer Federation.

Renfa Li (M'05–SM'10) received the Ph.D. degree in electronic engineering from Huazhong University of Science and Technology, Wuhan, China, in 2003. He is a Professor of computer science and electronic engineering, and the Dean in the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. He is the Director in the Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University. He is also an expert committee member in the National Supercomputing Center, Changsha, China. His research interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of Things.

Prof. Li is a member of the council of China Computer Federation and a Senior Member of ACM.

Keqin Li (M'90–SM'96–F'15) received the Ph.D. degree in computer science from the University of Houston, Houston, Texas, USA, in 1990. He is a SUNY Distinguished Professor of computer science. He has published more than 460 journal articles, book chapters, and refereed conference papers. His 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.

Prof. Li received several best paper awards. He is currently 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.