Price Performance-Driven Hardware Cost Optimization Under Functional Safety Requirement in Large-Scale Heterogeneous Distributed Embedded Systems

Guoqi Xie, *Member, IEEE*, Wenhong Ma, Hao Peng, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—The problem of optimizing hardware cost under functional safety requirement is a desirable work for a safety-critical embedded application. The state-of-the-art algorithms called enhanced explorative hardware cost optimization (EEHCO) and simplified EEHCO (SEEHCO) have been used to study this problem for a distributed embedded application by iteratively removing some processors from opened processors (i.e., open-to-close). However, EEHCO has powerful cost optimization capability but inferior time efficiency, and vice versa for SEEHCO in large-scale heterogeneous distributed embedded systems. This study presents a price performance-driven hardware cost optimization (PPHCO) method, which is the combination of PPHCO1 and PPHCO2, to achieve powerful cost optimization capability and superior time efficiency simultaneously. PPHCO1 iteratively selects the processor with the maximum price performance to open and overcomes the inferior time efficiency (i.e., close-to-open). PPHCO2 iteratively selects the processor with the minimum price performance to close and further optimizes the hardware cost on the basis of PPHCO1 without losing time efficiency (i.e., open-to-close). Through significantly reducing the iteration count, PPHCO overcomes the inferior time efficiency of the open-to-close method. Through adopting union fast functional safety verification (UFFSV), PPHCO achieves powerful cost optimization capability. Experiments confirm that PPHCO not only achieves stronger cost optimization capability but also has better time efficiency than state-of-the-art EEHCO and SEEHCO algorithms.**

***Index Terms*—Functional safety, hardware cost, price performance.**

I. Introduction

WE HAVE entered the era of Industrie 4.0, which creates a smart factory. Within the modular structured smart factory, cyber-physical systems (CPS) monitor physical processes, create a virtual copy of the physical world, and make decentralized decisions [1]. Therefore, the smart factory is essentially an industrial CPS (ICPS) [2]. As cloud computing technology provides powerful computing capabilities for Industrie 4.0, the manufacturing process is increasingly refined, and the computing scale of applications has become increasingly large. Industrie 4.0 expands the embedded systems through the information communication technology (ICT) on the basis of automation, and thus forming large-scale heterogeneous distributed embedded systems (e.g., smart grid, factory automation, and process industry) to complex distributed applications. For example, fast Fourier transform (FFT) is a large-scale distributed embedded application used for harmonic analysis in a smart grid and requires complex scientific computation [3]. Such distributed embedded application involves end-to-end computation and communication and can be described as a directed acyclic graph (DAG) [4], [5]. In addition, large-scale distributed embedded applications accordingly require a large amount of processors to support their execution. For example, the square kilometre array (SKA), which is a typical project for FFT application, aims to deploy the world's largest radio-telescope for the next decades [6], [7]. The SKA project uses a many-core platform, specifically the Kalray MPPA (multipurpose processing array) processor integrates 256 cores [6], [7]. In view of so many processor cores, large-scale heterogeneous distributed embedded systems could consume very high hardware costs, which can be understood

Manuscript received August 15, 2018; revised November 1, 2018, December 3, 2018, and January 4, 2019; accepted March 6, 2019. Date of publication March 27, 2019; date of current version January 27, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61972139, Grant 61932010, Grant 61702172, and Grant 61672217, in part by the National Key R&D Program of China under Grant 2017YFB0202901, and Grant 2017YFB0202905, in part by the Natural Science Foundation of Hunan Province under Grant 2018JJ3076, in part by the Open Research Project of the Electronic Information and Control of Fujian University Engineering Research Center, Minjiang University, China under Grant MJXY-KF-EIC1902, and in part by the Fundamental Research Funds for the Central Universities. *(Corresponding author: Guoqi Xie.)*

Guoqi Xie, Wenhong Ma, Hao Peng, and Renfa 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; wenhongma@hnu.edu.cn; hao_peng@hnu.edu.cn; lirenfa@hnu.edu.cn).

Keqin Li is 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, New York 12561 USA (e-mail: lik@newpaltz.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIE.2020.2905815.

Digital Object Identifier 10.1109/TIE.2019.2905815

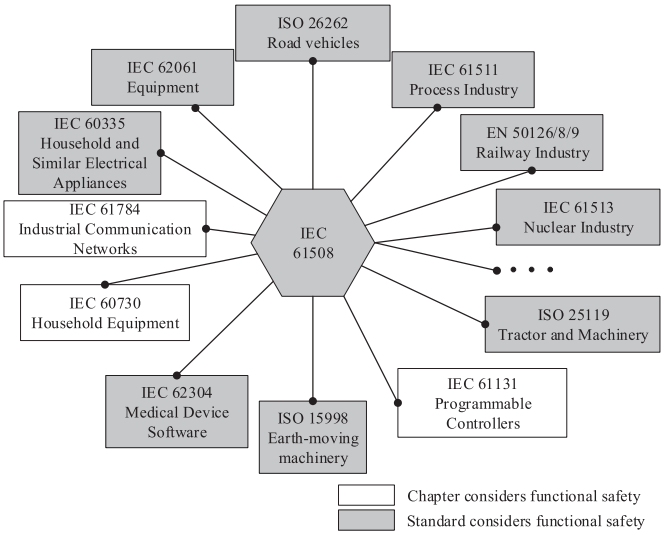


Fig. 1. Main functional safety standards with the areas of applications.

as the price of purchasing processors. The hardware cost of the required processor increases with the continuous growth of the application scale [5]. Currently, the unit price of processors for a typical industrial field bus of the controller area network (CAN) are from $25 to $110 [5], [8]. Most processors are not fully utilized in actual large-scale distributed embedded systems. Some additional processors can be removed to optimize hardware cost as long as such operation does not affect the correct execution of the embedded application. Therefore, optimizing hardware cost by reducing the number of processors is highly desirable [5].

In addition to optimizing hardware cost, an important issue of safety needs to be addressed in distributed embedded systems. In view of Industrie 4.0 and the safety industry as a whole, the costs of work-related accidents and injuries reach 476 billion Euros a year according to new global estimates [9]. That is, industrial safety issue is relatively severe; thus, how to ensure an embedded application's safety operation without physical injury and damage to the health of people is crucial. Functional safety means the freedom from unacceptable risk of physical injury or of damage to the health of people, either directly or indirectly as a result of damage to property or to the environment [10]. A safety-critical embedded application should comply with related functional safety standards for safe operation. Safety issues exist in automotive electronics, industrial safety components, factory automation, process industry, nuclear power instrument control, rail traffic signals, smart grid, and many other safety-related areas. Fig. 1 lists the main functional safety standards in the field of applications. Notice that chapters that consider functional safety (i.e., only partial chapters not all chapters of the standard book describe functionally safety) are also included in Fig. 1.

IEC 61 508, which is titled functional safety of electrical/electronic/programmable electronic safety-related systems (E/E/PES), is a basic functional safety standard applicable to all types of industries [10]. Functional safety standards for special industrial applications include the IEC 61 511 standard for the process industry, the IEC 61 513 standard for nuclear power plants, the IEC 62 061 standard for machinery, and the IEC 61 784 standard for industrial communication, etc. These standards are based on the IEC 61 508 standard and complement IEC 61 508 in specific areas. These standards will change over time; for instance, the latest version of IEC 61 513 is IEC 61 513:2011 released in 2011.

Reliability and response time are critical functional safety properties [5]. Reliability is the probability of survival in a given period of time, whereas response time is the time that the functional unit reacts to a given input. Reliability and real-time are two important measurable functional safety attributes during the design phase. Other properties (e.g., stability [11]) May also be the functional safety properties but are not the concern in this article (i.e., systems are assumed to be stable). Therefore, both reliability requirement and real-time requirement must be simultaneously satisfied towards satisfying functional safety requirement [5], [12]. Reliability requirement means to guarantee that the reliability value reaches a specified reliability goal, whereas real-time requirement means to guarantee response within a specified time constraint. Both requirements must be simultaneously satisfied. Considering high demand or continuous mode in IEC 61 508, the probability of dangerous failure per hour must be larger than or equal to 10^{-9} and less than 10^{-8} in safety integrity level (SIL) 4 [13]. As long as one of the requirements cannot be satisfied, the corresponding personal safety will not be guaranteed [5], [12].

A state-of-the-art method [5] has been proposed to address the problem of hardware cost optimization for a distributed embedded application under its functional safety requirement by proposing enhanced explorative hardware cost optimization (EEHCO) and simplified EEHCO (SEEHCO) algorithms. EEHCO iteratively closes some processors from all opened ones until the application's functional safety requirement cannot be satisfied (i.e., the open-to-close method). SEEHCO simplifies the iteration details to overcome the time inefficiency of EEHCO. EEHCO has powerful cost optimization capability but inferior time efficiency, and vice versa for SEEHCO. Such performance for EEHCO and SEEHCO is unacceptable in large-scale heterogeneous distributed embedded systems (e.g., smart grid), which are configured with many processors (at least hundreds of processors). The scale of a distributed embedded application running in such systems is relatively large (at least thousands of tasks).

Large-scale embedded applications require a large number of processors to participate in large-scale computation; however, the use of EEHCO will cause time inefficiency, thereby resulting in the long lifecycle. Meanwhile, embedded applications also require optimizing the hardware cost considerably for high production profit; however, the SEEHCO is powerless. In view of these problems, we need to present a new method that can simultaneously achieve powerful cost optimization capability and superior time efficiency in large-scale heterogeneous distributed embedded systems.

This article is oriented to large-scale heterogeneous distributed embedded systems. We introduce the concept of price performance and present a price performance-driven hardware cost optimization (PPHCO) algorithm. By price

performance-driven close-to-open and open-to-close methods, PPHCO solves the problem that EEHCO and SEEHCO cannot be compatible with both powerful cost optimization capability and superior time efficiency simultaneously when applied in large-scale heterogeneous distributed embedded systems. The detailed contributions compared with state-of-the-art EEHCO and SEEHCO proposed in [5] are summarized as follows.

1) We present the first price performance-driven hardware cost optimization (PPHCO1) algorithm (Algorithm 1), considering the time inefficiency of the open-to-close EEHCO algorithm. PPHCO1 iteratively selects the processor with the maximum price performance to open until the functional safety requirement is satisfied (i.e., close-to-open).

2) We present the second price performance-driven hardware cost optimization (PPHCO2) algorithm (Algorithm 2), considering that partial PPHCO1-generated processors can be closed while still satisfying the functional safety requirement. PPHCO2 iteratively selects the processor with the minimum price performance to close until the functional safety requirement cannot be satisfied. It also further optimizes the hardware cost on the basis of PPHCO1 without losing time efficiency.

3) We present the PPHCO algorithm (Algorithm 3) by combining PPHCO1 and PPHCO2 algorithms, considering that PPHCO2 can be involved multiple times until a stable value is reached. Through significantly reducing the iteration count, PPHCO overcomes the inferior time efficiency of the open-to-close method. Through adopting union fast functional safety verification (UFFSV) proposed in [12], PPHCO achieves powerful cost optimization capability.

4) Experiments confirm that PPHCO not only achieves stronger cost optimization capability but also has better time efficiency than state-of-the-art EEHCO and SEEHCO algorithms. Particularly, PPHCO just consumes 1112 s with good time efficiency, but EEHCO requires as much as 129 h with poor time efficiency to optimize hardware cost in the case of 320 processors.

II. Related Work

In this section, we review recent related works in cost optimization of DAG applications.

1) Resource cost optimization. Resource cost refers to system resource consumption. Optimizing resource cost under real-time requirement [14] and reliability requirement [15] has been studied in heterogeneous systems. These works either consider the real-time or reliability requirements and do not simultaneously include these two requirements.

2) Development cost optimization. Development cost refers to the workload of developing an application during the development lifecycle. The authors in [16] and [17] optimized the development cost under real-time requirement by presenting genetic algorithm-based and tabu search-based meta-heuristics, respectively. They made an important contribution to the modeling and optimization of development cost but ignored the reliability requirement. The authors in [18] optimized the development cost under reliability requirement; however, the real-time requirement was ignored.

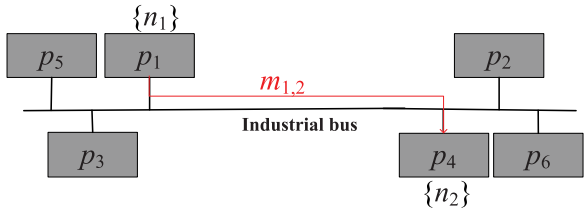


Fig. 2. Architecture of heterogeneous distributed embedded systems.

3) Hardware cost optimization. The authors in [19] optimized the hardware cost under real-time and security requirements by presenting integer linear programming and heuristics, respectively; however, the authors focused on security requirements rather than reliability requirements. The authors in [5] presented the EEHCO and SEEHCO algorithms to optimize the hardware cost under functional safety requirements. As discussed in Section I, neither EEHCO nor SEEHCO can achieve powerful cost optimization capability and superior time efficiency simultaneously in large-scale heterogeneous distributed embedded systems. In the next sections, we will present a new method that can simultaneously achieve these two objectives.

III. Models

A. System Architecture and Application Model

Many processors mounted to the same industrial bus (e.g., LonWorks, Profibu, CAN-FD, and WorldFIP) comprise of large-scale heterogeneous distributed embedded systems, as shown in Fig. 2.

Fig. 2 illustrates the simple execution process of a distributed embedded application. From Fig. 2, processor p_1 receives the data from the sensor to trigger the first task (i.e., the entry task of the application), which is represented as n_1 in DAG. After n_1 is executed completely in p_1, a message $m_{1,2}$ is sent from n_1 to its successor task n_2, which will be executed in processor p_4. $m_{1,2}$ is transmitted in the CAN bus. After triggering a series of tasks with data dependence, the final task (i.e., the exit task) is allocated to a processor and completes the process by sending the performing action to actuators.

On the basis of the architecture and execution process analysis, we let $P = \{p_1, p_2, \ldots, p_{|P|}\}$ represent a set of heterogeneous processors in heterogeneous distributed embedded systems, where $|P|$ is the size of set P. For any set X, we use $|X|$ to denote its size. A motivational example of distributed embedded application is shown in Fig. 3, and the DAG parameters G are explained as follows.

N represents a set of tasks in G, and $n_i \in N$ represents the ith task of G. Fig. 3 shows 10 tasks for the distributed embedded application. From the figure, n_1 is the entry task, and n_{10} is the exit task. Data dependence relationships exist between the tasks. For example, after n_1 is executed completely, messages $m_{1,2}$, $m_{1,3}$, $m_{1,4}$, $m_{1,5}$, and $m_{1,6}$ (denoted in edges) must be sent from n_1 to its successor tasks n_2, n_3, n_4, n_5, n_6, respectively. The weight value of each message represents the worst case response time (WCRT) of communication between tasks. For example, $c_{1,2}$ is the communication time from n_1 to n_2 in the

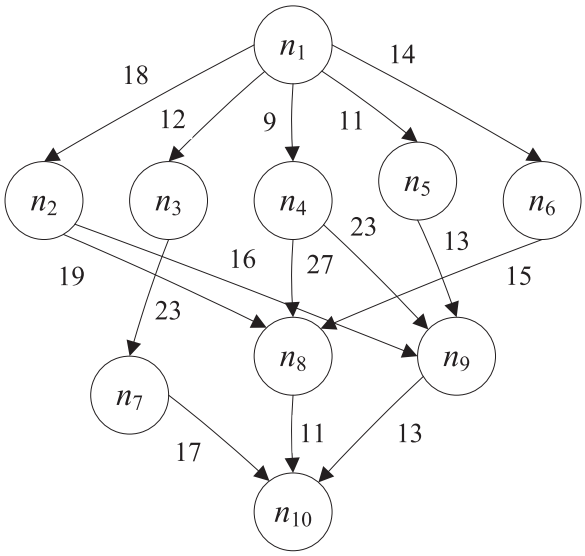


Fig. 3. Motivational distributed embedded application.

TABLE I
WCETs OF TASKS IN DIFFERENT PROCESSORS

Task/processor	p_1	p_2	p_3	p_4
n_1	16	14	9	11
n_2	19	13	18	9
n_3	13	11	19	9
n_4	8	13	17	11
n_5	13	12	10	17
n_6	16	13	9	18
n_7	15	7	11	13
n_8	11	5	14	15
n_9	12	17	19	15
n_{10}	7	21	16	9

worst case. Using the worst case ensures the real-time communication of the distributed embedded application from a safety perspective. Similarly, worst case execution time (WCET) is the execution time of a task in the worst case for safety design [20]. A task has different WCETs values in different processors due to the heterogeneity of processors. Table I presents the WCET matrix $|N| \times |P|$ of tasks in different processors (p_1, p_2, p_3, and p_4) of the motivational distributed embedded application. For example, the weight 14 of n_1 and p_2 in Table I represents the WCET of n_1 in p_2, denoted by $w_{1,2} = 14$. Further details about the application model can be found in [5].

We consider nonpreemptive scheduling for processors that are consistent with the CAN bus for simplicity. The motivational distributed embedded application in Fig. 3 is used to explain the proposed algorithms in the study. For simplicity, all the units of all parameters are ignored in the motivational application.

B. Hardware Cost Model

Heterogeneous processors are used in this study; thus, all processors have individual unit prices. Hence, let $\{price_1, price_2, \ldots, price_{|P|}\}$ represent the set of unit prices of heterogeneous processors. Table II shows that the example of the hardware costs of four processors p_1, p_2, p_3, and p_4 are

TABLE II
PROCESSOR PARAMETERS OF THE MOTIVATIONAL APPLICATION

Parameter/processor	p_1	p_2	p_3	p_4
$price_k$	10	20	30	40
λ_k	0.0006	0.0005	0.0002	0.0004

10, 20, 30, and 40, respectively. λ_k will be explained in the next section.

The hardware cost of the application is the sum of those of all opened processors and is calculated by

$$HC(G) = \sum_{p_k \in P_{\text{opened}}} price_k \tag{1}$$

where P_{opened} represents the opened processor set in heterogeneous distributed embedded systems.

C. Reliability Model

Random hardware failures occur unpredictably during the life cycle of a hardware element, and they follow a probability distribution [5], [12], [21], [22]. Transient failures obeying the Poisson distribution has been widely observed in numerous works [5], [12], [21], [22]. Let λ_k be the failure rate of processor p_k. Then, the reliability of n_i executed in p_k is denoted by

$$R(n_i, p_k) = e^{-\lambda_k w_{i,k}} \tag{2}$$

where $w_{i,k}$ represents the WCET of n_i in p_k.

This study only considers the processor failures and ignores communication failures, because the CAN protocol not only provides the differential transmission specification in the physical layer but also provides the subpacket verification rule by using CRC and ACK segments in the data link layer. Such measures can ensure that the failure rate of a CAN packet in a normal environment is 10^{-9} [23], which is considerably lower than a processor failure rate of 10^{-6} [24]. Therefore, the reliability of the distributed embedded application is then calculated by [5], [12], and [22]

$$R(G) = \prod_{n_i \in N} R(n_i, p_{pr(i)}) \tag{3}$$

where $p_{pr(i)}$ represents the assigned processor of n_i.

D. Problem Formulation

Consider an end-to-end industrial application with functional safety requirement in heterogeneous distributed embedded systems. Then, similar to [5], the problem to be addressed in this study is to reduce the hardware cost of the application without violating its functional safety requirement. The formal description is to minimize the hardware cost; that is

$$HC(G) = \sum_{p_k \in P_{\text{active}}} price_k$$

under the functional safety requirement of $<R_{\text{req}}(G), RT_{\text{req}}(G)>$, that is

$$R(G) \geqslant R_{\text{req}}(G) \tag{4}$$

TABLE III
EEHCO-GENERATED RESULTS WHEN ONLY ONE PROCESSOR IS ASSUMED TO BE REMOVED WHILE THE OTHER PROCESSORS ARE OPENED IN THE FIRST ITERATION

Removing processor	Opened processor set P_{opened}	Response time $RT(G)$	Reliability $R(G)$	Satisfying functional safety requirement?	Hardware cost $HC(G)$
p_1	$\{p_2, p_3, p_4\}$	94	0.971319	Yes	90
p_2	$\{p_1, p_3, p_4\}$	95	0.968894	Yes	80
p_3	$\{p_1, p_2, p_4\}$	84	0.956858	Yes	70
p_4	$\{p_1, p_2, p_3\}$	97	0.964544	Yes	60

TABLE IV
EEHCO-GENERATED SCHEDULE RESULTS WHEN EACH PROCESSOR IS IN THE SLEEP STATE IN THE SECOND ITERATION

Removing processor	Opened processor set P_{opened}	Response time $RT(G)$	Reliability $R(G)$	Satisfying functional safety requirement?	Hardware cost $HC(G)$
p_1	$\{p_2, p_3\}$	96	0.954946	Yes	50
p_2	$\{p_1, p_3\}$	98	0.960789	No	-
p_3	$\{p_1, p_2\}$	94	0.947716	No	-

and

$$RT(G) = AFT(n_{\text{exit}}) \leqslant RT_{\text{req}}(G). \tag{5}$$

$R_{\text{req}}(G)$ and $RT_{\text{req}}(G)$ represent the reliability and response time requirements, respectively. $AFT(n_{\text{exit}})$ represents the actual finish time of the exit task and the response time of the application G. The problem to be addressed in this study is an NP-hard optimization problem [5], because scheduling tasks with quality of service requirement for optimality in multiprocessors is NP-hard [25].

E. Existing EEHCO and SEEHCO Algorithm

The EEHCO algorithm is the most powerful algorithm for hardware cost optimization thus far [5]. The main idea of EEHCO is that it iteratively removes the processors, without which the minimum hardware cost is generated under the functional safety requirement, from P_{opened} until the application's functional safety requirements cannot be satisfied. Particularly, EEHCO adopts a reliability enhancement technique (RET) to satisfy the functional safety requirement as much as possible. RET aims to maximize reliability under real-time requirement and is essentially equivalent to the second FFSV (FFSV2) in UFFSV [12], whereas the first FFSV (FFSV1) aims to minimize response time under reliability requirement (see Section IV-A for more details about UFFSV).

We assume that the functional safety requirement is < 0.95, 97$>$, that is, the reliability requirement is $R_{\text{req}}(G) = 0.95$ and the real-time requirement is $RT_{\text{req}}(G) = 97$.

1) In the initial state, EEHCO assumes that all processors in heterogeneous distributed embedded systems are opened. In view of the motivational distributed embedded application, the opened processor set is $P_{\text{opened}} = \{p_1, p_2, p_3, p_4\}$.

2) Table III shows EEHCO-generated results when only one processor is assumed to be removed while the other processors are opened in the first iteration. From the table, removing each processor can satisfy the functional safety requirement of the distributed embedded application (for how to obtain the reliability and response time values, please refer to [5]). Removing p_4 will have the minimum hardware cost of 60, followed by p_3, p_2, and p_1; thus, EEHCO really removes p_4 (denoted with red color) in this iteration. Therefore, the opened processor set is correspondingly updated to $P_{\text{opened}} = \{p_1, p_2, p_3\}$.

3) Table IV shows EEHCO-generated results when only one processor is assumed to be removed while the other processors are opened in the second iteration. From Table IV, only removing p_1 can satisfy the functional safety requirement of the distributed embedded application, whereas removing p_2 or p_3 cannot. Therefore, EEHCO removes p_1 (denoted with red color) in this iteration. The opened processor set is correspondingly updated to $P_{\text{opened}} = \{p_2, p_3\}$.

4) Removing p_2 or p_3 from $P_{\text{opened}} = \{p_2, p_3\}$ to execute the application in one processor obviously cannot satisfy the functional safety requirement. Therefore, the final opened processor set is $P_{\text{opened}} = \{p_2, p_3\}$, and the hardware cost is $price_2 + price_3 = 20 + 30 = 50$.

As explained in Section I, the main limitation of EEHCO is time inefficiency. As reported in [5], the time complexity of EEHCO reaches $O(|N|^2 \times |P|^3)$. The experiments show that, for a large-scale distributed embedded application with 1151 tasks executed in 320 processors, EEHCO needs 131 h to obtain valid results. To address this issue, [5] simplified EEHCO and further presented the SEEHCO algorithm.

1) Similar to EEHCO, SEEHCO also assumes that all processors are opened in the initial state, that is, $P_{\text{opened}} = \{p_1, p_2, p_3, p_4\}$.

2) Similar to EEHCO, SEEHCO attempts to remove each processor and compares the results in the first iteration. Removing p_4 will have the minimum hardware cost of 60, followed by p_3, p_2, and p_1; thus, p_4 is removed in this iteration (denoted by the red color in Table III).

3) In the succeeding iterations, SEEHCO no longer attempts to remove each processor and compares the results but only directly uses the removing order in the first iteration. Such a simplification makes the time complexity of SEEHCO be reduced to $O(|N|^2 \times |P|^2)$ as reported in [5]. However, SEEHCO May result in higher hardware cost than EEHCO. For example, the removing order in the first iteration is p_4, p_3, p_2, and p_1, such that the second iteration should remove p_3. Unfortunately, removing p_3 will violate the functional safety requirement of the distributed embedded application. Therefore, SEEHCO-generated final opened processor set is $P_{\text{opened}} = \{p_1, p_2, p_3\}$, and the hardware cost is $price_1 + price_2 + price_3 = 10 + 20 + 30 = 60$, which is higher than that of EEHCO.

As discussed in Section I, although SEEHCO overcomes the time inefficiency of EEHCO, it is powerless in hardware cost optimization. According to our experiments, for a distributed embedded application with 1100 tasks and 256 processors, SEEHCO obtains valid results in 44 min, whereas EEHCO needs 131 h. In summary, neither EEHCO nor SEEHCO can simultaneously achieve powerful cost optimization capability and superior time efficiency. To solve the problems of EEHCO

Algorithm 1: The PPHCO1 Algorithm.

Input: $P = \{p_1, p_2, \ldots, p_{|P|}\}$, G, and $RT_{\text{req}}(G)$, $R_{\text{req}}(G)$
Output: $HC(G)$
1: Let all the processors be closed.
2: Sort all the processors in the closed processor list P_{closed} according to the descending order of the price performance values.
3: Define the opened processor list P_{opened} and its initial value is NULL;
4: **while**(P_{closed} is not null) **do**
5: $p_k \leftarrow P_{\text{closed}}.remove()$;
6: $P_{\text{opened}}.add(p_k)$;
7: Verify whether the application's functional safety requirement $<RT_{\text{req}}(G), R_{\text{req}}(G)>$ can be satisfied in the opened processor list P_{opened} by using the FFSV1 and FFSV2 algorithms;
8: **if**(if FFSV1 or FFSV2 returns true) **then**
9: Calculate the application's hardware cost $HC(G)$ using (1);
10: **break**;
11: **end if**
12: **end while**

and SEEHCO, we present a new method that can simultaneously achieve powerful cost optimization capability and superior time efficiency.

IV. Hardware Cost Optimization by Close-to-Open

A. PPHCO1 Algorithm

Contrary to EEHCO's opening of all processors, we let all of them be closed in advance. In other words, the hardware cost optimization is a close-to-open method rather than an open-to-close method. We initially provide the algorithm description (Algorithm 1), and then explain each line of the algorithm by combining the motivational distributed embedded application.

1) In Line 1, PPHCO1 allows all the processors to be closed, that is, $P_{\text{closed}} = \{p_1, p_2, \ldots, p_{|P|}\}$. In other words, no processor is provided for the application's execution in the initial state.

2) In Line 2, PPHCO1 sorts all the processors in the list P_{closed} according to the descending order of the price performance values to provide an opening order of all the processors. The price performance is defined as follows.

Definition 1(**Price performance**). The price performance of the processor refers to the reliability performance divided by the unit price (i.e., an inverse ratio), which can be expressed as

$$PP(p_k) = \frac{e^{-\lambda_k}}{price_k}. \tag{6}$$

In economics and engineering, price performance refers to a product's capability to deliver performance of any sort for its price. Generally, products with a high price performance is desirable [26]. In other words, the processors' opening order is: the higher the price performance, the higher the priority. Table V shows the price performance values of four processors, from

TABLE V
Price Performance Values of Four Processors

Parameter/processor	p_1	p_2	p_3	p_4
$PP(p_k)$	0.100060	0.050025	0.03334	0.02501

which we can obtain the processors' opening order, that is, p_1, p_2, p_3, and p_4.

3) In Line 3, PPHCO1 defines the opened processor list P_{opened}, and its initial value is null. In other words, P_{opened} is ready to load the processors that will be opened.

4) In Lines 4–12, PPHCO1 iteratively selects the processor with the maximum price performance to open until the functional safety requirement is satisfied. The details in the loop are as follows.

a) In Line 5, PPHCO1 takes out the processor with the maximum price performance from P_{closed} and adds this processor into P_{opened} in Line 6.

b) In Line 7, PPHCO1 verifies whether the application's functional safety requirement $<R_{\text{req}}(G), RT_{\text{req}}(G)>$ can be satisfied when executed in P_{opened} by UFFSV. UFFSV technique pertains to checking if the application satisfies a safe requirement set (i.e., real-time and reliability requirements) of design specifications [12]. The UFFSV technique has been solved in [12] by proposing the FFSV1 and FFSV2 algorithms. The FFSV1 algorithm solves the problem of minimizing the response time under reliability requirement, and the verification result can be assessed by comparing the obtained response time with the given real-time requirement. The FFSV2 algorithm solves the problem of maximizing the reliability under real-time requirement, and the verification result can be assessed by comparing the obtained reliability with the given reliability requirement. As long as either verification algorithm returns true, the verification returns true; otherwise, returns false. In this study, we directly use FFSV1 and FFSV2 together to verify the functional safety requirement (see [12] for more details about the FFSV1 and FFSV2 algorithms).

c) In Lines 8–11, if FFSV1 or FFSV2 returns true (i.e., the functional safety requirement is satisfied), then we can obtain a valid hardware cost using (1) in Line 9. When the functional safety requirement is satisfied, the iteration process (i.e., loop) can be stopped (i.e., break in Line 10).

B. Iteration Process of PPHCO1

In view of the motivational distributed embedded application, we execute the iteration process as follows.

1) We initially open processor p_1, which has the highest price performance. In this case, only one processor is used to execute the application. The response time and reliability values are 130 and 0.924 964, respectively, as shown in the first and second lines of Table VII. Obviously, neither FFSV1 nor FFSV2 enables the verification to be passed.

2) Subsequently, we open processor p_2, which has the second highest price performance. In this case, two processors p_1 and p_2 are used to execute the application. The reliability value

TABLE VI
PERFORMANCE ANALYSIS OF THE PROPOSED PPHCO1, PPHCO2, AND PPHCO ALGORITHMS

Algorithm	Time complexity	Main advantages
PPHCO1	The time complexity of PPHCO1 is $O(\|N\|^2 \times \|P\|^2)$ analyzed as below. (1) As reported in Ref. [12], the time complexity of FFSV1 and FFSV2 is $O(\|N\|^2 \times \|P\|)$. (2) PPHCO1 invokes FFSV1 and FFSV2 in the loop (Lines 3–17 in Algorithm 1), and that the maximum loop count is $\|P\|$. As $O(\|N\|^2 \times \|P\|^2)$ has less $\|P\|$ time complexity than EEHCO, it may adapt to the large-scale heterogeneous distributed embedded systems with a large number of processors.	PPHCO1 is a close-to-open method and its main advantages are below. (1) PPHCO1 quickly determines the processors that can be opened based on the price performance of each processor. (2) PPHCO1 just traverse partial processors satisfying functional safety requirement, whereas EEHCO must traverse all processors. (3) PPHCO1 reduces the exploration space for further hardware cost optimization given that unopened processors are no longer a concern.
PPHCO2	The time complexity of PPHCO2 is $O(\|N\|^2 \times \|P\|^2)$ analyzed as below. (1) PPHCO2 optimizes the hardware cost by removing processors from P_{opened}; thus, the maximum loop count of the while loop (Lines 4–10 in Algorithm 2) is $\|P\|$. (2) PPHCO2 invokes FFSV1 and FFSV2 in the while loop; thus, the time complexity of PPHCO2 is $O(\|N\|^2 \times \|P\|^2)$, which is similar to that of PPHCO1.	PPHCO2 is a open-to-close method and its main advantages are below. (1) PPHCO2 quickly determines the processors that can be closed based on the price performance of each processor. (2) PPHCO2 avoids the situation wherein EEHCO must attempt to close each processor and schedule other processors. (3) PPHCO2 always removes partial processors from the opened processor set to optimize the hardware cost.
PPHCO	The time complexity of PPHCO is $O(\|N\|^2 \times \|P\|^3)$ because it invokes PPHCO2 in the while loop. Although PPHCO reaches the same time complexity as EEHCO, its exploration space is continuously shrinking, thereby making its actual computational time is even shorter than that of time-efficient SEEHCO. Further details about the time efficiency can be found in Section VI.	(1) PPHCO can optimize the hardware cost to the extreme and reach a stable state. (2) PPHCO still has superior time efficiency compared with SEEHCO and EEHCO by observing experiments. (3) PPHCO achieves powerful cost optimization capability and superior time efficiency.

TABLE VII
ITERATION PROCESS USING PPHCO1 FOR THE MOTIVATIONAL APPLICATION

Opened processors P_{opened}	UFFSV	Response time $RT(G)$	Reliability $R(G)$	Satisfying functional safety requirement?	Hardware cost $HC(G)$
$\{p_1\}$	FFSV1	130	0.924964	No	10
	FFSV2	130	0.924964	No	10
$\{p_1, p_2\}$	FFSV1	94	0.947716	No	30
	FFSV2	94	0.947716	No	30
$\{p_1, p_2, p_3\}$	FFSV1	80	0.956380	Yes	60
	FFSV2	97	0.964544	Yes	60

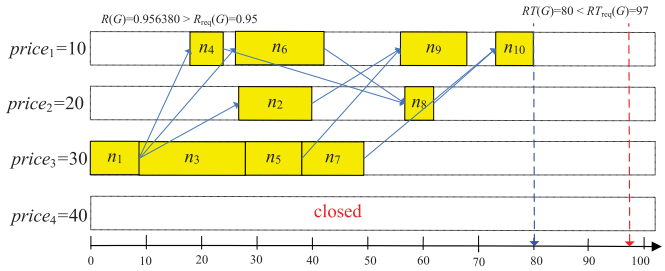


Fig. 4. Task mapping of the motivational application in processors $\{p_1, p_2, p_3\}$ by using FFSV1.

(0.947 716) is less than the reliability value of 0.95; thus, neither FFSV1 nor FFSV2 enables the verification to be passed.

3) We continue to open the processor p_3, which has the third highest price performance. In this case, three processors p_1, p_2, p_3 are used to execute the application. By invoking FFSV1, we can obtain the response time of 80 (less than 95) and reliability of 0.956 380 (larger than 0.95). By invoking FFSV2, we can obtain the response time of 97 (equal to real-time requirement of 97) and reliability of 0.964 544 (larger than reliability requirement of 0.95). That is, both FFSV1 and FFSV2 enable the UFFSV to be passed in this case (denoted with red color), and the iteration process is stopped without opening processor p_4, which has the lowest price performance.

Fig. 4 shows the task mapping of the motivational application in processors $\{p_1, p_2, p_3\}$ by using FFSV1. The arrows in the figure represent the communication between tasks if they have immediate data dependency relationships. For example, the arrow between n_1 and n_4 represents the communication between n_1 and n_4, and the WCRT is 9 (i.e., $c_{1,4} = 9$). The WCRT in this study is a theoretical WCRT upper bound rather than the actual communication time. In other words, the WCRT is pessimistic but safe and is useful for safety design. When the reliability satisfies $R(G) = 0.956\ 380 > R_{\text{req}}(G) = 0.95$ and the response time satisfies $RT(G) = 80 < RT_{\text{req}}(G) = 97$ (i.e., the functional safety requirement is satisfied), the obtained hardware cost is valid and that is $HC(G) = price_1 + price_2 + price_3 = 10 + 20 + 30 = 60$.

TABLE VIII
ITERATION PROCESS USING PPHCO2 FOR THE MOTIVATIONAL APPLICATION

Test processors ($P_{\text{opened}} - p_k$)	UFFSV	Response time $RT(G)$	Reliability $R(G)$	Satisfying functional safety requirement?	Opened processors P_{opened}	Hardware cost $HC(G)$
$\{p_1, p_2, p_3\}$	FFSV1	94	0.947716	No	$\{p_1, p_2, p_3\}$	60
- p_3	FFSV2	94	0.947716	No	$\{p_1, p_2, p_3\}$	60
$\{p_1, p_2, p_3\}$	FFSV1	94	0.953897	Yes	$\{p_1, p_3\}$	40
- p_2	FFSV2	98	0.960789	No	$\{p_1, p_2, p_3\}$	60
$\{p_1, p_3\}$	FFSV1	142	0.971999	No	$\{p_1, p_3\}$	40
- p_1	FFSV2	142	0.971999	No	$\{p_1, p_3\}$	40

The performance analysis (including time complexity and main advantages) of the proposed PPHCO1 algorithm is shown in Table VI.

V. HARDWARE COST OPTIMIZATION BY OPEN-TO-CLOSE

A. PPHCO2 Algorithm

Similar to PPHCO1, we first provide the algorithm description (Algorithm 2) and then explain each line of the algorithm by combining the motivational distributed embedded application.

1) In Line 1, PPHCO2 sorts all the processors in the list P_{opened} according to the ascending order of the price performance values. Contrary to PPHCO1, the processors' opening order for PPHCO2 is: The lower the price performance, the higher the priority.

Algorithm 2: PPHCO2 Algorithm.

Input: $P = \{p_1, p_2, \ldots, p_{|P|}\}$, G, and $RT_{\text{req}}(G)$, $R_{\text{req}}(G)$, PPHCO1-generated results
Output: $HC(G)$

1: Sort all the processors in the list P_{opened} according to the ascending order of the price performance values;
2: **while**(there is a processor that has not been obtained from P_{opened}) **do**
3: $p_k \leftarrow P_{\text{opened}}.get()$;
4: Verify whether the application's functional safety requirement $<RT_{\text{req}}(G), R_{\text{req}}(G)>$ can be satisfied in the processor set $(P_{\text{opened}} - p_k)$ by using the FFSV1 and FFSV2 algorithms;
5: **if**(FFSV1 or FFSV2 returns true) **then**
6: $P_{\text{opened}}.remove(p_k)$;
7: **end if**
8: **end while**
9: Calculate the application's new hardware cost $HC(G)$ using (1);

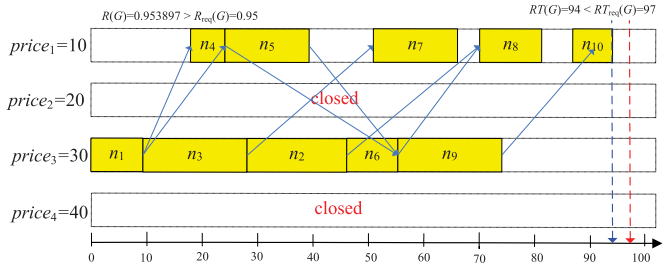


Fig. 5. Task mapping of the motivational application in processors $\{p_1, p_3\}$ by using FFSV1.

TABLE IX
ITERATION PROCESS BY FURTHER USING PPHCO2 FOR THE MOTIVATIONAL APPLICATION

Test processors $P_{\text{opened}} - p_k$	Response time $RT(G)$	Reliability $R(G)$	Satisfying functional safety requirement?	Opened processors P_{opened}	Hardware cost $HC(G)$
$\{p_1, p_3\}$ - p_3	130	0.924964	No	$\{p_1, p_3\}$	40
$\{p_1, p_3\}$ - p_1	142	0.958295	No	$\{p_1, p_3\}$	40

2) Similar to the PPHCO1 algorithm (Algorithm 1), only one while loop exists for the PPHCO2 algorithm (Algorithm 2)

a) In Line 3, PPHCO2 obtains processor p_k, which has the lowest price performance value from P_{opened}. p_k remains in P_{opened} and has not been removed.
b) In Line 4, PPHCO2 verifies whether the application's functional safety requirement $<R_{\text{req}}(G), RT_{\text{req}}(G)>$ can be passed in the opened processor list $(P_{\text{opened}} - p_k)$ by using the FFSV1 and FFSV2 algorithms.
c) In Lines 5–7, if FFSV1 or FFSV2 returns true, then PPHCO2 removes p_k from P_{opened}.

3) After all the processors are obtained from P_{opened} (Lines 2–8), we then calculate the application's hardware cost $HC(G)$ using (1).

B. Iteration Process of PPHCO2

We perform the following iteration process considering the motivational application.

1) We first obtain processor p_3, which has the lowest price performance value from $P_{\text{opened}} = \{p_1, p_2, p_3\}$. In this case, p_1 and p_2 are used to execute the application. The response time and reliability values are 94 and 0.947 716, respectively, as shown in the first and second lines of Table VIII. Therefore, the functional safety requirement $< 0.95, 97>$ cannot be satisfied, and p_3 cannot be removed from P_{opened}.

2) We then obtain the processor p_2, which has the second lowest price performance value from $P_{\text{opened}} = \{p_1, p_2, p_3\}$. In this case, p_1 and p_3 are used to execute the application. By invoking FFSV1, the response time and reliability values are 94 and 0.953 897, respectively, as shown in the third line of Table VIII. Therefore, the functional safety requirement $< 0.95, 97>$ can be satisfied, and p_2 can be removed from P_{opened}, such that P_{opened} is updated to $P_{\text{opened}} = \{p_1, p_3\}$ (denoted with red color). Notice that the response time and reliability values are 98 and 0.960 789, respectively, by invoking FFSV2, and the functional safety requirement $< 0.95, 97>$ cannot be satisfied in this case. The above results show that the use of UFFSV with FFSV1 and FFSV2 is more likely to satisfy the functional safety requirement than EEHCO with only FFSV2, thus it is possible to optimize hardware cost further.

3) We continue to obtain processor p_1 from $P_{\text{opened}} = \{p_1, p_3\}$, and the result shows that p_1 cannot be removed from P_{opened}.

Fig. 5 shows the task mapping of the motivational application in processors $\{p_1, p_3\}$ by using FFSV1. From Fig. 5, when the reliability satisfies $R(G) = 0.953\ 897 > R_{\text{req}}(G) = 0.95$ and the response time satisfies $RT(G) = 94 < RT_{\text{req}}(G) = 97$ (i.e., the functional safety requirement is satisfied), the obtained hardware cost is valid and that is $HC(G) = price_1 + price_3 = 10 + 30 = 40$.

The performance analysis (including time complexity and main advantages) of the proposed PPHCO2 algorithm is shown in Table VI.

C. PPHCO Algorithm

Our hardware cost optimization method has not ended yet after using PPHCO2. In view of the motivational distributed embedded application, the final opened processor set is $P_{\text{opened}} = \{p_1, p_3\}$ after using PPHCO2. However, we can further use PPHCO2 to remove possible processors from $P_{\text{opened}} = \{p_1, p_3\}$ without violating the functional safety requirement of the distributed embedded application. Table IX shows the iteration process in which PPHCO2 is further used for the motivational application.

PPHCO2 cannot satisfy the functional safety requirement in the case of $\{p_1, p_2, p_3\}$ - $p_3 = \{p_1, p_2\}$ (Table VIII). However, it May be occurred in PPHCO2 that it satisfies the functional safety requirement in $\{p_1, p_3\}$ - $p_3 = \{p_1\}$ (Table IX). The processor number is small for the motivational application; thus, Table IX can not indicate that the hardware cost is further optimized. However, this possibility exists for other applications, because

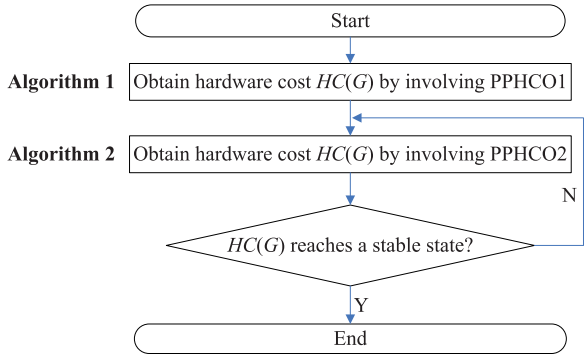


Fig. 6. Flowchart of the presented PPHCO algorithm.

Algorithm 3: The PPHCO Algorithm.

Input: $P = \{p_1, p_2, \ldots, p_{|P|}\}$, G, and $RT_{\text{req}}(G)$, $R_{\text{req}}(G)$
Output: $HC(G)$

1: The initial hardware cost $HC(G)$ the distributed embedded application is generated by invoking PPHCO1;
2: **while**(true) **do**
3: Obtain the new hardware cost $HC_{\text{new}}(G)$ of the distributed embedded application by invoking PPHCO2;
4: **if**($HC_{\text{new}}(G) < HC(G)$) **then**
5: $HC(G) \leftarrow HC_{\text{new}}(G)$;
6: **else**
7: **break**; //$HC_{\text{new}}(G) == HC(G)$
8: **end if**
9: **end while**

a task has different WCETs in different processors due to the heterogeneity of processors; this characteristic could occur in the situation that using fewer opened processors does not mean worse response time and reliability compared with using more opened processors.

The PPHCO algorithm is presented based on the aforementioned analysis, as shown in Algorithm 3. The flowchart of PPHCO has been shown in Fig. 6.

1) Line 1 shows that the initial hardware cost $HC(G)$ is generated by PPHCO1.

2) In the while loop (Lines 2–9), PPHCO obtains the new hardware cost $HC_{\text{new}}(G)$ of the distributed embedded application by invoking PPHCO2.

a) If the new hardware cost $HC_{\text{new}}(G)$ is less than $HC(G)$, then update $HC(G)$ to $HC_{\text{new}}(G)$ (i.e., $HC(G) \leftarrow HC_{\text{new}}(G)$). In other words, the hardware cost reduction is achieved through $HC(G) \leftarrow HC_{\text{new}}(G)$ (Lines 4–6). A processor is removed from the opened processor set in invoking PPHCO2 (Line 3) if the result is $HC_{\text{new}}(G) < HC(G)$ (Line 4).

b) If the new hardware cost $HC_{\text{new}}(G)$ is equal to $HC(G)$, then the while loop is stopped. In other words, the hardware cost reaches a stable value.

The performance analysis (including time complexity and main advantages) of the proposed PPHCO algorithm is shown in Table VI.

VI. Experiments

A. Experimental Conditions and Instructions

The algorithms compared with PPHCO1, PPHCO2, and PPHCO are state-of-the-art iterative hardware cost optimization (IHCO) [27], EEHCO [5], and SEEHCO [5]. IHCO iteratively removes many processors with a high hardware cost until the application's functional safety requirement cannot be satisfied [5]. EEHCO and SEEHCO have been explained in detail in Section III-E.

To implement fair comparison, we also use the equal application and processor parameters to [5] as a test bed (2.6 GHz Intel CPU and 4 GB memory) to perform experiments, that is, 10 ms $\leqslant w_{i,k} \leqslant$ 100 ms, 10 ms $\leqslant c_{i,j} \leqslant$ 100 ms, 0.000 001/ms $\leqslant \lambda_k \leqslant$ 0.000 009/ms, \$25 $\leqslant price_k \leqslant$ \$110. Similar to [5], we also use the hardware cost $HC(G)$, the number of opened processors $|P_{\text{opened}}|$ for applications, and computational time of the algorithm as the metrics.

Many distributed embedded applications can be represented by DAGs. In this study, we use two typical distributed embedded applications, namely, FFT and Gaussian elimination (GE). FFT implements the transform between time (or space) and frequency and can be used for harmonic analysis in a smart grid. GE solves linear equations and can be used for topology analysis in a smart grid. These two applications have also been implemented in embedded systems [7], [28]. We use these two applications because they have well-defined structures; particularly, FFT is a high-parallelism application, whereas GE is a low-parallelism application (refer to [29] for more details about their structures). We can fully reflect the advantages and characteristics of the proposed algorithm by comparing two representative applications with good structure and opposite parallelism with the same scale.

All the algorithms involve open and close operation of processors; thus, we analyze the performance of these algorithms under different numbers of processors. Similar to [5], we set the real-time and reliability requirements as $RT_{\text{req}}(G) = LB(G)$ and $R_{\text{req}}(G) = R_{\text{heft}}(G)$, respectively. $LB(G)$ and $R_{\text{heft}}(G)$ represent lower bound and reliability values generated by the heterogeneous earliest finish time (HEFT) algorithm. The lower bound refers to the minimum response time of an application without any constraint. Scheduling tasks with minimum response time in multiprocessors is known to be an NP-hard optimization problem; thus, the HEFT algorithm presented in [29] is a well-studied list scheduling algorithm and has been used to obtain the approximate lower bound in numerous works [5], [12] (Please refer to [29] for further details of how to obtain the lower bound).

The response time and reliability values are no longer provided because all the five algorithms aim to optimize the hardware cost under the functional safety requirements. Therefore, we directly observe the generated hardware costs, number of opened processors, and computational time values.

TABLE X
COMPUTATION TIME OF ALGORITHMS WHEN EXECUTING FFT APPLICATION FOR DIFFERENT NUMBERS OF PROCESSORS

Algorithm	$\|P\|$ = 64	$\|P\|$ = 128	$\|P\|$ = 192	$\|P\|$ = 256	$\|P\|$ = 320
IHCO [27]	2 s	3 s	8 s	269 s	21 s
EEHCO [5]	1238 s	12199 s (3.4 h)	56475 s (15.7 h)	144135 s (40 h)	471763 s (131 h)
SEEHCO [5]	48 s	221 s	690 s	1433 s	2635 s
PPHCO1	25 s	31 s	62 s	88 s	100 s
PPHCO2	27 s	32 s	50 s	94 s	85 s
PPHCO	82 s	96 s	144 s	340 s	232 s

TABLE XI
ITERATION COUNTS OF ALGORITHMS WHEN EXECUTING FFT APPLICATION FOR DIFFERENT NUMBERS OF PROCESSORS

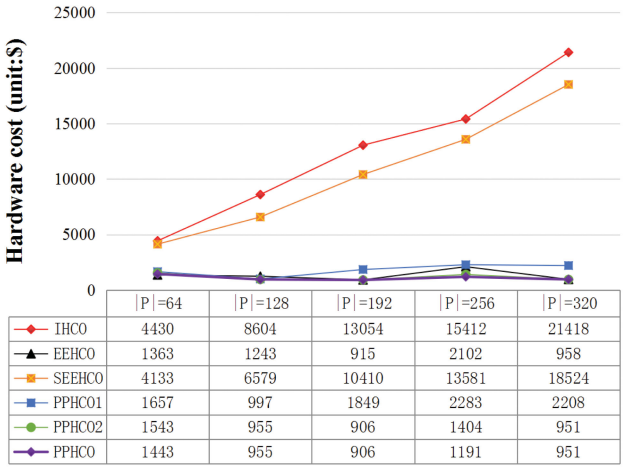
Algorithm	$\|P\|$ = 64	$\|P\|$ = 128	$\|P\|$ = 192	$\|P\|$ = 256	$\|P\|$ = 320
IHCO [27]	2	1	5	57	1
EEHCO [5]	1645	7661	18093	31356	50799
SEEHCO [5]	67	148	218	291	347
PPHCO1	35	30	53	66	66
PPHCO2	2	1	24	23	34
PPHCO	37	31	77	112	100

B. Experimental Details and Analyses

Experiment 1. This experiment observes the hardware costs, number of opened processors of the high-parallelism FFT application with 1151 tasks, and computational time values of the five algorithms under different numbers of processors. The processor number is changed from 64 to 320 with 64 increments.

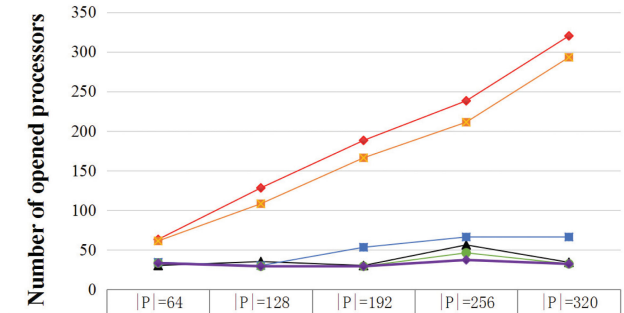
Table X shows the computational time of all the algorithms. PPHCO's computational time includes the computational time of invoking PPHCO1 once and PPHCO2 multiple times. In comparison with other algorithms, EEHCO is quite time-consuming, especially with more processors. For example, when the task number is 256 and 320, the computational time values of EEHCO reach 40 and 131 h, respectively. IHCO has the best time efficiency in most cases; however, IHCO's cost optimization capability is extremely limited. A promising result is that PPHCO yields better time efficiency than SEEHCO, with the exception of $|P| = 64$. PPHCO only requires 1/2003 time of SEEHCO to get the result when $|P| = 320$, respectively. Although PPHCO has high time complexity, its actual time efficiency is better than that of SEEHCO.

To find out why PPHCO has superior time efficiency, we list the iteration counts of algorithms when executing FFT applications for different numbers of processors, as shown in Table XI. The results show that the iteration counts of PPHCO are much less than those of EEHCO. For example, when $|P| = 320$, PPHCO just needs 100 iterations to obtain the optimized hardware cost of \$951, whereas EEHCO and SEEHCO needs 50799 and 347 iterations to obtain the optimized cost of \$958 and \$18 524, respectively. The hardware costs can be found in Fig. 7(a). Notice that the sum of iteration counts of PPHCO1 and PPHCO2 is not always equal to the iteration count of PPHCO in each column because PPHCO2 May be invoked multiple times. Overall, the potential reason for the superior time efficiency of PPHCO is that it needs much less iteration counts than EEHCO and SEEHCO. Further analysis, the essential reason for low iteration count of PPHCO is adopting



	|P|=64	|P|=128	|P|=192	|P|=256	|P|=320
IHCO	4430	8604	13054	15412	21418
EEHCO	1363	1243	915	2102	958
SEEHCO	4133	6579	10410	13581	18524
PPHCO1	1657	997	1849	2283	2208
PPHCO2	1543	955	906	1404	951
PPHCO	1443	955	906	1191	951

(a)



	|P|=64	|P|=128	|P|=192	|P|=256	|P|=320
IHCO	63	128	188	238	320
EEHCO	30	35	30	56	34
SEEHCO	61	108	166	211	293
PPHCO1	34	30	53	66	66
PPHCO2	33	29	29	46	32
PPHCO	33	29	29	37	32

(b)

Fig. 7. Results of FFT application for different numbers of processors. (a) Hardware cost (unit: \$). (b) Number of opened processors.

the price performance-driven close-to-open and open-to-close methods: 1) PPHCO1 quickly excludes most of the processors driven by price performance (i.e., close-to-open); 2) PPHCO2 quickly continues to exclude some processors from the opened processor driven by price performance (open-to-close); and 3) the iteration count of PPHCO2 is decreasing).

After analyzing the time efficiency, we analyze the hardware optimization capability. The curves in Fig. 7(a) indicate that IHCO and SEEHCO are at a relatively high cost level, whereas EEHCO, PPHCO1, PPHCO2, and PPHCO are at a relatively low cost level. Generally, IHCO constantly produces the highest hardware costs, followed by SEEHCO. With the increasing number of processors, the hardware costs of IHCO and SEEHCO increase correspondingly, whereas EEHCO, PPHCO1, PPHCO2, and PPHCO do not show similar linear growth. In general, the hardware cost difference between EEHCO and PPHCO is not extremely large; however, in the case of a large number of processors, PPHCO has a stronger hardware cost optimization capability than EEHCO; particularly, when $|P| = 256$, the hardware cost of PPHCO is only about half of

TABLE XII
COMPUTATIONAL TIME OF ALGORITHMS WHEN EXECUTING GE APPLICATION FOR DIFFERENT NUMBERS OF PROCESSORS

Algorithm	$\|P\|$ = 64	$\|P\|$ = 128	$\|P\|$ = 192	$\|P\|$ = 256	$\|P\|$ = 320
IHCO [27]	2 s	3 s	11 s	15 s	29 s
EEHCO [5]	1727 s	14164 s (3.9 h)	59714 s (16.6 h)	166115 s (46 h)	501123 s (129 h)
SEEHCO [5]	21 s	82 s	108 s	1775 s	5536 s
PPHCO1	33 s	26 s	31 s	200 s	57 s
PPHCO2	37 s	32 s	33 s	209 s	69 s
PPHCO	100 s	87 s	80 s	430 s	148 s

EEHCO and is about 1/10 of SEEHCO. These results show that PPHCO performs better than EEHCO in hardware cost optimization capability. The essential reason for the stronger hardware cost optimization capability of PPHCO than EEHCO and SEEHCO is that PPHCO not only uses FFSV1 but also FFSV2 to increase the likelihood to satisfy the functional safety requirement, whereas EEHCO and SEEHCO only use FFSV2, such that PPHCO is more thorough than EEHCO and SEEHCO in optimizing hardware cost.

In addition to hardware cost optimization capabilities, we aim to determine how many processors are opened to analyze the potential reasons. The trend of the curves in Fig. 7(a) and (b) are generally consistent. The percentage of processors opened by IHCO and SEEHCO exceeded 92% and 82%, respectively. IHCO cannot even close any processor. The number of opened processors using EEHCO is between 30–56, whereas that using PPHCO is between 27–39 in all cases. Hence, the number of opened processors using PPHCO is unaffected by the total number of processors. This result is due to the fact that PPHCO initially determines the processors that must be opened quickly on the basis of the maximum price performance of each processor by PPHCO1 and then quickly determines the processors that must be closed on the basis of the minimum price performance of each processor. Such operations thus avoid attempting to close each processor and schedule other processors.

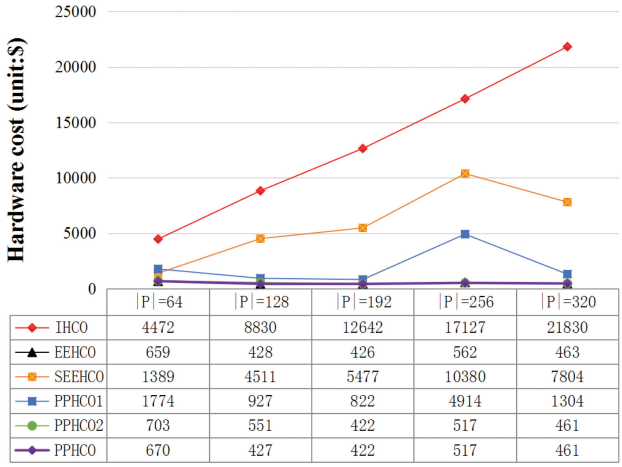
These results show that PPHCO is superior to SEEHCO and EECHCO in terms of time efficiency and hardware cost optimization capability. PPHCO does not aim to make a trade-off between time efficiency of SEEHCO and hardware cost optimization capability of EEHCO but to achieve better cost optimization capability and time efficiency than SEEHCO and EEHCO simultaneously.

Experiment 2. This experiment observes the hardware costs, number of opened processors of the low-parallelism GE application with 1175 tasks and the computational time values of the five algorithms under different numbers of processors. Similar to Experiment 1, the processor number is changed from 64 to 320 with 64 increments.

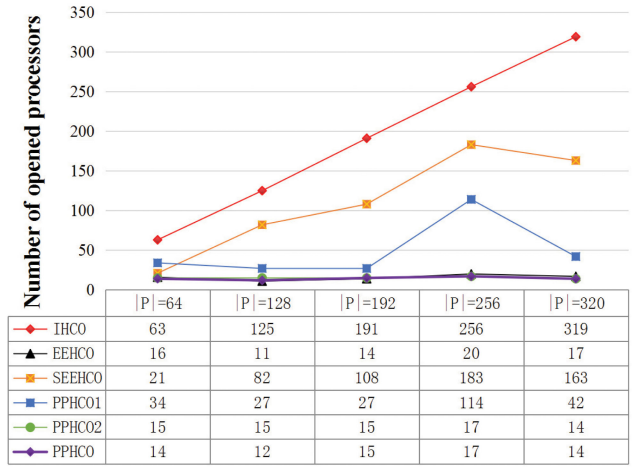
Similar to Table X for high-parallelism FFT application, Table XII for low-parallelism GE application shows that PPHCO is the more time-efficient algorithm than SEEHCO in the case of a large number of processors. For example, when $|P|$ =320, PPHCO only needs 1/37 time of that of SEEHCO to obtain the results; meanwhile, EEHCO is relatively time-consuming and needs 46 and 129 h to obtain the results.

TABLE XIII
ITERATION COUNTS OF ALGORITHMS WHEN EXECUTING GE APPLICATION FOR DIFFERENT NUMBERS OF PROCESSORS

Algorithm	$\|P\|$ = 64	$\|P\|$ = 128	$\|P\|$ = 192	$\|P\|$ = 256	$\|P\|$ = 320
IHCO [27]	2	4	2	1	2
EEHCO [5]	1960	8201	18437	32706	51224
SEEHCO [5]	107	174	276	329	350
PPHCO1	34	27	27	114	42
PPHCO2	19	12	12	97	28
PPHCO	56	45	39	211	70



(a)



(b)

Fig. 8. Results of GE application for different numbers of processors. (a) Hardware cost (unit: $). (b) Number of opened processors.

Table XIII lists iteration counts of algorithms when executing GE applications for different numbers of processors. Similar to the results in Table XI for the FFT application, Table XIII also shows that PPHCO needs much less iteration counts than EEHCO to obtain superior time efficiency.

The comparison of Figs. 7(a) and 8(a) indicates that EEHCO and PPHCO can generate lower hardware costs for low-parallelism GE application than for high-parallelism FFT application. Similar to Fig. 7(a), Fig. 8(a) shows that the hardware cost difference between EEHCO and PPHCO is small. When the

processor numbers are 128, 196, 256, and 320, PPHCO still outperforms EHECO. Although the hardware cost optimization advantage of PPHCO is weaker than EEHCO for low-parallelism applications, the results show that PPHCO is still powerful in hardware optimization capability for low-parallelism GE applications.

Furthermore, the trend of the curves in Fig. 8(a) and (b) is generally consistent. The comparison of Fig. 8(a) and (b) shows that EEHCO and PPHCO needs fewer number of processors for low-parallelism GE applications than that for high-parallelism FFT application. For example, the number of opened processors using EEHCO is merely between 11 and 20, whereas that using PPHCO is merely between 12 and 17.

The above analysis indicates that EEHCO and PPHCO are more powerful in hardware cost optimization capability for low-parallelism application than that for high-parallelism application. In addition, PPHCO still maintains superior time efficiency, whereas EEHCO is quite time-consuming for a large number of processors. Therefore, PPHCO is better than SEEHCO and EEHCO in terms of time efficiency and hardware cost optimization capability in large-scale heterogeneous distributed embedded systems.

VII. Conclusion

In this article, we present a novel hardware cost optimization algorithm called PPHCO, which can optimize the hardware cost to the extreme and reach a stable state driven by price performance. The greatest innovation of PPHCO is that it initially eliminates most of the processors no longer concerned by adopting the close-to-open method, such that it overcomes the time inefficiency of the existing open-to-close method. The greatest contribution of PPHCO is that it does not aim to make a tradeoff between the time efficiency of SEEHCO and the hardware cost optimization capability of EEHCO but simultaneously achieves powerful cost optimization capability and superior time efficiency simultaneously compared with EEHCO and SEEHCO. We summarized the essential reasons for superior time efficiency and powerful hardware cost optimization capability of PPHCO by experiments: 1) PPHCO needs much less iteration counts than EEHCO and SEEHCO to finish the cost optimization quickly by price performance-driven close-to-open and open-to-close methods; and 2) PPHCO uses both FFSV1 and FFSV2 to increase the likelihood of satisfying the functional safety requirement towards thorough hardware cost optimization, whereas EEHCO and SEEHCO only use FFSV2. By using PPHCO, engineering managers can quickly get the hardware cost required for a given design requirement to enable system designers to complete system design. The future work could study the hardware cost optimization for dynamic distributed embedded application.

Acknowledgment

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

References

[1] S. I. Shafiq, C. Sanin, E. Szczerbicki, and C. Toro, "Virtual engineering object/virtual engineering process: A specialized form of cyber physical system for industrie 4.0," *Procedia Comput. Sci.*, vol. 60, pp. 1146–1155, Jan. 2015.
[2] A. W. Colombo, S. Karnouskos, Y. Shi, S. Yin, and O. Kaynak, "Industrial cyber-physical systems," *IEEE Proc. IRE*, vol. 104, no. 5, pp. 899–903, 2016.
[3] B. Zeng, Z. Teng, Y. Cai, S. Guo, and B. Qing, "Harmonic phasor analysis based on improved FFT algorithm," *IEEE Trans. Smart Grid*, vol. 2, no. 1, pp. 51–59, Mar. 2011.
[4] J. Li, G. Xie, K. Li, and Z. Tang, "Enhanced parallel application scheduling algorithm with energy consumption constraint in heterogeneous distributed systems," *J. Circuits Syst. Comput.*, Nov. 2018.
[5] G. Xie, Y. Chen, R. Li, and K. Li, "Hardware cost design optimization for functional safety-critical parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2418–2431, Jun. 2017.
[6] [Online]. Available: http://www.ska.ac.za
[7] 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. Des. Architectures Signal Image Process.*, 2015, pp. 1–7.
[8] [Online]. Available: http://copperhilltech.com/embedded-can-interfaces/
[9] "Work-Related Accidents and Injuries Cost Eur 476 Billion a Year According to New Global Estimates," Apr. 2017. [Online]. Available: https://osha.europa.eu/en/about-eu-osha/press-room/eu-osha-presents-new-figures-costs-poor-workplace-safety-and-health-world
[10] I. E. Commission *et al.*, "Functional Safety of electrical/electronic/programmable Electronic Safety Related Systems," IEC 61508, 2000.
[11] L. Liu, G. Xie, and R. Li, "Synchronization stability analysis of medical cyber-physical cloud system considering multi-closed-loops," *J. Circuits Syst. Comput.*, Nov. 2018.
[12] G. Xie, G. Zeng, Y. Liu, J. Zhou, R. Li, and K. Li, "Fast functional safety verification for distributed automotive applications during early design phase," *IEEE Trans. Ind. Electron.*, vol. 65, no. 5, pp. 4378–4391, May 2018.
[13] [Online]. Available: https://en.wikipedia.org/wiki/IEC_61508
[14] J. Liu *et al.*, "Minimizing system cost with efficient task assignment on heterogeneous multicore processors considering time constraint," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2101–2113, Aug. 2014.
[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, Technical University of DenmarkDanmarks Tekniske Universitet, Department of Informatics and Mathematical ModelingInstitut for Informatik og Matematisk Modellering, 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, p. 50, May 2015.
[18] G. Xie, Y. Chen, Y. Liu, R. Li, and K. Li, "Minimizing development cost with reliability goal for automotive functional safety during design phase," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 196–211, Mar. 2018.
[19] 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.
[20] Z. Gu, C. Wang, M. Zhang, and Z. Wu, "Wcet-aware partial control-flow checking for resource-constrained real-time embedded systems," *IEEE Trans. Ind. Electron.*, vol. 61, no. 10, pp. 5652–5661, Oct. 2014.
[21] G. Xie *et al.*, "Reliability enhancement towards functional safety goal assurance in energy-aware automotive cyber-physical systems," *IEEE Trans. Ind. Informat.*, vol. 14, no. 12, pp. 5447–5462, Dec. 2018.
[22] G. Xie, G. Zeng, R. Li, and K. Li, "Quantitative fault-tolerance for reliable workflows on heterogeneous iaas clouds," *IEEE Trans. Cloud Comput.*, vol. 8, no. 4, pp. 1223–1236, Oct./Dec. 2020, doi: 10.1109/TCC.2017.2780098.
[23] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and using the controller area network communication protocol: Theory and practice*. Newyork, NY, USA: Springer Science and Business Media, 2012.
[24] A. Kazeminia, "Reliability optimization of hardware components and systems topology during early design phase," *J. Amer. Assoc. Pediatr. Ophthalmol. Strabismus*, vol. 18, no. 4, p. e9, 2014.

[25] J. D. Ullman, "Np-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.
[26] P. ratio. [Online]. Available: https://en.wikipedia.org/wiki/Price-performance_ratio
[27] 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.
[28] 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.
[29] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

Guoqi Xie (Member, IEEE) received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2014. He is currently a Professor at Hunan University. He was a Postdoctoral Research Fellow with Nagoya University, Nagoya, Japan.

His current research interests include real-time systems, automotive embedded systems, embedded safety and security.

Dr. Xie received the 2018 IEEE TCSC Early Career Researcher Award. He is currently serving on the editorial boards of *Journal of Systems Architecture, Microprocessors and Microsystems, Journal of Circuits, Systems and Computers*. He is an IEEE SENIOR MEMBER and ACM SENIOR MEMBER.

Wenhong Ma is currently working toward the Ph.D. degree with Hunan University, Changsha, China.

His current research interests include real-time systems, automotive embedded systems, embedded safety and security.

Hao Peng is currently working toward the M.S. degree with Hunan University, Changsha, China.

His current research interests include include real-time systems, automotive embedded systems, embedded safety and security.

Renfa Li (Senior Member, IEEE) is a Professor with the Department of Computer Engineering, College of Computer Science and Electronic Engineering, Hunan University, China. He is the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. His major interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of things. He is a member of the council of CCF and a senior member of ACM.

Keqin Li (Fellow, IEEE) is a SUNY Distinguished Professor of COMPUTER SCIENCE with the State University of New York. He was an Intellectual Ventures endowed Visiting Chair Professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China, during 2011–2014. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 580 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.