Toward Efficient Design Space Exploration for Fault-Tolerant Multiprocessor Systems

Bo Yuan, *Member, IEEE*, Huanhuan Chen, *Senior Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—The design space exploration (DSE) of fault-tolerant multiprocessor systems is very complex, as it contains three interacting NP-hard problems: 1) task hardening; 2) task mapping; and 3) task scheduling. In addition, replication-based task hardening can introduce new tasks, called replicas, into the system, enlarging the design space further. As a population-based global optimization algorithm, evolutionary algorithms (EAs) have been widely used to explore this huge design space over the last decade. However, as analyzed in this paper, the search space of previous works is highly redundant, resulting in poor efficiency and scalability. This paper proposes an efficient EA-based DSE method for the design of large-scale fault-tolerant multiprocessor systems. The main novelties of this paper include: 1) mapping exploration is explicitly separated, i.e., task mapping is optimized during the evolutionary search, while replica mapping is constructed heuristically according to the current co-synthesis state; 2) the design space of task hardening and task mapping are explored independently by a cooperative co-EA; and 3) as a complement to global search of EA, problem-specific local search operators are designed for both task hardening and task mapping, reducing the number of fitness evaluations required. Compared with the most relevant state-of-the-art method, the superiority of the proposed method is demonstrated using extensive experiments on a large set of benchmarks, e.g., 1.75×∼2.50× better results can be obtained on the benchmarks of 300 tasks and 30 processors.**

Manuscript received May 1, 2018; revised October 4, 2018 and January 29, 2019; accepted April 18, 2019. Date of publication April 23, 2019; date of current version January 29, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFC0804002, in part by the National Natural Science Foundation of China under Grant 91846111, Grant 91746209, and Grant 617611360, in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X386, in part by the Shenzhen Peacock Plan under Grant KQTD2016112514355531, in part by the Science and Technology Innovation Committee Foundation of Shenzhen under Grant ZDSYS201703031748284, Grant JCYJ20170817112421757, and Grant JCYJ20180504165652917, and in part by the Program for University Key Laboratory of Guangdong Province under Grant 2017KSYS008. *(Corresponding author: Xin Yao.)*

B. Yuan is with the Shenzhen Key Laboratory of Computational Intelligence, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: yuanb@sustc.edu.cn).

H. Chen is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China (e-mail: hchen@ustc.edu.cn).

X. Yao is with the Shenzhen Key Laboratory of Computational Intelligence, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China, and also with the CERCIA, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: xiny@sustc.edu.cn).

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/TEVC.2019.2912726

***Index Terms*—Bi-level optimization, cooperative co-evolutionary algorithm (CCEA), design space exploration (DSE), fault-tolerance, multiprocessor system, task mapping.**

I. Introduction

As THE rapid advances in transistor size scaling, high operation frequency, and low voltage level, the number of transient faults (usually caused by radiation) in circuits has dramatically increased in the past years. At the same time, as the technology advancement continues, more processors have been integrated into a single chip to meet the growing computing demand of modern applications. Therefore, the tolerance to transient faults (or soft errors) is one of the major requirements in the design of nanoscale multiprocessor systems [1], [2], [42].

During the decades, various fault tolerance techniques have been proposed against transient faults, such as replication [3], re-execution [4], and checkpointing [5]. Since they are usually characterized by different time and space overheads, there is a tradeoff in the assignment of different fault tolerance techniques to different portions of a system. This tradeoff, together with traditional steps in system-level co-synthesis, i.e., task mapping and task scheduling, makes the design space of fault-tolerant multiprocessor system increase dramatically.

In general, the design space exploration (DSE) process of a fault-tolerant multiprocessor system consists of the following [6]–[9].

1) Task hardening, i.e., selecting one of the fault tolerance techniques for each task (a task is the atomic unit performed by the system).
2) Task mapping, i.e., mapping each task to one of the processors in the multiprocessor platform.
3) Task scheduling, i.e., scheduling the tasks on each processor.

All of them are NP-hard problems. It is notable that replication-based task hardening can introduce new tasks [3], i.e., replicas, into the system, and these replicas should be mapped and scheduled as well, thereby enlarging the original design space further. In this case, the design of fault-tolerant multiprocessor systems can be regarded as a bi-level optimization problem due to its nested structure. Specifically, the upper level optimization is task hardening, and the lower level optimization is task (including replicas) mapping and scheduling, thereby making the overall optimization computationally very intensive.

Given all these degrees of freedom, in order to obtain optimal designs of fault-tolerant multiprocessor systems, a variety of DSE methods have been proposed over the last decade, especially the ones based on evolutionary algorithms (EAs) [6]–[9]. As a population-based metaheuristic optimization algorithm, EAs often provide good near optimal solutions to many types of problems, but their high time complexity [22], [23] is a prohibiting factor in real-word applications. Besides, EAs usually do not involve any high-level problem-specific knowledge beyond that required in fitness evaluation [24], [25]. Due to the huge design space of fault-tolerant multiprocessor systems, we need a customized EA-based DSE method rather than a general EA.

This paper builds upon the analysis that the search space of previous EA-based DSE methods is highly redundant, because: 1) task hardening is implicitly embedded in task mapping, and the number of replicas that might be introduced by replication is assumed to be maximum, resulting in long chromosomes [8], [9], [11]–[14] and 2) both task mapping and replica mapping are encoded in the same chromosome, resulting in a many-to-one relationship between genotypes and phenotypes [7], [11]–[16]. A detailed discussion will be given in Section II-B.

As a remedy, in this paper: 1) the mapping of a task is done differently depending on whether it is an original task or a replica, i.e., task mapping is optimized during the evolutionary search, while replica mapping is constructed heuristically according to the current co-synthesis state; 2) the design space of task hardening and task mapping are searched separately by a cooperative co-evolutionary algorithm (CCEA); and 3) as a complement to the global search, problem-specific local search operators are designed for both task hardening and task mapping to reduce the number of fitness evaluations required. The overall goal of this paper is to speed up the whole DSE process of the EA-based method and enable the EA-based DSE method applicable to large-scale problems. Compared with the most relevant state-of-the-art method, the superiority of the proposed method is demonstrated using extensive experiments on a large set of benchmarks, e.g., $1.75\times \sim 2.50\times$ better results can be obtained on the benchmarks of 300 tasks and 30 processors. The major contributions of this paper are summarized as follows.

1) An in-depth analysis is given for the encoding redundancy of previous EA-based DSE methods.
2) A separating method is developed to explore the mappings of original tasks and replicas explicitly.
3) A divide-and-conquer CCEA framework is proposed to solve task hardening and task mapping simultaneously.
4) Problem-specific local search operators are designed in the framework for further accelerating the DSE process.
5) Extensive experiments on a large set of benchmarks demonstrate the performance of the proposed DSE method.

The rest of this paper is organized as follows. Section II gives a critical overview of previous methods. In Section III, the adopted problem model is briefly introduced. Our proposed method is then detailed in Section IV. The experimental studies are given in Section V. Finally, Section VI concludes this paper.

II. Related Work

In this section, a comprehensive review of previous DSE methods for fault-tolerant multiprocessor system design is given at first, and then, their drawbacks, especially the redundancy in search space, are analyzed in detail.

A. Overview

COFTA [26] can be regarded as the first system co-synthesis framework for fault-tolerant multiprocessor systems. Fault detection is achieved through the placement of assertions and duplicate-with-compare tasks during the co-synthesis process, and assertion is preferred since it has less overhead than duplication.

Duplication is considered in [27]. The key idea is to insert duplicated tasks in the idle time slots of the processors such that fault detection incurs no performance degradation. One heuristic is to embed task insertion in the co-synthesis process, and another heuristic is to perform task insertion as the second step after the co-synthesis of the nominal system.

When applying more complex fault tolerance techniques, such as replication and re-execution, EA-based DSE methods have been widely used in the last decade.

Task replication is explicitly applied in [10] where multiple task instances are executed independently on different processors. For each task, the information about the number of instances, mapping priority, and scheduling priority is encoded in separate chromosomes, making up an individual of the EA.

Task replication is implicitly implemented in [11] by mapping a task to a set of different processors. For each task, the priority list for all its mapping edges and the number of mapping edges to be selected are encoded as integer genotypes in the chromosome of the EA. This paper is further extended to consider voter placement during the DSE process [12]. The same authors also use Boolean genotypes to encode task instances and the problem is solved by an integer linear programming solver driven by an EA [40].

Task re-execution, i.e., the task instance is executed again if a fault is detected, together with replication, is implicitly implemented in [13]. For each task, the processors that the task is mapped to are encoded as a list of integer values in the chromosome of the EA, where multiple mappings of the same task onto the same processor is interpreted as re-execution, and multiple mappings of the same task onto different processors is interpreted as replication. This paper is further extended to consider the selection of fault detectors [14].

Both replication and re-execution are explicitly considered in [28]. Tabu search (TS) is used to explore both task hardening and task mapping. The TS-based DSE method is further extended to take hardware hardening and fault detection implementation into account in [29] and [30], respectively. As the performance of TS is generally dependent on its initial solution, their work starts from several different pregenerated solutions for improving the quality of the results.

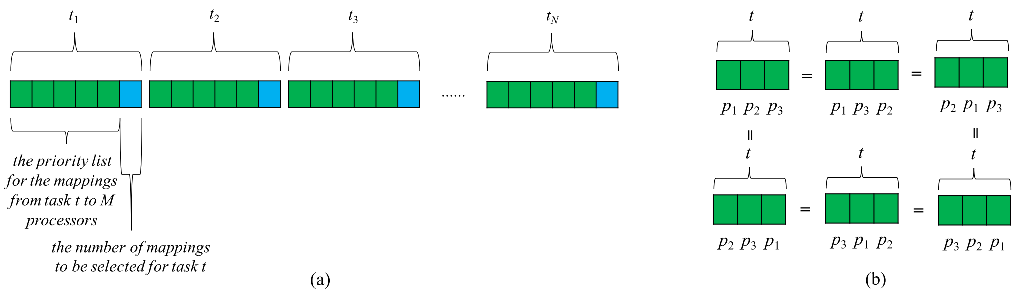


Fig. 1. Motivational examples of the encoding redundancy for (a) task replication and (b) task mapping.

An extended set of fault tolerance techniques is explicitly considered in [15], including the hardware architectures offering fault detection/tolerance. A two-layer (TL) EA framework is proposed, where the outer layer EA is used to explore task hardening, and each generated solution is evaluated by the inner layer EA that performs the exploration of task mapping and replica mapping. It is notable that the EA-based fitness evaluation of the solutions generated in the outer layer is very time consuming. This paper is further extended to consider resource allocation [16] and task grouping [7].

Passive replication, active replication, and re-execution are considered in [8]. For each task, the redundancy degree of re-execution, mappings of active and passive task instances, and mapping of the voter are encoded in the chromosome of the EA, and a randomized heuristic is designed to repair the infeasible solutions. This paper is further extended to consider resource allocation and task dropping [9].

B. Discussion

As mentioned above, various EA-based DSE methods, using different encoding schemes and optimization frameworks, have been proposed. However, we find that the search space of these methods is highly redundant.

EA is population-based, and in general, an individual must hold information to represent a complete solution, i.e., both task hardening and task mapping. If task hardening schemes, especially replication, is implicitly implemented in task mapping [8], [9], [11]–[14], the chromosome length of task mapping has to be very long. This is because the length is determined by the maximum number of task instances that may be introduced by replication. For example, if a task can be mapped to any processor, the maximum number of instances of a task is equal to the number of processors that could perform the task. As shown in Fig. 1(a), the maximum chromosome length of N tasks distributed on M processors is $N(M + 1)$. However, the real number of instances for a task is often much less than the number of processors, only a small part of the chromosome contributes to the fitness evaluation. If re-execution is considered as well as replication, long chromosome is unnecessary for a task hardened by re-execution, because all instances of the task should be mapped to a single processor.

An effective way to avoid the aforementioned problem of long chromosomes is to explicitly separate task hardening from task mapping in chromosome encoding [7], [10], [15], [16]. For example, task hardening is encoded first and the output is an extended task graph including both tasks and replicas, and then the encoding of task mapping operates on the extended task graph [10]. However, the chromosome length of task mapping may be different for different extended task graphs. This problem could be resolved in the TL EA [7], [15], [16], where the outer layer EA is used to explore task hardening, while the inner layer EA is used to explore task mapping on each extended task graph independently. The TL EA seems to be the most promising DSE framework, but the EA-based fitness evaluation is very time consuming.

In addition to the problem of long chromosomes, there is a many-to-one relationship between genotypes and phenotypes in most previous methods, as multiple replicas of a task are encoded in the same chromosome [7], [11]–[16]. For example, if we use integer encoding for task mapping, different permutations of a fixed integer set indicate the same mapping scheme for a replicated task. As shown in Fig. 1(b), the mappings three replicas of task t to three processers $\{p_1, p_2, p_3\}$ can be interpreted from six different chromosomes. This is because all replica instances of the same task are homogeneous and their mappings are interchangeable. However, all these permutations are regarded as different points in the search space, and all these points may be visited and evaluated during the evolutionary search. A similar problem is considered in [41], where the symmetry of mapping tasks to processors of the same processor type is eliminated by clustering.

To consider high-level problem-specific knowledge beyond that required in fitness evaluation, the work in [21] proposed an importance measure (IM)-based local search approach to enhance the existing EA-based DSE methods, but it is only applicable to the hardening level selection of processors, which is a very costly hardware fault-tolerant technique, compared with software fault-tolerant techniques, such as task replication and re-execution.

As pointed out above, the search space of previous EA-based DSE methods is very large and highly redundant, because of long chromosomes [8], [9], [11]–[14] and the many-to-one relationship between genotypes and phenotypes [7], [11]–[16]. The goal of this paper is to reduce these redundancies by separating replica mapping from task mapping and using a divide-and-conquer strategy (i.e., CCEA) for task hardening and task mapping. Furthermore, the problem-specific local search is incorporated in the DSE process.

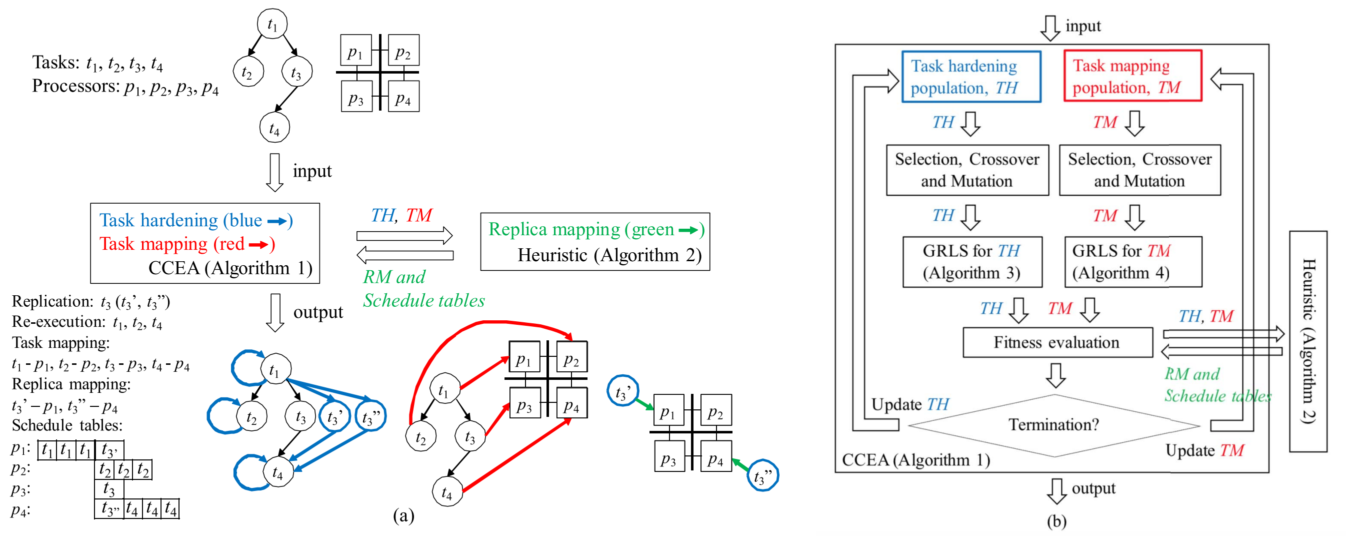

Tasks: t_1, t_2, t_3, t_4
Processors: p_1, p_2, p_3, p_4
input
Task hardening (blue)
Task mapping (red)
CCEA (Algorithm 1)
TH, TM
RM and Schedule tables
Replica mapping (green)
Heuristic (Algorithm 2)
output
Replication: t_3 (t_3', t_3")
Re-execution: t_1, t_2, t_4
Task mapping:
t_1 - p_1, t_2 - p_2, t_3 - p_3, t_4 - p_4
Replica mapping:
t_3' – p_1, t_3" – p_4
Schedule tables:
(a)
Task hardening population, TH
Task mapping population, TM
Selection, Crossover and Mutation
GRLS for TH (Algorithm 3)
GRLS for TM (Algorithm 4)
Fitness evaluation
Termination?
Update TH
Update TM
CCEA (Algorithm 1)
Heuristic (Algorithm 2)
output
(b)

Fig. 2. Whole (a) DSE framework and the (b) CCEA framework.

III. Problem Model

This section briefly introduces the adopted system model, fault tolerance techniques, and system requirements in the fault-tolerant multiprocessor system design.

A. System Model

The multiprocessor platform consists of a set of heterogeneous processors interconnected by an on-chip network [8], [9], as shown in Fig. 2(a). Generally, the multiprocessor platform can be described using an architecture graph [21], $A(P, F)$, characterized by a set of processors $P = \{p_1, p_2, \ldots, p_M\}$ and a full connectivity communication fabric $F = \{(p_i, p_j), \ldots\}$.

The application of a system is made up of a set of sequential and/or concurrent tasks, or processes. Tasks are the atomic unit performed by the system and communicate with each other. In general, the application can be described using a task graph [6]–[9], $G(T, E)$, as shown in Fig. 2(a). $G(T, E)$ is characterized by a set of tasks $T = \{t_1, t_2, \ldots, t_N\}$ and the dependency between task pairs $E = \{(t_i, t_j), \ldots\}$. In addition, each task $t_i \in T$ is characterized by the worst-case execution time (WCET) wcet_i^j on each processor $p_j \in P$.

B. Hardening Technique

As mentioned in Section II, replication and re-execution are the most widely used software hardening techniques against transient faults or soft-errors [8], [9]. Re-execution uses time redundancy to tolerate fault occurrences, while replication provides space redundancy, thus the combination of re-execution and replication may achieve an optimal tradeoff for the whole system design. Both of them are considered in this paper, and in this paper, the term "hardening" represents the assignment of fault tolerance technique.

In re-execution scheme, the initial state must be saved in advance in the memory. Once a fault is detected at the end of the task execution, all the input values are rolled-back to the initial state and the same task instance is executed again. In replication scheme, multiple instances (at least two) of the hardened task are mapped to a set of different processors, and all task instances are executed independently of fault occurrences. It is notable that replication can modify the topology of the original task graph by introducing replicas [6], [13], [28].

C. System Requirements

The transient faults may occur anywhere in the system during one operation cycle of the application, so the tolerance to transient faults is regarded as a hard constraint, i.e., a specified number (marked as K) of different transient faults has to be tolerated [6], [7]. As a perfect fault detector is generally assumed at the end of the task execution, thus, if a task is hardened by replication, to tolerate K faults, $K + 1$ instances of the task are executed independently; if a task is hardened by re-execution, to tolerate K faults, the maximum number of executions of the task instance is $K + 1$.

It should be noticed that several assumptions are implicated in the K-fault tolerance systems: 1) task instances with different execution time fail with the same probability and 2) there is no objective function related to fault tolerance, as K-fault tolerance is regarded as a hard constraint.

Given a multiprocessor platform $A(P, F)$, and an application $G(T, E)$ that shall be implemented on the platform, the goal of the design of fault-tolerant multiprocessor systems is to: 1) assign replication or re-execution to each task, i.e., task hardening; 2) map each task instance (including replicas) to one processor, i.e., task mapping; and 3) schedule the task instances (including replicas) on each processor, i.e., task scheduling, such that the overall execution period of the given application is minimized.

IV. Proposed Method

The overall goal of our proposed method is to speed up the whole EA-based DSE process (including task hardening, task mapping, and task scheduling), through reshaping the search space and reducing the number of fitness evaluations.

Algorithm 1 CCEA for Task Hardening and Task Mapping

```
Input: architecture graph $A(P,F)$ and task graph $G(T,E)$ and $WCETs$ and $K$, and $P_S, P_c, P_m$
Output: the task hardening population $TH$, the task mapping population $TM$, the replica mapping population $RM$, and the schedule tables
01: $TH = \{th^i\}, th^i = \{th_1, th_2, \ldots th_N\} \in \{0,1\}^N, i = \{1,2,\ldots PS_{TH}\}$
02: $TM = \{tm^i\}, tm^i = \{tm_1, tm_2, \ldots tm_N\} \in \{1,2,\ldots,M\}^N, i = \{1,2,\ldots PS_{TM}\}$
03: evaluate $fit(th^i, tm^j), i = \{1,2,\ldots PS_{TH}\}, j = \{1,2,\ldots PS_{TM}\}$
04: $fit^i_{TH} = \min(fit(th^i, tm^j))$ and $thbest = \mathrm{argmin}_i(fit^i_{TH}), i = \{1,2,\ldots PS_{TH}\}$
05: $fit^i_{TM} = \min(fit(th^j, tm^i))$ and $tmbest = \mathrm{argmin}_i(fit^i_{TM}), i = \{1,2,\ldots PS_{TM}\}$
06: repeat
07: for $i = 1$ to $PS_{TH}$ do // evolve the task hardening population
08:   $(th^j, th^k) \leftarrow$ Selection for Reproduction $(TH)$
09:   $oth^i \leftarrow$ Crossover $(th^j, th^k, P_c)$
10: end for
11: for $i = 1$ to $PS_{TH}$ do
12:   $oth^i \leftarrow$ Mutation $(oth^i, P_m)$
13: end for
14: for $i = 1$ to $PS_{TH}$ do
15:   $fit^i_{NTH} \leftarrow$evaluate $oth^i$ by combining it with $tmbest$
16: end for
17: $TH \leftarrow$ Selection for Survival $(TH, OTH, fit_{TH}, fit_{OTH})$
18: for $i = 1$ to $PS_{TM}$ do // evolve the task mapping population
19:   $(tm^j, tm^k) \leftarrow$ Selection for Reproduction $(TM)$
20:   $otm^i \leftarrow$ Crossover $(tm^j, tm^k, P_c)$
21: end for
22: for $i = 1$ to $PS_{TM}$ do
23:   $otm^i \leftarrow$ Mutation $(otm^i, P_m)$
24: end for
25: for $i = 1$ to $PS_{TM}$ do
26:   $fit^i_{OTM} \leftarrow$ evaluate $otm^i$ by combining it with $thbest$
27: end for
28: $TM \leftarrow$ Selection for Survival $(TM, OTM, fit_{TM}, fit_{OTM})$
29: update $thbest$ and $tmbest$
30: until maximum number of fitness evaluations reached
```

Algorithm 2 Heuristic Replica Mapping Algorithm

```
Input: architecture graph $A(P,F)$ and task graph $G(T,E)$ and $WCETs$ and $K$, the task hardening solution $th$ and the task mapping solution $tm$
Output: the replica mapping solution $rm$ and the schedule table
01: obtain the extended task graph $G'(T', E')$
02: compute the priority for each task $t_i \in T'$
03: initialize $C_i = \{P/p_j\}$ for each task $t_i \in T$ and $tm_i = j$
04: initialize $I_i = 0$ for each processor $p_i \in P$
05: $L = \{$the source node from $T'\}$
06: while $L \neq \varnothing$ do
07:   select $t_r \in L$ with the highest priority
08:   if $t_r \in T$ do
09:     $t_r$ runs on $p_j$ as given in $tm$
10:   end if
11:   if $t_r \in \{T'/T\}$ do
12:     estimate $EFT_j = I_j + wcet^j_r$ for each $p_j \in C_i$ (it is assumed that $t_r$ is a replica of task $t_i$)
13:     select $p_k \in C_i$ with the least EFT value
14:     $rm(t_r) = p_k$
15:     $C_i = \{C_i/p_k\}$
16:   end if
17:   schedule $t_r$ on $p_k$, and adjust the recovery slack of $t_r$ according to [6]
18:   $L = \{L/t_r\}$
19:   add successors of $t_r$ to $L$
20: end while
21: return $rm$ and schedule tables
```

Algorithm 3 GRLS for Task Hardening

```
Input: the given task hardening solution $th$
Output: new task hardening solution $th$
01: for each task on the critical path $t_i \in CP$ do
02:   if $th_i = 1$ do //re-execution
03:     $th_i = 0$ //replication
04:     break
05:   end if
06: end for
07: return $th$
```

The proposed DSE framework is shown in Fig. 2(a). Both architecture and task graphs are the inputs to the DSE framework. In the proposed DSE framework, instead of using the TL EA strategy [7], we use a divide-and-conquer strategy to avoid the overestimation of chromosome length, i.e., the design space of task hardening and task mapping are explored independently using a CCEA framework [i.e., Algorithm 1, the left part in Fig. 2(a)]. In addition, to reduce the redundancy in genotype–phenotype mappings, we propose to explicitly separate the mappings of replicas from the mappings of tasks in chromosome encoding. The mappings of replicas are constructed heuristically [i.e., Algorithm 2, the right part in Fig. 2(a)]. The outputs of the DSE framework are the final solutions of fault tolerance technique assignment for each task, the processor assignment for each task (including replicas), and the schedule table for each processor.

Fig. 2(b) shows the details of the CCEA framework. The search space of task hardening and task mapping are explored independently using two interacting populations. The fitness value is the overall execution period of the hardened task graph, in other words, the earliest finish time (EFT) of the sink node. As a complement to the EA-based global search framework, problem-specific local search operators (i.e., Algorithms 3 and 4) are designed for both task hardening and task mapping. In order to evaluate the solution, task hardening and task mapping should be combined as a complete solution.

The details are given in the following sections.

A. Coevolution for Task Hardening and Task Mapping

CCEAs have been used with success in many applications [31]–[33]. We propose, for the first time, to co-evolve solutions of task hardening and those to task mapping simultaneously.

In our CCEA framework, the problem of task hardening and task mapping is decomposed as two interacting subproblems. Each subproblem is assigned to a population, and each population evolves (through selection, crossover, and mutation) independently of the other. Since any given individual from a particular population represents only task hardening or task mapping, a partner, who has the best fitness value, is selected from the other population. Then, the individual is combined with its partner to form a complete solution and its fitness value is evaluated based on Algorithm 2, where the schedule table is built during the heuristic replica mapping process. It should be noted that a complete solution consists of not only task hardening and task mapping but also replica mapping and task/replica scheduling, the latter can be constructed heuristically and will be introduced in Section IV-B.

The genetic algorithm (GA) is adopted in our CCEA framework to work as the evolutionary engine for each population. The detailed design of the major steps for evolving the

Algorithm 4 GRLS for Task Mapping

```
Input: the given task mapping solution $tm$
Output: new task hardening solution $tm$
01: for each task on the critical path $t_i \in CP$ do
02:   $p_j = tm_i$
03:   if $j! = \text{argmin}_k\ (WCET_i^k)$ do
04:     $tm_i = k$, st. $wcet_i^k < wcet_t^j$
05:     break
06:   end if
07: end for
08: return $tm$
```

populations of task hardening and task mapping is given as follows.

1) Encoding: In the GA, the solution to the optimization problem is encoded in a chromosome as a set of parameters. Since task hardening and task mapping are regarded as two independent subproblems in our CCEA framework, the encoding of their solutions used in our implementation is straightforward.

Each chromosome in the *task hardening* population is represented by an N-dimensional 0-1 vector $\text{th} = \{\text{th}_1, \text{th}_2, \ldots, \text{th}_N\} \in \{0, 1\}^N$, where N is the number of tasks. Each position of the vector describes the fault tolerance technique that assigned to the task, i.e., $\text{th}_i = 0$ indicates that task t_i is hardened by replication, and, $\text{th}_i = 1$ indicates that task t_i is hardened by re-execution.

Each chromosome in the task mapping population is represented by an N-dimensional integer vector $\text{tm} = \{\text{tm}_1, \text{tm}_2, \ldots, \text{tm}_N\} \in \{1, 2, \ldots, M\}^N$, where N is the number of tasks, M is the number of processors, and the processors are numbered in sequence. Each position of the vector describes the processor that the task is mapped to, i.e., $\text{tm}_i = j$ indicates task t_i is mapped to processor p_j.

2) Crossover: In the GA, crossover is a genetic operator that recombines two parent chromosomes to produce one or two child chromosomes from them. Many crossover operators have been defined and can be used on the 0-1 or integer vector, such as one-point crossover, two-point crossover, and uniform crossover.

In our implementation, we use one-point crossover, the simplest one, i.e., a single crossover point on both parent vectors is selected first, and then all data beyond that point in either vector is swapped between the two parent vectors, the resulting vectors are the child chromosomes. In this way, the applied fault tolerance techniques are mixed for the task hardening solutions, or the applied mapping schemes are mixed for the task mapping solutions.

3) Mutation: In the GA, mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state to maintain the genetic diversity of the population. In general, mutation operators involve a probability that an arbitrary gene in a chromosome would be changed from its original state, and this probability should be set low.

In our implementation, we use flip mutation for the 0-1 vector, i.e., the value of the chosen gene is inverted, and use uniform mutation for the integer vector, i.e., the value of the chosen gene is replaced by a uniform random value selected between the specified upper and lower bounds for that gene. In this way, the applied fault tolerance technique is changed for the task in the task hardening solution, or the assigned processor is changed for the task in the task mapping solution.

4) Selection: Selection occurs two times during each generation in the GA. In our implementation, selection for reproduction is performed before the crossover operator is applied, which is based on a purely random basis without bias to filter any individual, and selection for survival is performed to reduce the population to its original size, and we use an elitist scheme through choosing the best individuals from the combined pool of parents and children.

5) Fitness Evaluation: Each individual from one population is combined with its partner from the other population, and then its fitness value is evaluated based on the following heuristic construction of replica mapping (Algorithm 2). In our implementation, the best solution found so far is defended as the partner in the population.

The outline of the proposed CCEA for task hardening and task mapping is given in Algorithm 1, where PS, P_c, and P_m indicate the population size, probability of crossover, and probability of mutation. The algorithm starts with initial populations TH and TM consisting of PS_{TH} and PS_{TM} random individuals for task hardening and task mapping (lines 1 and 2). The fitness of each individual fit_{TH} or fit_{TM} is initialized as the minimum fitness when combine it with any individual from the other population, and the individual who has the best fitness is regarded as the partner thbest or tmbest in its population (lines 3–5). During each generation, both TH and TM are evolved, the population of PS_{TH} and PS_{TM} individuals generates PS_{TH} and PS_{TM} children through the crossover operator (line 9 or 20) and the mutation operator (line 12 or 23). The offspring OTH or OTM are evaluated by combining them with tmbest or thbest (line 15 or 26) and then used to update the current population (line 17 or 28). Finally, the partners, thbest and tmbest, are updated based on the new populations (line 29). When the given maximum number of fitness evaluations is reached, the algorithm stops (line 30).

It should be noticed that tmbest or thbest is selected as the partner solution for fitness evaluation of the given solution, not for crossover and mutation. Besides, how to evaluate the fitness value of a solution is an open problem in the CCEA field. In fact, there is a tradeoff between evaluation accuracy and evaluation time. For example, if we select a set of partner solutions (e.g., 10) to evaluate one solution, the accuracy will increase, but the runtime will be extended by ten times. In this paper, we just use a simple evaluation strategy that only a single partner solution is used to evaluate the given solution. As we do not use the whole partner population (i.e., 200 in our experiments) to evaluate the given solution from the point of view of evaluation time, the resulting solution (tmbest or thbest) is not necessarily the best one.

B. Separation of Task Mapping From Replica Mapping

In the proposed DSE method, task mapping is optimized during the aforementioned CCEA process, while replica mapping is constructed heuristically according to the current state of system co-synthesis, i.e., the current mappings of tasks and the current schedule tables on the processors. As shown in Fig. 2(b), fitness evaluations in the CCEA framework are based on the feedback from replica mapping, we will introduce our heuristic for the construction of replica mapping. Then, we will discuss the advantages of the proposed method over a full evolutionary search (used in the TL EA framework) for both task mapping and replica mapping.

In our implementation, given task hardening and task mapping, replica mapping is performed during task scheduling, i.e., the mappings of replicas are integrated into the static scheduling algorithm proposed in [6]. Our proposed heuristic replica mapping algorithm is given in Algorithm 2.

At first, it is easy to obtain the extended task graph *G*' by inserting the replicas (determined by the task hardening solution *th*) into the original task graph *G* (line 1). Then, each task is assigned a priority which is computed based on the maximum path length between the task and the sink task, and the replicas of a task have the same priority as the original task (line 2). The candidate processor set for mapping the replicas of task t_i is initialized as $C_i = \{P/p_j\}$, where p_j is the processor where t_i has been mapped as given in the task mapping solution tm (line 3). The initial current idle time I_i for each processor p_i is initialized as 0 (line 4).

The tasks and replicas are placed in the ready list if all their predecessors have been already mapped. All ready tasks and replicas from the list are investigated, and the task or replica which has the highest priority is selected for scheduling or scheduling and mapping. It is notable that although the replicas have the same priority as the original task, we always select the original task first, since its mapping has been given in task mapping. The replica mapping algorithm loops until the ready list is empty.

At first, task or replica t_r is selected from the ready list according to the priority (line 7), if t_r is an original task, t_r runs on processor p_j as given in task mapping (lines 8–10). While, if t_r is a replica, for each processor $p_j \in C_i$ (it is assumed that t_r is a replica of task t_i), the EFT of t_r on p_j is estimated based on the current idle time of p_j and wcet_r^j (line 12), and t_r is mapped to processor p_k with the least EFT value (lines 13 and 14), then p_k is removed from C_i to avoid mapping two replicas of the same task to the same processor (line 15). And then, t_r is scheduled and its recovery slack is adjusted such that it can accommodate recovering of tasks and replicas scheduled before t_r on the same processor p_k (line 17). As the slack recovery adjustment strategy is out of the scope of this paper, we suggest finding the details of this strategy in [6]. After the adjustment of the recovery slack, t_r is removed from the ready list (line 18) and its successor tasks and replicas are added to the list (line 19). After scheduling all the tasks and replicas in the extended task graph, the algorithm returns both mappings of replicas and schedule tables on processors (line 21). It is notable that the integration of replica mapping does not improve the time complexity of the original static scheduling algorithm, and the impact on runtime is negligible as shown in the experiments (Section V).

Besides, replica mapping can be performed after task scheduling as well. For example, the static scheduling algorithm [6] is invoked first, and then the replicas are inserted in the schedule tables one by one, and meanwhile the schedule tables are adjusted correspondingly. In general, this kind of methods incurs much higher time complexity than the aforementioned heuristic [27], so it is not applicable to be integrated into the EA-based DSE framework, where replica mapping and scheduling, as a part of fitness evaluation process, are invoked frequently.

Compared with a full evolutionary search for both task mapping and replica mapping [7], the benefits of the proposed method of separation of task mapping and replica mapping are threefold.

1) Given a task hardened by replication with redundancy degree d and generally $d \leq M$ (M is the number of processors), the number of all permutations that would be visited during a full evolutionary search is A_M^d (i.e., $C_M^d \cdot A_d^d$), while the number of all different mapping schemes is C_M^d, where C_m^n indicates the number of n-combinations of m and A_m^n indicates the number of n-permutations of m. In the proposed method, the original task is mapped first to one of the M processors, and then the mappings of other $d-1$ replicas are constructed by the deterministic heuristic, thus the number of permutations that would be visited by the proposed method is $M \cdot C_{M-1}^{d-1}$ (i.e., $C_M^d \cdot d$) in the worst case, and the redundancy is reduced by at least a factor of A_{d-1}^{d-1}.
2) Generally, it is not a good choice of mapping the replicas of a task to the same processor as the original task, or mapping two replicas of a task to the same processor. However, for a full evolutionary search, it is very hard to maintain this requirement due to the randomness in crossover and mutation operations. In the proposed method, the task is mapped first, and its replicas are mapped heuristically one by one, so infeasible mappings can be easily avoided during the heuristic replica mapping process as shown in Algorithm 2.
3) Above all, by separating replica mapping from task mapping, it is possible to optimize task hardening and task mapping in a divide-and-conquer way (i.e., CCEA) rather than a nested TL structure [7], because the length of task mapping chromosome is not determined by any specific task hardening scheme any more. This is detailed in Section IV-A.

In the proposed method, replica mapping is determined heuristically, and in general, there is an inevitable mapping bias in the heuristic deterministic search. It seems that a full evolutionary search for both task mapping and replica mapping has a stronger tendency to converge toward global optima, but it is faced with a much larger search space as discussed. As shown in the experiments (Section V-A), the proposed method has better performance in terms of both efficiency and effectiveness, and a detailed analysis will be given later. In this paper, "efficiency" represents the cost of runtime of the algorithms to achieve the objective fitness value, and "effectiveness" represents the obtained fitness values of the algorithms within the given runtime.

C. Problem-Specific Local Search

For real-world optimization problems, it is often effective to incorporate problem-specific knowledge into local search operators, which are referred to as memes in the case of memetic algorithm (MA) [34]–[36], where local search operators are used to complement the evolutionary or any population-based global search framework. For combinatorial optimization problems, local search operators generally work in the form of heuristics that are customized to a specific problem. In our implementation, the key idea is inherited from the greedy reassignment local search (GRLS) operators [37], [38], which is to reassign the gene values of part of the parent chromosome by taking advantage of the greedy information extracted from the problem instance. In this paper, this idea is extended for both task hardening and task mapping.

1) GRLS for Task Hardening: The idea of replication is to distribute the time overhead of fault-tolerance to multiple processors, in a local time slot, it generally causes less time delay than re-execution does for a hardened task. The GRLS operator designed for task hardening tries to flip the hardening technique from re-execution to replication for the given task. The outline of the proposed GRLS operator for task hardening is given in Algorithm 3. The key steps are flipping the hardening technique from re-execution to replication (lines 2 and 3 in Algorithm 3).

2) GRLS for Task Mapping: Obviously, a task can be finished earlier on a faster processor, when it is assumed that the resources on the processor are endless and the intertask communication is free. The GRLS operator designed for task mapping tries to replace the current processor by a faster one for the given task. Here, a faster processor p means that the wcet of task t on processor p is less than that of task t on processor q to which t is currently mapped. The outline of the proposed GRLS operator for task mapping is given in Algorithm 4. The key steps are replacing the current processor by a faster one (lines 2–4 in Algorithm 4).

In order to release the time overhead added to the iterative process of GA, the time complexity of the operator should be as low as possible. In fact, for each task, the priority list of processors in terms of WCET can be obtained in advance, thus the greedy information of the problem instance only needs to be computed once. Besides, for both Algorithms 3 and 4, only one change is allowed, as the solution/individual should be re-evaluated once a change is made.

It is notable that we only apply the local search operations to the tasks on the critical path (CP) of the extended task graph (line 1 in Algorithms 3 and 4). CP can be obtained according to the schedule tables on the processors. The new solution is combined with tmbest or thbest for the further evaluation of its fitness value, and then used to replace the original solution if the fitness is better.

In fact, for some specific combinations of N, M, and K, we can judge which one (re-execution or replication) is better for most tasks, but not for all tasks (as shown in Section V-E). Algorithms 3 and 4 can be regarded as knowledge-guided mutations to test whether the change can improve the current solution or not. But they are not as fully random as mutations are, because the task is chosen from the critical path and the move is guided by knowledge. In fact, more sophisticated heuristics can be used in the design of the local search operators for further improvements, but complex heuristics are time consuming. On the other hand, for EAs, greedy heuristics may make the solutions fall into local optima easily.

V. Experimental Studies

In this section, the performance of the proposed method is experimentally investigated. First, we show that by integrating the heuristic replica mapping algorithm (Algorithm 2) into the TL EA framework [7], it can speed up the whole DSE process. Then, the effectiveness of the CCEA framework and the GRLS operators is verified by extensive experiments, especially on large-scale benchmarks. Finally, a real-world case study of cruise control system (CCS) [39] is used to test the performance of the proposed method.

In our experiments, a large set of benchmarks with different scales are synthesized, e.g., $N = 40, 60, 80, 100, 200$, and 300 that indicate the number of tasks in the application, and $M = 4, 6, 8, 10, 20$, and 30 accordingly that indicate the number of processors in the multiprocessor platform. The WCETs of the tasks on the processors range from 10 to 50 ms and are randomly determined. The communication time between tasks is determined by the Manhattan distance between the locations of processors, e.g., 1 ms between processors p_1 and p_2 and 2 ms between processors p_1 and p_4 in Fig. 2(a). The detection time and recovering time involved in replication and re-execution are assumed to be 6 and 12 ms, respectively. The number (K) of transient faults that has to be tolerated is assumed to be 1, 2, or 3. We implement the algorithms [including both TL and cooperative co-evolutionary (CC) frameworks] in MATLAB and use the same scheduling algorithm [6]. All the experiments are performed on a 2×2.0 GHz Intel Xeon E5-2683 platform with 8×16 GB memory. For a fair comparison, all the tested algorithms are implemented as monolithic processes and no CPU core parallelism is exploited.

A. Effectiveness of Mapping Separation

In the TL EA framework [7], the inner layer EA is used to explore both task mapping and replica mapping and return the fitness values for the given solutions of task hardening generated by the outer layer EA. Instead of using a full evolutionary search in the inner layer, we propose to construct replica mapping by heuristics, e.g., using Algorithm 2. One hundred different solutions of task hardening are randomly generated as the inputs of the inner layer. Fig. 3 shows the evolutionary curves of the average fitness values (execution period of the whole application) using different methods, i.e., a full evolutionary search for both task mapping and replica mapping, and an evolutionary search for task mapping and the heuristic replica mapping algorithm (Algorithm 2). The methods are marked as Full GA and GA+Heuristic, respectively. GA is used as the evolutionary search engine, and the population size (PS), probability of crossover (P_c), and probability of mutation (P_m) are set as PS $= 40$, $P_c = 0.8$, and $P_m = 0.8/N$.

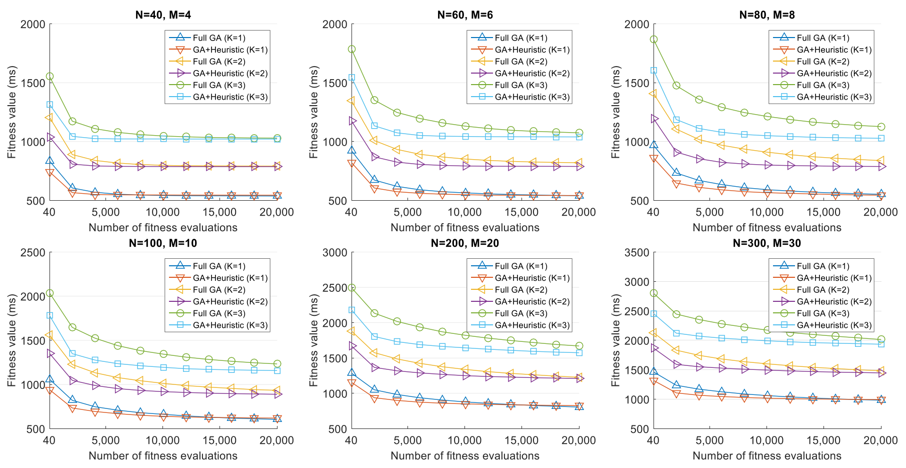


Fig. 3. Evolutionary curves of the average fitness value with the number of fitness evaluations using the full GA and GA+Heuristic on different benchmarks.

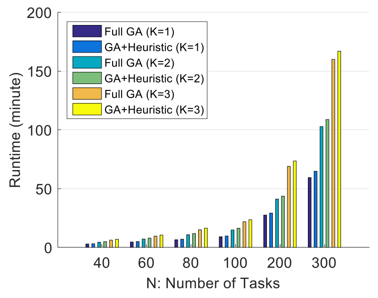


Fig. 4. Average runtime of the Full GA and GA+Heuristic on different benchmarks.

It is notable that as the inner layer, the time complexity of the GA does not allow evolving a large population within a reasonable amount of time. The maximum number of fitness evaluations is set to be 20 000 to limit the runtime, and in most cases, both algorithms have converged to good solutions. Fig. 4 shows the average runtime for different methods within the same number fitness evaluations (i.e., 20 000) on different benchmarks of different scales and fault tolerance requirements.

For GA+Heuristic, replica mapping is determined during task scheduling. As shown in Fig. 3, its advantage over Full GA is significant, e.g., GA+Heuristic not only obtained much better initialized solutions, but also converged faster to more near optimal solutions. For a given solution of task hardening, a task mapping solution that is better than the final solution obtained by the Full GA can be obtained by GA+Heuristic using a much small number of fitness evaluations (2000∼6000) instead of 20 000. In other words, the efficiency of the EA-based fitness evaluation would be improved significantly (3∼10 times). Furthermore, as the benchmark size scales up or the value of K increases, the advantage of GA+Heuristic over the Full GA becomes greater.

As shown in Fig. 4, for each benchmark case, there is no obvious difference between the runtime of the Full GA and GA+Heuristic when they are given the same number of fitness evaluations, and the runtime ranges from 3 min ($N = 40$ and $K = 1$ benchmarks) to 160 min ($N = 300$ and $K = 3$ benchmarks), which is very time consuming, as it is used as the inner loop in the TL EA framework. For example, if the maximum number of fitness evaluations of the outer layer GA is set to be 400 only, it would take about 45 days (400×160 min) to optimize $N = 300$ and $K = 3$ benchmarks.

As GA+Heuristic converged faster than the Full GA, it is expected to speed up the whole TL EA-based DSE process by using GA+Heuristic as the inner layer EA, this new method is marked as two-layer genetic algorithm II (TLGAII), while the original one using the Full GA as the inner layer EA is marked as two-layer genetic algorithm I (TLGAI). Fig. 5 shows their evolutionary curves of the average fitness values on $N = 40, 60, 80,$ and 100 benchmarks. In order to limit the computational time overhead, the maximum number of fitness evaluations of the inner layer GA is set to be 2000, instead of 20 000 used in the previous experiments. The maximum number of fitness evaluations of the outer layer GA is set to be 400, so the total number of fitness evaluations is $400 \times 2000 = 800\,000$.

As shown in Fig. 5, the advantage of TLGAII over TLGAI in term of efficiency is clear, e.g., TLGAII not only obtained much better initialized solutions but also converged fast to better near optimal solutions. This is because that given a limited computation overhead (e.g., 2000 fitness evolutions) for the inner layer GA, it is difficult for the Full GA to make a full evaluation of the given solutions generated by the outer layer GA, while GA+Heuristic could. Although the efficiency of the TL EA-based DSE method has been significantly improved by TLGAII, this kind of method still scales poorly due to the TL framework.

B. *Effectiveness of Coevolution*

In order to avoid the TL framework, we propose to optimize task hardening and task mapping separately by the CC framework. The new CCEA method is marked as cooperative co-evolutionary genetic algorithm (CCGA). The parameters of CCGA are set the same as those of the TLGAs, except for the population size that is set as PS = 200. Besides, the maximum number of fitness evaluations for CCGA is set to be 400 000. Fig. 6 shows the evolutionary curves (CCGA versus

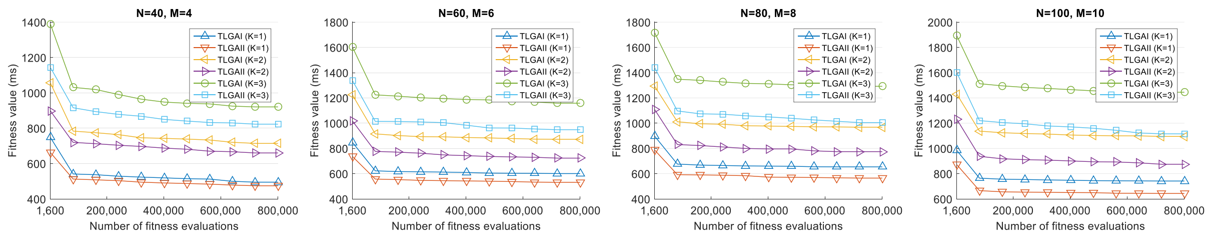


Fig. 5. Evolutionary curves of the average fitness value with the number of fitness evaluations using TLGAI and TLGAII on different benchmarks.

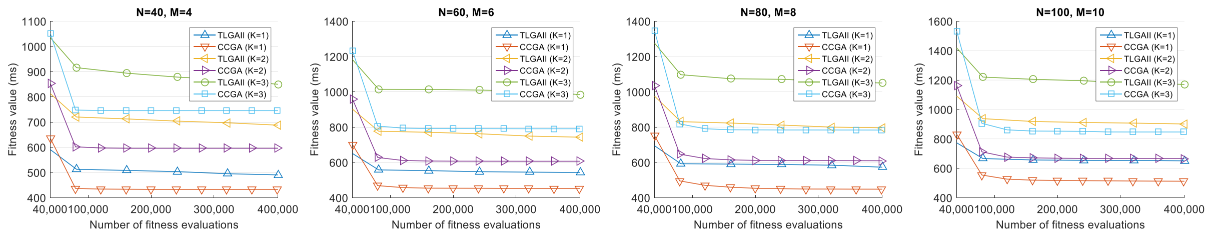


Fig. 6. Evolutionary curves of the average fitness value with the number of fitness evaluations using CCGA and TLGAII on different benchmarks.

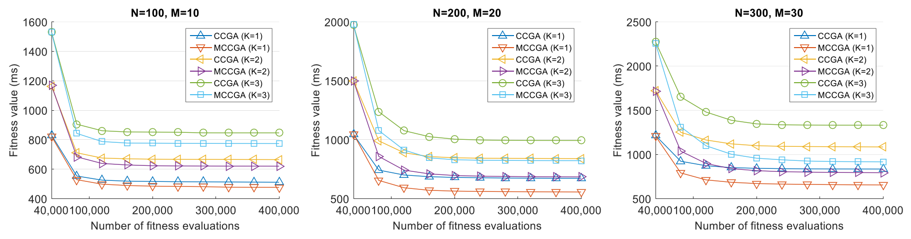


Fig. 7. Evolutionary curves of the average fitness value with the number of fitness evaluations using MCCGA and CCGA on different benchmarks.

TLGAII) of the average fitness values on $N = 40, 60, 80$, and 100 benchmarks.

As shown in Fig. 6, the advantage of CCGA over TLGAII in term of efficiency is clear. CCGA converged much faster to better near optimal solutions. As the benchmark size scales up or the value of K increases, the advantage of CCGA over TLGAII becomes greater.

C. *Effectiveness of GRLS*

We show GRLS could help to improve the efficiency of the CC framework further, especially on large-scale benchmarks. We integrate the GRLS operators into the CCEA framework, and the new method is marked as memetic CCGA (MCCGA). The parameters of MCCGA are set the same as those of CCGA. Once the state of a solution is changed by the GRLS operator, its fitness should be re-evaluated, so even with the same number of maximum fitness evolutions (e.g., 400 000), the number of iterations of MCCGA is much smaller than that of CCGA. Fig. 7 shows their evolutionary curves of the average fitness values, when $N = 100, 200$, and 300.

As shown in Fig. 7, the advantage of MCCGA over CCGA in term of efficiency is obvious. Although both methods start with similar solutions (because of the same population initialization process), MCCGA converged faster to better near optimal solutions. The performance difference between MCCGA and CCGA becomes larger as the problem grows larger.

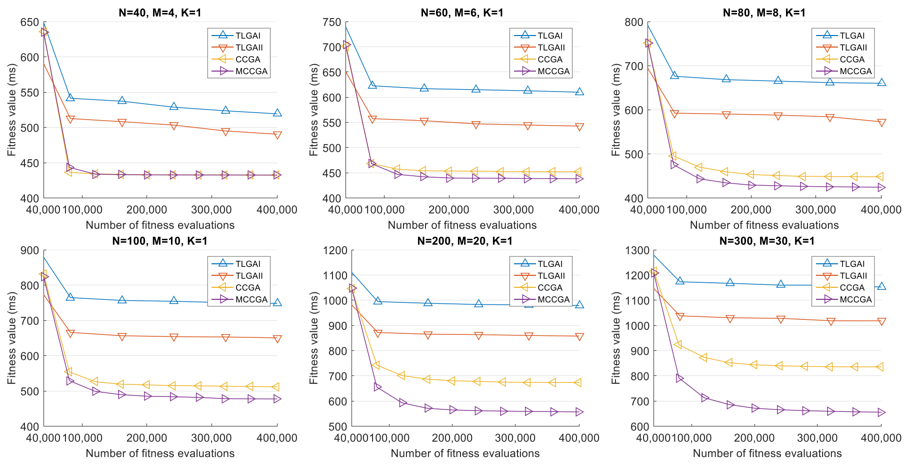
D. *Comprehensive Comparison*

Finally, a comprehensive comparison of TLGAI, TLGAII, CCGA, and MCCGA is shown in Fig. 8. The superior performance of MCCGA can be seen clearly, especially on large-scale benchmarks, e.g., the final fitness values (after 400 000 fitness evaluations) of MCCGA on $N = 300$ and $M = 30$ benchmark are about $1.75\times\sim 2.50\times$ smaller than those of TLGAI.

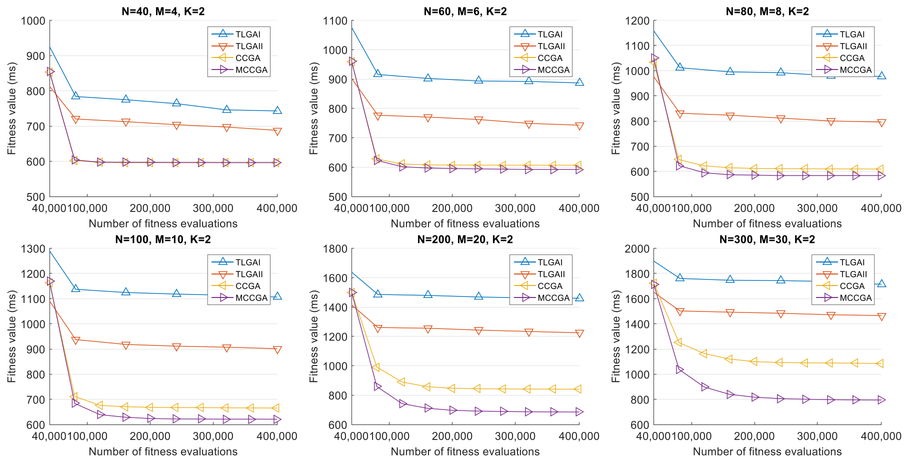
We have performed statistical tests for the results of every paired EAs, i.e., TLGAI versus TLGAII, TLGAII versus CCGA, and CCGA versus MCCGA on each benchmark instance. A two-tailed t-test is conducted with a null hypothesis stating that there is no difference between two algorithms in comparison. The null hypothesis is rejected if the p-value is smaller than the significance level $\alpha = 0.05$. We find that: 1) the results of TLGAII are statistically better than those of TLGAI on all instances; 2) the results of CCGA are statistically better than those of TLGAII on all instances; and 3) the results of MCCGA are statistically better than those of CCGA on the large instances ($N \geq 80$).

E. *Changing Ratio of N/M*

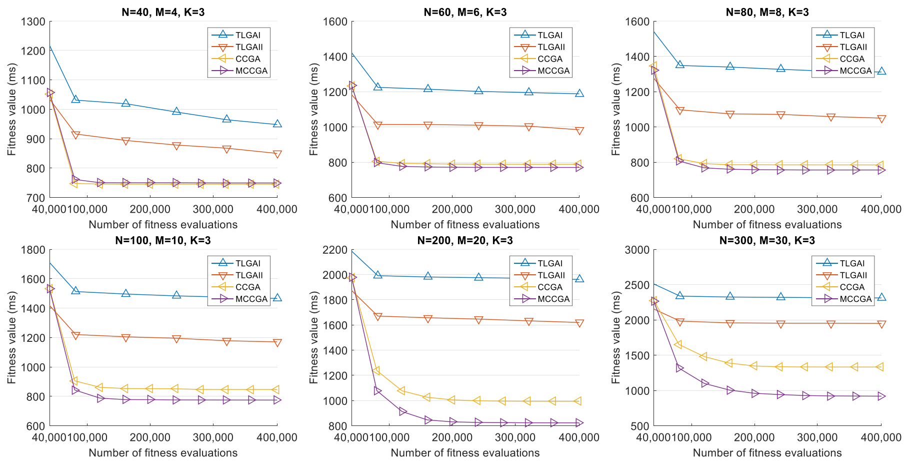
In order to investigate which task hardening technique (re-execution or replication) is more suitable in different scenarios, we change the ratio of the number of tasks to that of processors and compute the percent of re-executed tasks. As shown in Fig. 9, if the number of tasks compared to



(a)



(b)



(c)

Fig. 8. Evolutionary curves of the average fitness value with the number of fitness evaluations using TLGAI, TLGAII, CCGA, and MCCGA on different benchmarks (a) ($K = 1$), (b) ($K = 2$), and (c) ($K = 3$).

the number of processors is relatively low (e.g., $N = 40$ and $M = 10$, or $N = 100$ and $M = 30$), the percent of re-executed is low, this is because replication is more effective since the replicas (i.e., space redundancy) can be distributed on different processors; on the other hand, if the number of tasks is much higher (e.g., $N = 200$ and $M = 10$, or $N = 100$ and $M = 5$), the percent of re-executed is high, this is because re-execution (time redundancy) is more effective since the space resources is very limited.

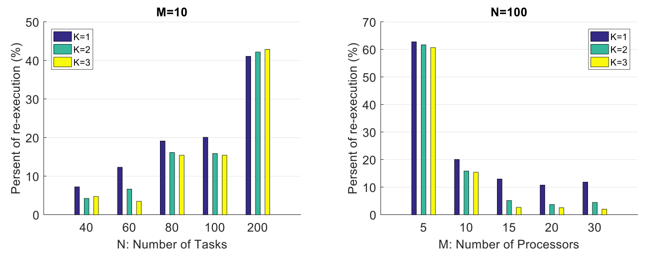


Fig. 9. Percent of re-executed tasks with changing ratio of the number of tasks to that of processors.

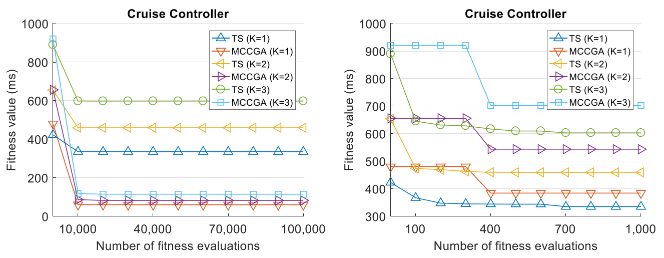


Fig. 10. Evolutionary curves of the average fitness value with the number of fitness evaluations using MCCGA and TS on the cruse controller system.

F. Case Studies

A CCS [39] is considered as a case study for the proposed method. The task graph of CCS has 20 tasks, and four heterogeneous processors are assumed on the target multiprocessor platform in this experiment. The WCET of each task on each processor is randomly decided in the range of 1∼1.5 times its standard execution time which has been given in [39]. As an individual-based metaheuristic, TS is commonly considered as a very efficiency search method for fault-tolerant multiprocessor system design [6], [28]–[30]. For a fair comparison, a customized TS is included here as the baseline algorithm. The key idea is the use of two vectors, *Tabu* and *Wait*, to implement the selective history during the search. Specifically, the purpose of the *Tabu* vector is to avoid revisiting the recent past solutions, while the purpose of the *Wait* vector is to diversify the search, more details can be found in [28]. Fig. 10 shows the evolutionary curves of the average fitness value with the number of fitness evaluations using MCCGA and TS on CCS. It should be noticed that it is possible to accept bad solutions during the TS process, we record and show the best historical solution until the current iteration in the figures for TS. It can be seen that MCCGA can converge to much better solutions, if we have enough fitness evaluation times (i.e., 10 000), as shown in Fig. 10(left). But, if given a very limited fitness evaluation times (i.e., 1000), TS can obtain better results, as shown in Fig. 10(right). As this paper focuses on offline DSE of fault-tolerant multiprocessor systems, we think MCCGA is a better candidate method.

VI. Conclusion

This paper aims at improving the efficiency of the EA-based DSE method for fault-tolerant multiprocessor system design. In order to reduce the redundancy in genotype-to-phenotype mappings in the previous EA-based methods, we have proposed to separate the mapping exploration explicitly. In order to avoid the problem of long task mapping chromosomes, we have proposed to employ the CCEA framework. In addition, in order to accelerate the DSE process further, we have designed new GRLS operators and embedded them in the CCEA framework. The effectiveness and efficiency of the proposed strategies are demonstrated by extensive experiments. The experimental results show that using the proposed DSE method, MCCGA, it is possible to obtain 1.75×∼2.50× better results on the benchmarks of 300 tasks and 30 processors, compared to the state-of-the-art.

A limitation of this paper is that we only consider replicating a task or re-executing a task. Our future work is to extend the proposed method to include the combination of replication and re-execution, and other hardware and software fault tolerance techniques. Besides, we will also consider multiple objectives in our future work, such as reliability, performance, and temperature [43].

References

[1] J. Henkel *et al.*, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *Proc. 50th Annu. Design Autom. Conf.*, May 2013, pp. 1–10.
[2] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. 50th Annu. Design Autom. Conf.*, Austin, TX, USA, May 2013, pp. 1–10.
[3] J. Henkel *et al.*, "Design and architectures for dependable embedded systems," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2011, pp. 69–78.
[4] M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-phase low-energy N-modular redundancy for hard real-time multi-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1497–1510, May 2016.
[5] N. Kandasamy, J. P. Hayes, and B. T. Murray, "Transparent recovery from intermittent faults in time-triggered distributed systems," *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 113–125, Feb. 2003.
[6] M. Salehi *et al.*, "Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 7, pp. 2426–2437, Jul. 2016.
[7] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 389–402, Mar. 2009.
[8] C. Bolchini and A. Miele, "Reliability-driven system-level synthesis for mixed-critical embedded systems," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2489–2502, Dec. 2013.
[9] S.-H. Kang *et al.*, "Reliability-aware mapping optimization of multi-core systems with mixed-criticality," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2014, pp. 1–4.
[10] S.-H. Kang *et al.*, "Static mapping of mixed-critical applications for fault-tolerant MPSoCs," in *Proc. 51st Annu. Design Autom. Conf.*, Jun. 2014, pp. 1–6.
[11] A. Jhumka, S. Klaus, and S. A. Huss, "A dependability-driven system-level design approach for embedded systems," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2005, pp. 372–377.
[12] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, "Reliability-aware system synthesis," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Apr. 2007, pp. 409–414.
[13] F. Reimann *et al.*, "Symbolic voter placement for dependability-aware system synthesis," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2008, pp. 237–242.
[14] J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2011, pp. 247–256.
[15] J. Huang, K. Huang, A. Raabe, C. Buckl, and A. Knoll, "Towards fault-tolerant embedded systems with imperfect fault detection," in *Proc. 49th Annu. Design Autom. Conf.*, San Francisco, CA, USA, Jun. 2012, pp. 188–196.

[16] C. Bolchini and A. Miele, "Reliability-driven system-level synthesis of embedded systems," in *Proc. Int. Symp. Defect Fault Tolerance Very Large Scale Integr. Syst.*, Oct. 2010, pp. 35–43.

[17] C. Bolchini, A. Miele, and C. Pilato, "Combined architecture and hardening techniques exploration for reliable embedded system design," in *Proc. Great Lakes Symp. VLSI*, 2011, pp. 301–306.

[18] C. Bolchini, P. L. Lanzi, and A. Miele, "A multi-objective genetic algorithm framework for design space exploration of reliable FPGA-based systems," in *Proc. Congr. Evol. Comput.*, Jul. 2010, pp. 1–8.

[19] P. Stralen and A. Pimentel, "A SAFE approach towards early design space exploration of fault-tolerant multimedia MPSoCs," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2012, pp. 393–402.

[20] J. Gan, P. Pop, F. Gruian, and J. Madsen, "Robust and flexible mapping for real-time distributed applications during the early design phases," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Dresden, Germany, Mar. 2012, pp. 935–940.

[21] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Dresden, Germany, Mar. 2014, pp. 1–6.

[22] F. Khosravi, F. Reimann, M. Glaß, and J. Teich, "Multi-objective local-search optimization using reliability importance measuring," in *Proc. 51st Annu. Design Autom. Conf.*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

[23] J. He and X. Yao, "Towards an analytic framework for analysing the computation time of evolutionary algorithms," *Artif. Intell.*, vol. 145, nos. 1–2, pp. 59–97, Apr. 2003.

[24] Y. Yu and Z.-H. Zhou, "A new approach to estimating the expected first hitting time of evolutionary algorithms," *Artif. Intell.*, vol. 172, no. 15, pp. 1809–1832, Oct. 2008.

[25] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.

[26] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.

[27] B. P. Dave and N. K. Jha, "COFTA: Hardware-software co-synthesis of heterogeneous distributed embedded systems for low overhead fault tolerance," *IEEE Trans. Comput.*, vol. 48, no. 4, pp. 417–441, Apr. 1999.

[28] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Reliability-aware co-synthesis for embedded systems," *J. VLSI Signal Process. Syst.*, vol. 49, no. 1, pp. 87–99, 2007.

[29] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2005, pp. 864–869.

[30] V. Izosimov, I. Polian, P. Pop, P. Eles, and Z. Peng, "Analysis and optimization of fault-tolerant embedded systems with hardened processors," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Apr. 2009, pp. 682–687.

[31] A. Lifa, P. Eles, Z. Peng, and V. Izosimov, "Hardware/software optimization of error detection implementation for real-time embedded systems," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2010, pp. 41–50.

[32] T. Schwarzer *et al.*, "Symmetry-eliminating design space exploration for hybrid application mapping on many-core architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 297–310, Feb. 2018.

[33] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 193–208, Apr. 2014.

[34] Y. Mei, X. Li, and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 435–449, Jun. 2014.

[35] S. He *et al.*, "Cooperative co-evolutionary module identification with application to cancer disease module discovery," *IEEE Trans. Evol. Comput.*, vol. 20, no. 6, pp. 874–891, Dec. 2016.

[36] Y. Yuan and H. Xu, "Multiobjective flexible job shop scheduling using memetic algorithms," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 336–353, Jan. 2015.

[37] L. Tersi, S. Fantozzi, and R. Stagni, "Characterization of the performance of memetic algorithms for the automation of bone tracking with fluoroscopy," *IEEE Trans. Evol. Comput.*, vol. 19, no. 1, pp. 19–30, Feb. 2015.

[38] Á. Rubio-Largo, M. A. Vega-Rodríguez, and D. L. González-Álvarez, "A hybrid multiobjective memetic metaheuristic for multiple sequence alignment," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 499–514, Aug. 2016.

[39] B. Yuan, B. Li, T. Weise, and X. Yao, "A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures," *IEEE Trans. Evol. Comput.*, vol. 18, no. 6, pp. 846–859, Dec. 2014.

[40] B. Yuan, B. Li, H. Chen, and X. Yao, "Defect-and variation-tolerant logic mapping in nanocrossbar using bipartite matching and memetic algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 9, pp. 2813–2826, Sep. 2016.

[41] N. Kandasamy, J. P. Hayes, and B. T. Murray, "Dependable communication synthesis for distributed embedded systems," *Rel. Eng. Syst. Safety*, vol. 89, no. 1, pp. 81–92, 2005.

[42] M. Glaß, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich, "Symbolic reliability analysis and optimization of ECU networks," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2008, pp. 158–163.

[43] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2011, pp. 189–196.

Bo Yuan (M'15) received the B.Sc. and Ph.D. degrees in electronic information science and technology from the University of Science and Technology of China, Hefei, China, in 2009 and 2014, respectively.

From 2012 to 2013, he was a visiting student with the School of Computer Science, University of Birmingham, Birmingham, U.K. He is currently a Research Assistant Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. His current research interests include evolutionary computation, electronic design automation, and machine learning.

Huanhuan Chen (SM'16) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 2004, and the Ph.D. degree in computer science from the University of Birmingham, Birmingham, U.K., in 2008.

He is currently a Full Professor with the School of Computer Science and Technology, USTC. His current research interests include neural networks, Bayesian inference, and evolutionary computation.

Dr. Chen was a recipient of the 2015 International Neural Network Society Young Investigator Award, the 2012 IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award, the IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, and the 2009 British Computer Society Distinguished Dissertations Award. He is an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS and the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE.

Xin Yao (F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the Ph.D. degree from USTC, in 1990.

He is currently a Chair Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China, and also a part-time Professor with the School of Computer Science, University of Birmingham, Birmingham, U.K. His current research interests include evolutionary computation and ensemble learning.

Dr. Yao was a recipient of the 2001 IEEE Donald G. Fink Prize Paper Award, the 2010 and 2016 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Awards, the 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, and several other best paper awards. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008, and the President of the IEEE Computational Intelligence Society (CIS) from 2014 to 2015. He is a Distinguished Lecturer of the IEEE CIS.