Chinese Journal of Electronics
Vol.23, No.2, Apr. 2014

Evolutionary Generation Approach of Test Data for Multiple Paths Coverage of Message-passing Parallel Programs*

TIAN Tian and GONG Dunwei

(*School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, Jiangsu 221116, China*)

Abstract — Test data generation, the premise of software testing, has attracted scholars in the software engineering community in recent years. Influenced by task partitioning, process scheduling, and network delays, parallel programs are executed in a non-deterministic way, which makes test data generation of parallel programs different from that of serial programs in essence. This paper investigated the problem of generating test data for multiple paths coverage of message-passing parallel programs. A mathematical model of the above problem was built based on each given path and its equivalent ones. It was solved by using a genetic algorithm to generate all desired data in one run. The proposed method was applied to five benchmark programs, and compared with the existing methods. The experimental results show that the proposed method greatly shortens the number of iterations and time consumption without reducing the coverage rate.

Key words — Software testing, Parallel program, Multiple paths, Test data, Genetic algorithm.

I. Introduction

There are various complicated problems to be effectively solved in the national economies, such as image processing, weather forecasting, and molecular dynamic simulation. Taking high-performance parallel machines or computer clusters as the hardware platform, parallel computing is of essence to reduce the solution time and improve the solution accuracy, thus becoming one main way to solve problems.

Generally speaking, there are three approaches to developing a parallel program: (1) automatically parallelizing a sequential code; (2) adopting a parallel programming language; (3) extending traditional programming languages by a message-passing environment. Previous practices have shown that the last approach can produce an efficient program on the premise of a few loads of a programmer, so it has been the most commonly used way to design a parallel program. To date, typical message-passing environments for extending traditional programming languages include Message passing interface (MPI)[1], Parallel virtual machine (PVM), CM, and Express. Among them, the most notable and important one is MPI that has an excellent compatibility and several free versions, and has gained great attention in the last decade[2]. Therefore, the message passing parallel program extending MPI was the focus of this study.

Currently, most work on the testing of parallel programs focuses on communication sequences to detect deadlocks or unintentional races, or improper usages of the Application program interface (API). On the premise of normal communication with neither deadlocks nor unintentional races, the testing of parallel programs to improve their credibility has become an urgent problem.

The core of software testing is to generate effective test data with a rapid speed. There are various criteria for generating test data, for example, statement coverage, branch coverage, condition coverage and decision/condition coverage, all of which amount to path-oriented test data generation. Therefore, among various criteria, path coverage has become the most commonly used criterion[3]. However, previous work on test data generation is mainly suitable for serial programs.

In view of this, this study focused on the problem of path coverage for the message passing parallel programs. It is known that a complicated program often has many paths. It will undoubtedly improve the efficiency of generating test data if test data that cover multiple target paths are simultaneously generated. To this end, the problem of generating test data that cover multiple paths was investigated in this study. According to the non-deterministic behaviors of parallel programs, the problem of generating test data for multiple paths coverage was formulated as a multi-objective optimization problem, and an appropriate fitness function was designed when employing a genetic algorithm to solve the above problem, so as to improve the efficiency of generating test data.

II. Related Work

*Manuscript Received Jul.2013; Accepted Nov.2013. This work is supported by the National Natural Science Foundation of China (No.61375067, No.61075061, No.61203304), the Natural Science Foundation of Jiangsu Province (No.BK2012566, No.BK2010187), Graduate Student Research Innovation Program of Jiangsu Province College (No.CXZZ11_0292) and the Fundamental Research Funds for the Central Universities(No.2013XK09).

The testing of parallel programs has attracted more and more scholars in software engineering community in recent years. Krammer *et al.* checked for improper usages of MPI[4]. Vetter *et al.* developed Umpire, a verification tool for MPI programs, and used it to detect such defects as deadlocks, mismatched collective operations, and resource exhaustion[5]. Lei *et al.* presented a method of reachability testing to guarantee each synchronous partial order sequence to be executed exactly once so as to detect deadlocks and assertion violation[6]. Souza *et al.* also presented a method of combining coverage criteria and reachability testing to achieve low testing cost[7].

Godefroid presented a model-checking based method that controls thread scheduling, and explores all states of parallel programs. The number of interactions inside parallel programs, however, will exponentially grow along with its size, resulting in the problem of state space exploration[8]. Partial order reduction can be used to solve this problem based on the fact that when there is no interaction between threads, different execution sequences have the same state. So, if an execution sequence finds a defect, such as deadlock or data race, its equivalent sequences can also detect this defect. Since the static techniques are incapable of handling pointers, Flanagan *et al.* proposed a method of dynamic partial order reduction[9]. Although partial order reduction reduces the search space, exhaustive searching cannot be realized[10]. Vakkalanka *et al.* presented a model-checking tool, called ISP, in which all processes of a given MPI program are executed in control of an interleaving scheduler. ISP incorporated with Partial order reduction avoiding elusive interleaving (POE) algorithm and dynamic rewriting based scheduling of MPI function call interleaving guarantees checking for all deadlocks and local assertion violation[11]. For shared memory parallel programs, Sun *et al.* presented their testing criteria by extending those of serial programs[12].

It can be seen that previous work concerning the testing of parallel programs concentrates mainly on problems arising from concurrent execution, and tests communication sequences for detecting deadlocks, or resource competition, or incorrect usage of API. On the precondition of correct communication, there are no studies on how to generate test data to improve the reliability of parallel programs.

There exists some work to study the coverage testing of message-passing parallel programs. Yang *et al.* applied all-du-path criterion to shared memory or message-passing parallel programs, and developed an algorithm to find all-du-paths for a program under test[13]. Souza *et al.* defined several testing criteria based on control and communication flows as well as the data and message-passing flows, respectively. In addition, a tool was developed to support the proposed criteria[14,15]. However, the corresponding methods of generating test data to meet these criteria have not yet been presented.

Our study is different from previous methods in the following three aspects. First, we focus on generating test data covering paths rather than investigating coverage criteria. Second, instead of seeking paths, we set multiple paths as the target ones in advance. Finally, we study effective methods of generating test data by taking non-deterministic behaviors of parallel programs into account.

Software testing is premised on test data generation. Generating desired test data by adopting appropriate methods can improve the efficiency of software testing. Heuristics generate test data with an uncertain process, such as methods of generating test data based on genetic algorithms. Generating test data of complex software using genetic algorithms has been one of research focuses in the community of software engineering in recent years.

To solve the problem of path coverage, the first work in which genetic algorithms were used to automatically generate test data, was done by Xanthakis *et al.* in 1992[16]. Ahmed[17], Bueno[18], and Watkins[19] *et al.* employed genetic algorithms to obtain test data meeting the requirement of path coverage. Lin *et al.* defined the fitness function by employing the extended Hamming distance to solve the problem of comparing paths with the same nodes but different orders[20]. Xie *et al.* presented three approaches for designing the fitness function based on the similarity of paths for generating test data that cover a given path[21]. McMinn *et al.* simultaneously searched for test data for multiple paths by dedicated genetic algorithms, thus improving the efficiency of generating test data[22]. Aiming at the problem of multiple paths coverage, Gong *et al.* presented methods of generating test data by using genetic algorithms under different hardware environments[23,24].

The above methods of generating test data are mainly suitable for serial programs. Unlike serial programs, parallel programs possess characteristics such as concurrency, communication, and synchronization. The above methods, however, obviously omit them. Therefore, the efficiency of generating test data will be drastically reduced if these methods are directly applied to the testing of parallel programs. Consequently, it is considerably necessary to study effective methods of generating test data according to the characteristics of message-passing parallel programs.

Tian *et al.* preliminarily studied the problem of generating test data for path coverage of parallel programs, and proposed a method of generating test data using genetic algorithms[25]. However, this method generates test data that cover only one path in one run of genetic algorithms. Clearly, the efficiency of the above method in generating test data will be drastically reduced when there are multiple paths to be covered.

In view of this, the problem of generating test data for covering multiple paths of message-passing parallel programs was investigated in this study. According to the characteristics of parallel programs, the above problem was first converted to a multi-objective optimization problem, and then the genetic algorithm was adopted to solve the converted problem. The efficiency of the proposed method in generating test data was evaluated by several benchmark parallel programs.

III. Problem Formulation

1. Basic concepts

A parallel program is denoted as $S = \{S^0, S^1, ..., S^{N_s-1}\}$, where $S^i(i = 0, 1, ..., N_s - 1)$ is the i-th process and N_s is the number of processes. The input vector of S is assumed as $\boldsymbol{x} = \{x_1, x_2, ..., x_{N_x}\}$, where $x_j(j = 1, 2, ..., N_x)$is the j-th input component of $\boldsymbol{x}$, and its range is D_{x_j}, then the range of $\boldsymbol{x}$ is $D = D_{x_1} \times D_{x_2} \times ... \times D_{x_{N_x}}$.

Node Considering the i-th process, S^i, a node is a statement block of S^i, satisfying that all statements contained in the block are either executed or not. A node may be the condition of a loop or a branch statement, or one/several successive statement(s), or a sending/receiving statement. The j-th node of S^i is denoted as n^i_j.

Path It is a sequential of nodes in S when $S = \{S^0, S^1, ..., S^{N_s-1}\}$ is run under $\boldsymbol{x} \in D$, and denoted as p. More specifically, the sequential of nodes in S^i is denoted as p^i, the number of nodes contained in p^i is called the path length of p^i, and denoted as $|p^i|$. Then the path traversed by $\boldsymbol{x}$, p, is composed by N_s process paths, and denoted as $p(\boldsymbol{x}) = p^0p^1...p^{N_S-1}$.

Equivalent path[25] Considering two paths, $p_{j,0} = p^0_{j,0}p^1_{j,0}...p^{N_S-1}_{j,0}$ and $p_{j,1} = p^0_{j,1}p^1_{j,1}...p^{N_S-1}_{j,1}$, if different sequences of nodes between $p^i_{j,0}$ and $p^i_{j,1} (i = 0, 1, ..., N_S - 1)$ are only caused by different message-passing orders, $p_{j,1}$ is called an equivalent path of $p_{j,0}$. It is clear that $p_{j,0}$ may have several equivalent paths which constitute an equivalent path set, denoted as $\tilde{p}_{j,0}$.

Because of the space limitation, the detailed method of seeking for equivalent paths is not given here. For interested readers, please refer to Ref.[25] for details.

For l given paths, $p_{1,0}, p_{2,0}, ..., p_{l,0}$, their equivalent path sets, denoted as $\tilde{p}_{1,0} = \{p_{1,1}, p_{1,2}, ..., p_{1,|\tilde{p}_{1,0}|}\}, \tilde{p}_{2,0} = \{p_{2,1}, p_{2,2}, ..., p_{2,|\tilde{p}_{2,0}|}\}, ..., \tilde{p}_{l,0} = \{p_{l,1}, p_{l,2}, ..., p_{l,|\tilde{p}_{l,0}|}\}$,can be obtained by using the above method, where $p_{j,k} \in \tilde{p}_{j,0}, j = 1, 2, ..., l, k = 1, 2, ..., |\tilde{p}_{j,0}|$ is the k-th equivalent path of $p_{j,0}$, and $|\tilde{p}_{j,0}|$ is the number of paths contained in $\tilde{p}_{j,0}$.

2 Problem formulation

Considering l given paths, $p_{1,0}, p_{2,0}, ..., p_{l,0}$, and all their equivalent paths, there are $\sum_{j=1}^{l} |\tilde{p}_{j,0}| + l$ target paths. For each target path, it is necessary to construct one function so as to formulate the problem of generating test data to cover the target path. Therefore, $\sum_{j=1}^{l} |\tilde{p}_{j,0}| + l$ functions corresponding to these target paths should be constructed so as to formulate the problem considered in this study.

Considering path $p_{j,k} = p^0_{j,k}p^1_{j,k}....p^{N_S-1}_{j,k}, j = 1, 2, ..., l, k = 0, 1, ..., |\tilde{p}_{j,0}|$ and denoting the path traversed by $\boldsymbol{x}$ as $p(\boldsymbol{x}) = p^0p^1...p^{N_S-1}$, then the similarity between p^i and $p^i_{j,k}$ belonging to $p(\boldsymbol{x})$ and $p_{j,k}$, respectively, denoted as $n(p^i, p^i_{j,k})$, can be represented as:

$$n(p^i, p^i_{j,k}) = \frac{|p^i \cap p^i_{j,k}|}{\max\{\,|p^i|\,, \left|p^i_{j,k}\right|\,\}} \tag{1}$$

where $|p^i \cap p^i_{j,k}|$ is the number of successively same nodes of p^i and $p^i_{j,k}$ from the first one.

Then the similarity between $p(\boldsymbol{x})$ and $p_{j,k}$ is:

$$f_{j,k}(\boldsymbol{x}) = \frac{1}{N_S} \sum_{i=0}^{N_S-1} n(p^i, p^i_{j,k}) \tag{2}$$

It is clear that the greater value of $f_{j,k}(\boldsymbol{x})$, the closer $p(\boldsymbol{x})$ comes to $p_{j,k}$, and $f_{j,k}(\boldsymbol{x})$ is equal to one, if and only if $p(\boldsymbol{x})$ is the target path, $p_{j,k}$. Consequently, the mathematical model of the problem considered in this study can be formulated as follows:

$$\begin{cases} \max\{f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x})\}, \\ \max\{f_{2,0}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), ..., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x})\}, \\ \vdots \\ \max\{f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l.0}|}(\boldsymbol{x})\} \\ \text{s.t. } \boldsymbol{x} \in D \end{cases} \tag{3}$$

where the j-th objective, $\max\{f_{j,0}(x), f_{j,1}(x), ..., f_{j,|\tilde{p}_{j,0}|}(x)\}, j = 1, 2, ..., l$, corresponds to the given path, $p_{j,0}$, and its equivalent paths, $p_{j,k} \in \tilde{p}_{j,0}, k = 1, ..., |\tilde{p}_{j,0}|$.

IV. Method of Evolutionary Generation of Test Data for Multiple Paths Coverage (EGMP)

A genetic algorithm was adopted to solve the above model so as to generate test data covering multiple paths in this section. Except for designing the fitness function, others aspects are the same as those of the algorithm for generating test data for serial programs, and their details can be found in Ref.[23]. Therefore, the fitness function is required to specifically design according to the characteristics of parallel programs.

1. Fitness function

According to Eq.(2), $\sum_{j=1}^{l} |\tilde{p}_{j,0}| + l$ objective function values of $\boldsymbol{x}$, $f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), f_{2,2}(\boldsymbol{x}), ..., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x}), ..., f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})$,can be obtained by executing the program under test S with $\boldsymbol{x}$.

On the one hand, if $p_{j,k}$ can be traversed by $\boldsymbol{x}$, max $\{f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), f_{2,0}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), ..., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x}), ..., f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})\} = f_{j,k}(\boldsymbol{x}) = 1$. On the other hand, if $\max\{f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), f_{2,0}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), ..., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x}), ..., f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})\} = 1$, there must exist j and k such that $f_{j,k}(\boldsymbol{x}) = 1$. In other words, the necessary and sufficient condition for one of target paths to be covered by $\boldsymbol{x}$ is that $\max\{f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), f_{2,0}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), ..., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x}), ..., f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})\} = 1$.

This means that $\max\{f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), f_{2,0}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), ..., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x}), ..., f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})\}$ can be used to evaluate whether $\boldsymbol{x}$ satisfies the coverage requirements of $\sum_{j=1}^{l} |\tilde{p}_{j,0}| + l$ target paths or not. When $f_{j,k}(\boldsymbol{x}) = 1$, $f_{j',k'} < 1$, where $j' = j \wedge k' \neq k$ or $j' \neq j$, since only one path can be traversed by x.

An individual corresponding to test datum $\boldsymbol{x}$ is also denoted as $\boldsymbol{x}$ without causing confusion. Based on the above analysis, the fitness function of $\boldsymbol{x}$, denoted as $F(\boldsymbol{x})$, can be formulated as follows:

$$\begin{aligned} F(\boldsymbol{x}) = \max\{&f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), \\ &f_{2,0}(\boldsymbol{x}), f_{2,1}(\boldsymbol{x}), .., f_{2,|\tilde{p}_{2,0}|}(\boldsymbol{x}), ..., \\ &f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})\} \end{aligned} \tag{4}$$

where $f_{j,k}(\boldsymbol{x}), j = 1, 2, ..., l, k = 0, 1, ..., |\tilde{p}_{j,0}|$ is defined as Eq.(2).

It is clear that the greater value of $F(\boldsymbol{x})$, the greater the similarity between $p(\boldsymbol{x})$ and one of the target paths, the better of $\boldsymbol{x}$. When the fitness of $\boldsymbol{x}$ satisfies that $F(\boldsymbol{x}) = f_{j,k}(\boldsymbol{x}) = 1$, suggesting that the test datum covering $p_{j,0}$ or its equivalent paths has been found, the fitness function, *i.e.*, Eq.(4), will be changed into:

$$\begin{aligned}\max\{&f_{1,0}(\boldsymbol{x}), f_{1,1}(\boldsymbol{x}), ..., f_{1,|\tilde{p}_{1,0}|}(\boldsymbol{x}), ..., f_{j-1,0}(\boldsymbol{x}),\\ &f_{j-1,1}(\boldsymbol{x}), ..., f_{j-1,|\tilde{p}_{j-1,0}|}(\boldsymbol{x}), f_{j+1,0}(\boldsymbol{x}), f_{j+1,1}(\boldsymbol{x}),\\ &..., f_{j+1,|\tilde{p}_{j+1,0}|}(\boldsymbol{x}), ..., f_{l,0}(\boldsymbol{x}), f_{l,1}(\boldsymbol{x}), ..., f_{l,|\tilde{p}_{l,0}|}(\boldsymbol{x})\}\end{aligned} \tag{5}$$

2. Proposed algorithm

For l given paths, the steps of generating test data covering them based on the genetic algorithm are shown as Fig.1. In the algorithm initialization, the equivalent paths of all given paths can be obtained by the method in subsection III.1, and together with these given paths, the uncovered path set is formed (lines 1-3). In each iteration, the traversed path is obtained by executing S with a decoded individual as the input of S (lines 6-7). If the traversed path belongs to the uncovered path set, the individual is output as one desired test datum, and the corresponding given path and its equivalent paths are deleted from the uncovered path set (lines 8-10). At this time, if the uncovered path set is empty, the algorithm is ended (lines 11-13); otherwise, the fitness function is adjusted according to the traversed path (lines 14-15). Finally, the fitness of each individual is calculated according to Eq.(4) and the genetic operators are performed to generate offspring (lines 17-20). The above process is repeated until all desired test data are found.

```
Begin
1 Obtain equivalent paths of each given path and instrumented program;
2 Initialize the population;
3 SET is composed of all given paths and their equivalent paths;
4 While the maximal number of iterations has not been reached
5       For each individual
6           Decode it as the input of the program;
7           Execute the program and obtain the traversed path;
8           If the traversed path is one of the uncovered target paths $p_{j,k}$
9               Output the individual and the traversed path;
10              Delete $p_{j,0}, p_{j,1}, ..., p_{j,|\tilde{p}_{j,0}|}$ from SET;
11              If SET= $\varnothing$
12                  return;
13              Endif
14              Adjust the fitness Eq.(4);
15          Endif
16      Endfor
17      Calculate the individual's fitness according to Eq.(4);
18      Selection;
19      Crossover;
20      Mutation;
21 Endwhile
End
```

Fig. 1. Pseudo-code of proposed algorithm

V. Experiments

The proposed method was applied to several parallel programs, and compared with the method in Ref.[25] and the random method to evaluate its effectiveness.

The configuration of the PC cluster in the experiments was as follows. Hardware of the calculation nodes was configured as 2 ×Intel Xeon E5650 CPU, 24GB memory, 1×160GB RAID hard disks, and the thousand trillion ether network. The software configuration was Linux operation system, GridView management software, and MPI+C programming language.

Five programs were chosen as test objects. These programs are used in Refs.[7, 14, 25] for evaluating the performance of different methods of testing parallel programs. S_1 was employed to multiply two matrices, and judge the relationships among elements of one matrix. S_2 was a function of the industrial testing program and implemented character retrieval. S_3 was converted from a serial benchmark program and judged the maximum of the input data and the triangle types. S_4 was used to calculate the greatest common divisor. S_5 solved the liner equation and judged the maximum of the coefficient matrix.

Binary coding was used to encode an individual, and the genetic operators were roulette-wheel selection, one-point crossover, and one-point mutation with their probabilities of 0.9 and 0.3, respectively. To evaluate the performance of different methods in generating test data, the following three indicators were employed: the number of iterations, the time consumption, and the coverage rate, where the coverage rate is the ratio of the number of the paths that are traversed by the generated test data to the total number of the given paths.

To reduce the negative impact of stochastic factors on a method, for each program, each method was independently run 30 times. The experimental results of each run were recorded, and the average number of iterations, the average time consumption, and the coverage rate were then calculated. To further evaluate whether the difference between the proposed method and the comparative one was significant in the above indicators or not, the Mann-Whitney U-test was employed with a significant level of 0.05. If the statistical value of an indicator was smaller than 0.05, there exists a significant difference between these two methods in this indicator.

For S_1, S_2, S_3, S_4, and S_5, the number of given paths was three, four, five, two and five, respectively, and the corresponding population size was set to 50, 200, 50, 200 and 200. The largest number of iterations was 10,000 for all these programs. Table 1 reports the experimental results.

Table 1 demonstrates that for all these five programs, (1) the proposed method reduces the number of iterations by 50%, 62%, 56%, 43% and 71% of the method in Ref.[25], by 59%, 71%, 65%, 54% and 80% of the random method, respectively; (2) the proposed method reduces the time consumption by 46%, 61%, 60%, 55% and 53% of the method in Ref.[25], by 51%, 68%, 60%, 14% and 69% of the random method, respectively; (3) the statistical test results further suggest that the proposed method has significant differences with the comparative methods in these two indicators; (4) the proposed method significantly improves the path coverage rate by 40%, 53%, 27%, 40% and 37% of the random method.

In addition, the method in Ref.[25] is slightly superior to the proposed one for S_1 and S_4, statistical analysis, however, indicates that they have no significant difference; but for S_2,S_3 and S_5, the proposed method has a higher coverage rate of 27%, 13% and 16% than the comparative method, and their

Table 1. Indicator values for other programs

Program	Method	# of iterations			Time consumption (s)			Path coverage rate (%)		
		Average	variance	U	average	variance	U	average	variance	U
S_1	EGMP	8010.3	4901445.0		9.829	7.857		80.0	859.3	
	Method in Ref.[25]	16141.0	22851483.0	0	18.262	26.423	0	81.1	346.9	0.596
	Random Method	19683.2	72226759.0	0	20.229	78.460	0	40.0	1066.7	0
S_2	EGMP	8108.7	6846427.0		30.046	93.364		85.8	153.5	
	Method in Ref.[25]	21384.0	12851166.4	0	77.651	165.214	0	59.1	145.1	0
	Random Method	28232.4	91368224.0	0	92.825	1003.787	0	32.5	672.9	0
S_3	EGMP	8669.7	7070268.0		10.968	11.974		78.0	516.0	
	Method in Ref.[25]	19730.1	110681748.0	0	27.356	111.898	0	65.0	451.6	0.018
	Random Method	24802.2	110015243.0	0	27.146	131.199	0	51.3	471.6	0
S_4	EGMP	7701.7	8603563.0		72.278	757.969		71.7	947.2	
	Method in Ref.[25]	13619.3	22317611.0	0	161.932	3326.304	0	73.3	1122.2	0.733
	Random Method	16628.6	11259119.0	0	129.330	708.187	0	31.7	580.6	0
S_5	EGMP	5368.2	8173218.0		35.380	352.672		96.7	135.6	
	Method in Ref.[25]	18613.8	55147740.0	0	75.571	1156.147	0	80.7	332.9	0
	Random Method	26592.3	82620392.0	0	112.668	1498.057	0	60.0	533.3	0

difference is significant.

The main reason of causing the above results is as follows. For a path with a low complexity, the method in Ref.[25] only spends less iterations and time to generate desired test data, whereas more iterations and time are required to generate test data that cover a higher complexity path. Sometimes, the desired test data cannot yet be found even the largest number of iterations has reached. Taking advantage of the fact that an individual can be shared by multiple paths during the evolution, the proposed method simultaneously considers all uncovered paths, which makes it easy to generate test data that cover paths with a low complexity in the early evolution of the population. In addition, along with a part of desired test data being found, the problem to be solved is constantly simplified, which is helpful to seek for other desired test data.

To sum up, the proposed method can effectively generate test data covering multiple paths. To be more specific, the proposed method expends less time and number of iterations than the existing methods on the premise of not reducing the coverage rate.

VI. Conclusion

Along with the widespread applications of parallel programs, the testing of parallel programs has attracted various scholars in the software engineering community, and achieved some valuable results. Unfortunately, there is a lack of effective methods of generating test data for multiple paths coverage.

The problem of generating test data that cover multiple paths of message-passing parallel programs was studied. The above problem was first formulated as a multi-objective optimization problem, and then a method based on genetic algorithms was proposed to solve it, which can synthesize multiple test data to cover the given paths or their equivalent ones in one run. The proposed method was applied to five benchmark parallel programs, and compared with the existing methods. The experimental results showed that the proposed method spends less number of iterations and time without reducing the path coverage rate, thus improving the efficiency of testing parallel programs.

It is worth noting that a parallel program often contains many paths, whereas only several paths were chosen as target ones in this study. If the proposed method is used to generate test data that cover many paths, an optimization problem with many objectives will be formulated, which is very difficult to solve. Therefore, seeking appropriate methods of decomposing the above model is a topic to be addressed.

References

[1] M. Snir, S. Otto, H. Steven, D. Walker, J. Dongarra, MPI: The Complete Reference, MIT Press, Cambridge, USA. 1996.

[2] G.L. Chen, H. An, L. Chen, Q.L. Zheng, J.L. Shan, Parallel Algorithms, Higher Education Press, Beijing, China, 2004.

[3] J.H. Shan, Y. Jiang, P. Sun, "Survey on path-wise automatic generation of test data", *Acta Electronica Sinica*, Vol.32, No.1, pp.134–145, 2004.

[4] B. Krammer, M. Resch, "Correctness checking of MPI one-sided communication using Marmot", *Proc. of 13th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Inte cxrface*, Bonn, Germany, Lecture Notes in Computer Science, Vol.4192, pp.105–114, 2006.

[5] J.S. Vetter, B.R. Supinski, "Dynamic software testing of MPI applications with umpire", *Proc. of Supercomputing ACM/ IEEE Conference*, Dallas, USA, pp.79, 2000.

[6] J. Lei, R. Carver, "Reachability testing of concurrent programs", *IEEE Transactions on Software Engineering*, Vol.32, No.6, pp.382–403, 2006.

[7] S.R.S. Souza, P.S.L. Souza, M.C.C. Machado, "Using coverage and reachability testing to improve concurrent program testing quality", *Proc. of 23rd International Conference on Software Engineering and Knowledge Engineering*, Miami Beach, USA, pp.207–212, 2011.

[8] P. Godefroid, "Model checking for programming languages using verisoft", *Proc. of 24th Symposium on Principles of Programming Languages*, Paris, France, pp.174–186, 1997.

[9] C. Flanagan, P. Godefroid, "Dynamic partial-order reduction for model checking software", *Proc. of 32nd Symposium on Principles of Programming Languages*, New Orleans, USA, pp.110–121, 2005.

[10] S. Koushik, "Effective random testing of concurrent programs", *Proc. of 22nd International Conference on Automated Software Engineering*, Atlanta, Georgia, pp.323–333, 2007.

[11] S. Vakkalanka, M. DeLisi, G. Gopalakrishnan, R.M. Kirby, R. Thakur, W. Gropp, "Implementing efficient dynamic formal verification methods for MPI programs", *Proc. of 15th European PVM/MPI Users' Group Meeting on Recent Advances*

in Parallel Virtual Machine and Message Passing Interface, Dublin, Ireland, Lecture Notes in Computer Science, Vol.5205, pp.248–256, 2008.

[12] C.Y. Sun , L.P. Lori, "All-uses testing of shared memory parallel programs", *Software Testing, Verification and Reliability*, Vol.13, No.1, pp.3–24, 2003.

[13] C.S.D. Yang, A.L. Souter, L.L. Pollock, "All-du-path coverage for parallel programs", *Proc. Of ACM International Symposium on Software Testing and Analysis*, Clearwater Beach, USA, pp.153–162, 1998.

[14] S.R.S. Souza, S.R. Vergilio, P.S.L. Souza, "Structural testing criteria for message-passing parallel programs", *Concurrency and Computation: Practice and Experience*, Vol.20, No.16, pp.1893–1916, 2008.

[15] P.S.L. Souza, S.R.S. Souza, E. Zaluska, "Structural testing for message-passing concurrent programs an extended test model", *Concurrency and Computation: Practice and Experience*, Vol.25, No.18, pp.149–158, 2013.

[16] S. Xanthakis, C. Ellis, C. Skourlas, "Application of genetic algorithms to software testing", *Proc. of 5th International Conference on Software Engineering and Applications*, Toulouse, France, pp.625–636, 1992.

[17] M.A. Ahmed, I. Hermadi, "GA-based multiple paths test data generator", *Computer & Operations Research*, Vol.35, No.10, pp.3107–3127, 2008.

[18] P. Bueno, M. Jino, "Automatic test data generation for program path using genetic algorithms", *International Journal of Software Engineering and Knowledge Engineering*, Vol.12, No.6, pp.691–709, 2002.

[19] A. Watkins, "The automatic generation of test data using genetic algorithms", *Proc. of 4th Software Quality Conference*, Austin, USA, pp.300–309, 1995.

[20] J.C. Lin, P.L. Yeh, "Using genetic algorithms for test case generation in path testing", *Proc. of 9th Asian Test Symposium*, Taipei, China, pp.241–246, 2000.

[21] X.Y. Xie , B.W. Xu, Shi L., C.H. Nie, "Genetic test case generation for path-oriented testing", *Journal of Software*, Vol.20, No.12, pp.3117–3136, 2009.

[22] P. McMinn, M. Harman, D. Binkley, P. Tonella, "The species per path approach to search-based test data generation", *Proc. of International Symposium on Software Testing and Analysis*, Portland, USA, pp.13–24, 2006.

[23] D.W. Gong, W. Q. Zhang, Y. Zhang, "Evolutionary generation of test data for multiple paths coverage", *Chinese Journal of Electronics*, Vol.19, No.2, pp.233–237, 2011.

[24] D.W. Gong, T. Tian, X.J. Yao, "Grouping target paths for evolutionary generation of test data in parallel", *Journal of Systems and Software*, Vol.85, No.13, pp.2531–2540, 2012.

[25] T. Tian, D.W. Gong, "Evolutionary generation of test data for path coverage of message-passing parallel programs", *China journal of computers*, Vol.36, No.11, pp.2212–2223, 2013.

TIAN Tian was born in Shandong Province, China, in 1987. She received the M.S. degree from Qufu Normal University, China, in 2010. Currently, she is a Ph.D. candidate in China University of Mining and Technology. Her research interests include genetic algorithms and parallel software testing. (E-mail: tian_tiantian@126.com)

Gong Dunwei was born in Jiangsu Province, China, in 1970. He received the Ph.D. degree in Control theory and control Engineering from China University of Mining and Technology in 1999. He is a professor in the School of Information and electronic engineering, China University of Mining and Technology. His research interest is search-based software engineering. (E-mail: dwgong@vip.163.com)