Directory Controller Verification Based on Genetic Algorithm

Li Luo(✉), Li Zhou, Hailiang Zhou, Quanyou Feng, and Guoteng Pan

College of Computer, National University of Defense Technology, Changsha 410073, China
li_luo@nudt.edu.com

Abstract. Directory protocol is the most widely used implementation cache consistency method in large-scale shared memory multi-core processor which is very complex and difficult to verify. In this paper, we propose a random test generation method based on genetic algorithm to verify directory controller of a type of 64-core processor, analyze the test features to code the symbols of genetic algorithm, and evaluate the merits of the test using the fitness function based on functional coverage. We establish the relationship between coverage and test vector, analyze the relationship between coverage and test stimulus through a genetic algorithm. The experimental results show that compared with the pseudo-random method, the functional coverage rate of this method is increased by nearly 20%–30%, the detection rate of bugs is relatively high, and the verification efficiency and quality are also improved.

Keywords: Directory-based protocol · Directory controller · Functional coverage · Genetic algorithm (GA) · Coverage directed test generation (CDG)

1 Introduction

Cache is a key component of microprocessor. With the development of processor structure to multi-core and many-core, cache consistency protocol is becoming more and more complex. How to ensure the correctness of consistency protocol has always been the focus of industry and academia. With the increase of the number of processor cores, the cache consistency protocol is the most commonly implementation method. Compared with the snoopy protocol, it has good scalability and reliability, and sends consistency requests accurately adopting peer-to-peer mode [1, 2]. Because of the state space explosion of directory consistency protocol, software simulation verification method has been a common verification means. The biggest challenge of software simulation method is how to generate high-quality test stimuli to cover function points. At the same time, the quality and progress of verification are usually measured by coverage. The goal of this paper is to verify the function of a 64-core processor directory controller. There are 1764 protocol message attribute function points, 58 key status registers in the directory controller, and 1822 functional points in total. The key problem of directory controller verification is how to generate effective test stimuli to cover functional points.

D. Dong et al. (Eds.): ACA 2020, CCIS 1256, pp. 209–220, 2020.
https://doi.org/10.1007/978-981-15-8135-9_15

The manual test stimuli take exhausting time, and the random test stimuli cover the function points with great blindness. We propose an approach for automatic coverage directed test generation (CDG) based on genetic algorithm, aim at constructing efficient test generators for checking the important behavior and specification of the design under test (DUT), improving the coverage progress rate; and designing stimulus that can reach uncovered tasks (coverage points). We establish the relationship between coverage and test vector, analyze the relationship between coverage and test through a genetic algorithm. The experimental results illustrate the effectiveness of the proposed algorithm in achieving the goals of CDG. Compared with the pseudo-random method, the functional coverage rate of this method is increased by nearly 20%–30%. The discovery rate of bugs is also improved effectively.

The rest of this paper is as follows. Section 2 reviews related work. Section 3 describes target DUT. Section 4 presents the details of the proposed genetic algorithm. Section 5 illustrates the experimental results. Finally, Sect. 6 concludes the paper.

2 Related Work

The software simulation method has always been a common verification method in the chip research. The test stimulus of software simulation mainly includes manual test, random test and coverage directed test generation (CDG). The manual test stimulus takes exhausting time and cost. Sometimes it is difficult to meet the needs of a large number of test vectors covering a wide range in regression.

Nagamani, A.N. [3] presented the first implementation of a generation framework that used feedback from coverage analysis to direct microarchitecture simulation, during the verification of IBM z-series servers, a CDG system had the potential to bring considerable advantages for a reasonable price. For models of a larger scale, the contribution was even greater. These advantages can save machine and personal time, and thus save money overall.

Fine S et al. put forward a random test generation method based on Bayesian network. This method was successfully applied to verify the PowerPC Northstar pipeline. Through improving the parameters of the training network, the verification efficiency and the hit rate of the simulation vector were effectively improved [5, 6].

Ilya Wagner etc. [7] used a random instruction generator driven by a Markov chain model, which has higher error efficiency compared with the random instruction test generation technology.

Yi Jiang-fang [8] used Bayesian network to describe the relation between the inputs and the branch statements. The new simulation vectors were generated by reasoning on the network. Experiments results indicated that the average vector length generated by the Bayesian network using different reference algorithms is about 10% of the original one, but the best path coverage even exceeds the original one.

Ai Yang-yang [9] discuss the Cache coherency protocol, analyze the coverage directed test generation (CDG) method based on Bayesian network reasoning and applied the method to Cache consistency verification. Taking the verification of the Cache coherence protocol of the FT processor as an example, the results show that the CDG method can increase coverage by nearly 30% in comparison with the pseudo-random test.

Bose M. [10] presented a genetic algorithm based framework to automatically generate biases. They targeted utilization of specific buffers for a new version of the PowerPC architecture. The results showed that the GA is effective in achieving high buffer utilization. Also, in targeting multiple objectives, the best approach to be used depends on whether the objectives were related.

Wang Shu-peng [11] proposed a coverage-directed test generation based on genetic algorithm (GA), which was used to verify two high-performance 32-bit multi-core processors. Results show that the proposed method can significantly reduce simulation time and improve verification efficiency.

Shen Hai-hua [12] presented genetic algorithm (GA) based coverage directed test generation (CDG), and built a coverage directed test generation platform. Experimental results showed that CDG can apparently accelerate the verification process and improve the reached coverage from 83.3% to 91.7%, which implied that verification efficiency was greatly improved and skilled manpower was cut down dramatically.

Nagamani, A.N [13] proposed a genetic algorithm based on heuristic test set generation method for fault detection in Reversible Circuits, which avoided the need for an exhaustive search. The approach validated on benchmark circuits considering missing-gate fault (complete and partial), bridging fault and stuck-at fault with optimum coverage and reduced computational efforts.

Genetic algorithms are intelligent approach to automate the generation of effective solution for black-box optimization without requirements of experience knowledge and resources. Only according to the input and output of DUT, Genetic algorithms learn the relationship between the feedback test vector and the coverage point automatically, which improves the automation of verification.

Directory-based protocols are very complex, which need rich test vector test to meet the requirement of functional verification, and random test produces a lot of redundant stimulus. In order to break through the bottleneck of verification, the genetic algorithm does not traverse the whole search space, we use genetic algorithm to mine the relationship between the coverage and the stimulus, and then guide the generation of random test stimulus, improve the growth rate of coverage, reduce the simulation time of redundant stimulus, and increase the efficiency of verification.

3 Background

Our research background is a 64-core processor developed, which is global shared memory and CMP structure, as shown in the Fig. 1, including processor core, cache, network on chip (NOC), directory control unit (DCU) and memory control unit (MCU).

Core: CPU core, which completes the scheduling and execution of instructions.

Cache: Two cores share one cache.

NOC: Interconnection network on chip, which provides information message exchange between caches and between cache and external memory of the CPU.

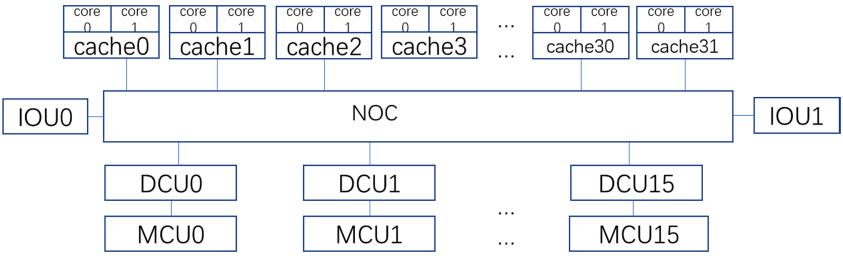

Fig. 1. Target system of 64-core CMP structure

DCU: directory controller, which records the usage of data block copies in each cache, and completes the maintenance of data consistency between each cache.

MCU: memory controller, attaching External DRAM memory, to achieve read-write access control of memory.

IOU: IO controller, connecting the PCIe device controller and other IO devices.

Two cores share a cache, and each cache line data has four states of MESIO (Modified, Exclusive, Shared, Invalid, Owned). The function of the directory controller is to support global data sharing, track and record the use of each cache line data, generate access requests to the memory controller MCU, process cache consistency protocol messages, send snoopy requests, accept cache snoopy responses, and complete cache requests.

The directory controller records the tags of all cache line dates and the using status of data copies in the cache, it tracks and modifies them according to the current received message commands and directory status to maintain the cache consistency among the caches in the whole processor.

The directory controller adopts the distributed directory implementation method. The directory protocol implementation method is divided into centralized and distributed [14], the centralized directory design is simple, the network traffic around the target controller is often the access hotspot, thus affecting the consistent transmission delay, resulting in the network power consumption hotspot. In order to improve the directory parallel processing ability of multi-core processors, the cache consistent transactions of the target system are evenly distributed to the DCU, and a DCU has the same address code for the same memory. In order to further improve the parallel processing performance, the DCU is divided into two individual banks, which are cross accessed by 6th address bit, 16 DCUs are designed on the 64-core chip. The directory table adopts the configurable group association mode, and the test configuration is 24-way 64 entries group association organization. The directory table is shown in the Fig. 2.

The organization and addressing mode of directory entry are the same as that of the cache tag. Before filling the cache line data, DCU allocates a directory entry to record using status of cache line data. If the cache line is replaced or does not retain the data copy, the directory will retrieve the corresponding directory entry (Fig. 3).

DCU entry mainly includes tag, busy, valid, vector, ECC bits and other information. The tag of the DCU entry, that is, the high 24-bit of the memory address. The busy defines the busy status of the directory entry. If the snoopy request is not completed, the new request with the same address cannot be processed. Valid means that the directory

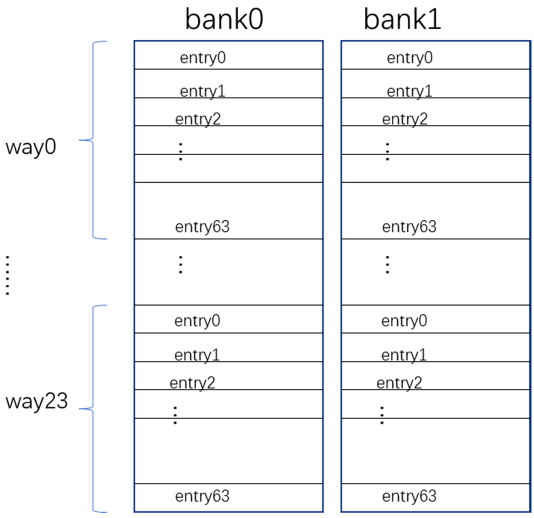

Fig. 2. Directory table structure of DCU

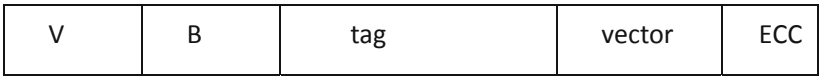

Fig. 3. Content of DCU entry content

entry is valid, at least one cache line has data copy. Vector defines which cache has data copy. ECC check bit is the Hamming check bit of the catalog entry.

The 16 DCUs of the target system are relatively independent, and DCU0 manages the cache consistency record mapping the memory access space of mcu0. Therefore, the functional verification of the directory controller can be tested against an independent DCU0.

A simulation environment is built for dcu0, as shown in Fig. 4. Cache is an IP design, it is replaced by cache model in DCU simulation environment, which is a functional model with an accurate clock, realizes the cache function in the processor. IOU and MCU select the function model with an accurate clock, simulates IO transaction and memory access transaction.

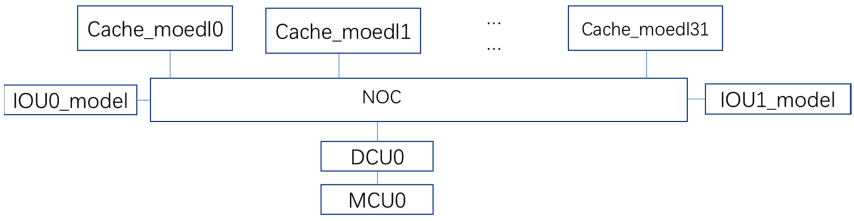

Fig. 4. Simulation environment of DCU0

DCU0 can receive requests from 32 caches or 2 IO controllers, MCU response, cache snoopy response, output cache snoopy request, or memory access or IOU access request. These input and output cache consistency information of dcu0 is transmitted in the format of on-chip network message, which can be divided into four categories: read or write request of cache/IOU, snoopy request, read or write response/snoopy response.

The main verification of DCU0 is the correct processing of cache consistency message and the directory table function, including directory entry hit, replacement, directory entry busy hit, directory table full hit, directory entry misses and other function points. There are 1764 protocol message attribute function points, 58 important status registers, and 1822 functions in total. Random test is used in the simulation process, redundancy test vectors often appear, which is easy to occur the verification period, the coverage of function points increases slowly. The key of DCU verification is how to generate effective test stimulus automatically, in order to break through the bottleneck of coverage and speed up the function verification.

We use a genetic algorithm to mine the relationship between the coverage and the stimulus, and then guide the generation of random test stimulus, improve the growth rate of coverage, reduce the simulation time of redundant stimulus, and increase the efficiency of verification. The effective stimulus generation is abstractly transformed into a genetic algorithm with improving coverage evolution.

4 Test Generation Based on Genetic Algorithm

The main process of genetic algorithm can be described as follows: the possible solution code of the problem is expressed as chromosome, and a chromosome population is randomly generated. Then, the chromosome individuals in the population are placed in a certain environment, and according to the survival principle of the fittest, the individuals with better adaptation environment are selected for replication, crossover, mutation and other operations, the next generation of individuals who are more adaptable to the environment. Such a generation evolves and keeps the offspring with large adaptive function. When the fitness function reaches the threshold, the evolution stops, and the optimal solution is obtained.

The essential feature of genetic algorithm is to code chromosomes through feasible solutions of the problem, to maintain optimization of crossover and mutation operators between generations, to define fitness function fitness in genetic algorithm, to judge the quality of chromosomes in the population, and to achieve multi-directional and global search to find the optimal solution of the problem.

4.1 Question Encoding

When genetic algorithm is used to solve the optimization problem, it is necessary to map the feasible solution of the problem from the solution space to the search space that the genetic algorithm can deal with, that is, to code the feasible solution of the problem with chromosome code.

The test generator is based on a genetic algorithm, its genetic code is related to the test stimulus. The genetic code is transformed into a feature vector, i.e. gene, by extracting the features of test stimulus. The gene sequence is built to obtain a chromosome.

Test stimulus features mainly come from the following aspects: first, opcode, non-cacheable read, shared read, noncacheable write, and cache maintenance commands of the message, etc.; second, the number of send SENDID, which indicates which cache and IO controller the data comes from, the third is address correlation of each message contained, if it is relevant, then the read and write addresses are the same. The fourth is bank number, the fifth is length encoding, whether the message request is 1 byte, 2 bytes, 16 bytes or 64 bytes data. So, gene expression includes five feature vectors.

In the implementation of genetic algorithm, genetic coding adopts symbol coding, and each bit field represents a feature vector. Using symbol coding corresponds to the problem itself, which is simple, easy to understand, and faster and more stable than binary coding in solving optimization problems. For example, gene expression is shown in Fig. 5. For gene coding (4,5,1,1,0), it can determine that cache4 sends a request that the opcode is op5. In this chromosome, the address is the same as the first gene address, access the directory table bank1 and read 1 byte of data.

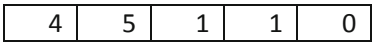

Fig. 5. An example of gene

A test stimulus consists of multiple test vectors, i.e. forming a chromosome. We define 32 genes to form a chromosome. As shown in Fig. 6, the chromosome is composed of 32 genes, Gene 0 is cache4, which sends out a request that the opcode is op5. The address is at the first address of this chromosome address, that is, random address. Access directory bank1, and read 1 byte of data. Gene 1 is cache0 which sent out a request with opcode OP1, the address is the same as the first gene address of this chromosome, access the directory table bank1, and read 64 bytes of data; gene 31 is the request of IOU 2 that the opcode is op0, the address is different from the first gene address of this chromosome, and read 16 bytes of data.

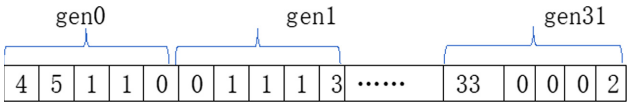

Fig. 6. An example of chromosome

4.2 Fitness Function

Genetic algorithms use a fitness value to evaluate the quality of chromosome. The evaluation of solutions represented by fitness values is important to guide the learning and evolution process in terms of speed and efficiency. The function verification based on simulation is to get the maximum function coverage in a short period of time. Different scenarios and functions defined by test stimulus are reflected by the population of chromosomes. Fitness function in genetic algorithms is used to judge the quality of

chromosomes in the population. Therefore, in the application of this study, the fitness function is to reach the most function points. In genetic algorithms, a chromosome C represents a test stimulus, and there are N function points to be tested and covered in the DCU0. For a chromosome C, its fitness function fitness. Its fitness function is the number of function points of coverage. The number of function points included in chromosome C is M, the gene i of chromosome C covering the jth function point is defined as cov[i, j], and Covering the jth function point of chromosome C is defined as follows:

$$\begin{cases} \text{cov}[j] = |(\text{cov}[i, j]), \text{ for, } i = 1, 2, \ldots .M \\ \text{s.t. } \text{cov}[i, j] = 1, \text{ Gene i covers the jth function;} \\ \text{else } \text{cov}[i, j] = 0, \text{ Gene i does not cover the jth function} \end{cases} \tag{1}$$

$$\text{Fitness}_c = \max(\sum_{j=1}^{N} (w_j * cov[j]))/N \tag{2}$$

w_j represents the weight and importance of the jth function point. Its value can be adjusted according to the design features. For example, when only the module of the directory table bank0 is tested, the function point weight of bank1 can be set to 0.

4.3 Mutation Operator and Crossover Operator

The mutation of genetic algorithm itself is a kind of local random search, which is combined with random and crossover operators to ensure the effectiveness of genetic algorithm, make genetic algorithm have the ability of local random search, and keep the diversity of population, so as to prevent premature convergence. In order to avoid invalid operation, we adopt fixed-point mutation. The position of mutation operation is the first and second position of gene, as shown in Fig. 7, an example of mutation operation.

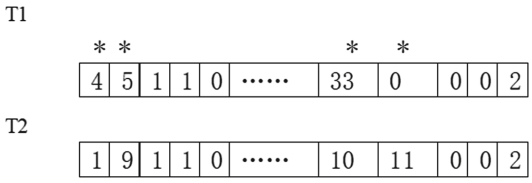

Fig. 7. Example of mutation operation

An example of mutation operation shown in Fig. 7 is to perform the mutation operation on chromosome T1. If there is a mark "*" in the characteristic position, the mutation will occur. T1 generates a new chromosome T2 through mutation. We select the mutation feature as opcode and SENDID. We need to pay attention to the legitimacy of the variation, such as the legality of opcode. IO devices can only send non-cacheable read and non-cacheable write requests, such as sending cacheable requests, this must be an illegal operation and needs to be deleted.

The crossover of genetic algorithm is the operation of replacing and recombining part of the structure from the parent to generate new individuals. Its purpose is to generate new individuals in the next generation, just like the process of human evolution, so

that the search ability of genetic algorithm can be improved greatly. Crossover is carried out according to probability. The higher the frequency of crossover, the faster the optimal solution can converge, but too high will lead to premature convergence. Common crossover includes single point crossover, multi-point crossover, uniform crossover, etc. we select single point crossover, and the crossover operator will randomly exchange some feature bits of two chromosomes according to the crossover rate, so as to generate a new feature combination. The purpose of crossover is to combine the useful features together to produce more effective and active coverage of function points. The specific operation is to set a crossing point in the chromosome code, then exchange the partial structure of the two chromosome codes before and after the crossing point, and form two new chromosome codes, that is, two new test stimuluses. Figure 8 introduces an example of a single point crossing, through which two chromosomes T1 and T2 in Fig. 9 can be changed into two new chromosomes T′1 and T′2.

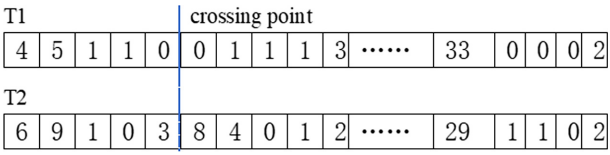

Fig. 8. Chromosomes before cross operation

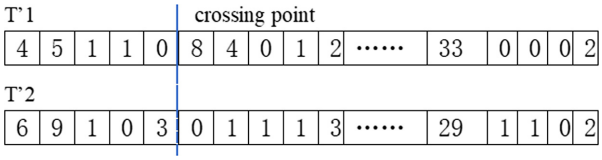

Fig. 9. New chromosomes after cross operation

4.4 Parameters of Genetic Algorithm

The genetic algorithm needs to determine the size of the test set, that is, the number of genetic populations POPSIZE. When the POPSIZE value is small, the calculation time of the algorithm is short, but the probability of the algorithm converging to the optimal solution is low, that is, the global search ability is small, and the local optimal solution may be obtained instead of the global optimal solution. With the increase of POPSIZE, the probability of convergence to the optimal solution will increase, but the calculation time of the algorithm will also increase significantly. Our algorithm defines the population size as 32 chromosomes.

Genetic algorithm crossover is built on probability. The higher the crossover frequency is, the faster the optimal solution can converge, but too high will lead to premature convergence. Mutation is kind of local random search, at the same time, it makes the genetic algorithm keep the diversity of the population, so as to prevent the premature

convergence. In the mutation operation, the mutation rate cannot be too large. Otherwise it may degenerate into random search. At this time, some important mathematical characteristics and search ability of the genetic algorithm no longer exist. In a compromise, the crossover probability PC = 0.8 and the mutation probability PM = 0.1 are defined here.

5 Experimental Results

The parameters of the genetic algorithm are set as follows: population size POPSIZE = 32, Maximum number of evolutionary iterations MAXGENERATION = 10, crossover probability PC = 0.8, mutation probability PM = 0.1. According to fitness function of Formula-1 and Formula-2, chromosome evaluation in genetic algorithm is realized, The simulation scenario of testbench1 is 32 caches and two IOU devices accessing a directory controller DCU0, sending out 4 K requests, simulating 90 K cycles, obtaining 100% coverage, pseudo-random test stimulus reaches 73%, as shown in Fig. 10, testbench2 simulation scenario 2 is 4 caches and two IOU devices accessing DCU0, sending out 256 requests, simulating 1 K cycles. At the same time, the pseudo-random test motivation reaches 81% of the functional coverage, as shown in Fig. 11.

In the regression test of directory simulation, there are 24 bugs found by testbench1 of genetic algorithm, 5 bugs of high quality, and 21 bugs found by testbench2 of genetic algorithm. The results and performance comparison of the algorithm are shown in Table 1.

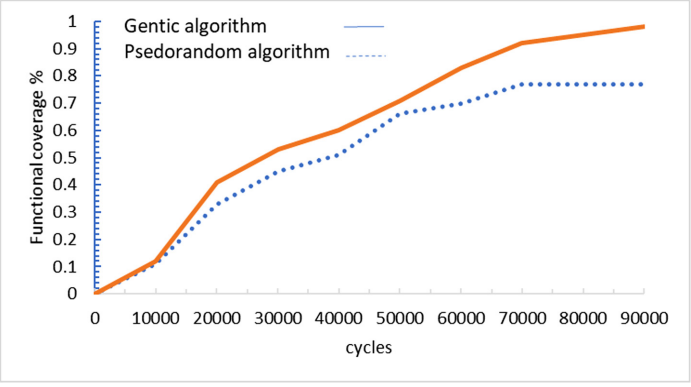

Fig. 10. A coverage comparison of testbench1 simulation scenario

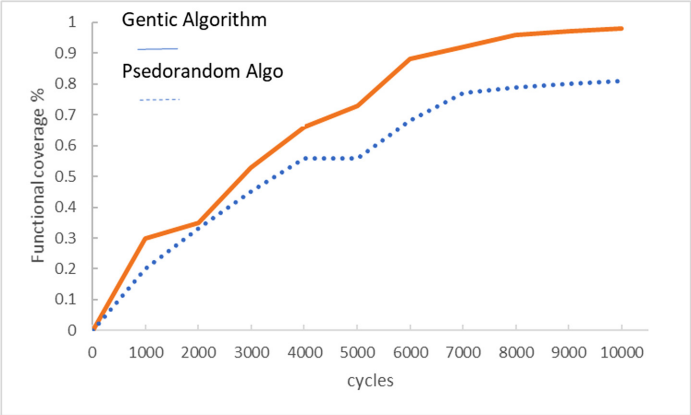

Fig. 11. B coverage comparison of testbench2 simulation scenario

Table 1. Performance comparison of algorithms

Test scenario	Algorithm	Simulation time (cycles)	Test vector	Functional coverage	Detecting bugs in regression test
TestBench1	Genetic algorithm	90k	4k	100%	24
TestBench1	Pseudorandom algorithm	90k	4k	73%	19
TestBench2	Genetic algorithm	10k	1k	100%	21
TestBench2	Pseudorandom algorithm	10k	1k	81%	15

6 Conclusions

In this paper, we propose a random test generation method based on genetic algorithm to verify directory controller of a type of 64-core processor, which uses a fitness function based on function coverage to evaluate the quality and value of verification. The genetic algorithm is used to establish the relationship between the coverage analysis results and the effective stimulus to direct the generation of higher quality tests. The experimental results demonstrate that compared with the pseudorandom test generator, the proposed test generator can achieve higher function coverage in a short time, reduce the verification time and improve the verification efficiency. The parameters of the genetic algorithm in this paper are fixed, which is likely to cause the genetic algorithm to fall into the local optimal solution. In the future, we will do further research on the adaption of the parameters and test weight of the genetic algorithm, further expand the local optimal solution space of the genetic algorithm, and generate higher quality tests.

Acknowledgments. This work is supported by National Key Research and Development Program of China No. 2018YFB0204301.

References

1. Hill, M.D., Sorin, D.J., Wood, D.A.: A Primer on Memory Consistency and Cache Coherence. Synthesis Lectures on Computer Architecture, November 2011
2. Simoni, R., Horowitz, M.: Modeling the performance of limited pointers directories for cache coherence. In: Proceedings of the 18th International Symposium on Computer Architecture, pp. 309–318 (1991)
3. Nativ, G., Mittennaier, S., Ur, S., Ziv, A.: Cost evaluation of coverage directed test generation for the IBM mainframe. In: Proceeding of the 2001 International Test Conference, Baltmore, pp. 793–802 (2001)
4. Fine, S., Ziv, A.: Coverage directed test generation for functional verification using Bayesian networks. In: Design Automation Conference, pp. 286–291 (2003)
5. Braun, M., Fine, S., Ziv, A.: Enhancing the efficient of Bayesian network based coverage directed test generation. In: Proceedings of IEEE International High-Level Design and Test Workshop, Sonoma, pp. 75–80 (2004)
6. Fine, S., Freund, A., Jaeger, I., Naveh, Y., Mansour, Y.: Harnessing machine learning to improve the success rate of stimuli generation. IEEE Trans. Comput. **55**(11), 1344–1355 (2006)
7. Wagner, I., Bertacco, V., Austin, T.: Microprocessor verification via feedback-adjusted Markov models. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **26**(6), 1126–1138 (2007)
8. Yi, J., Dong, T., Xu, C.: An efficient approach to simulation vector generation using Bayesian network. J. Comput.-Aided Des. Comput. Graph. **19**(5), 616–621 (2007). (in Chinese)
9. Ai, Y., Luo, L., et al.: A Bayesian network based test generation method for cache coherency protocol verification. Comput. Eng. Sci. **39**(8), 1397–1402 (2017). (in Chinese)
10. Bose, M., Shin, J., Rudnick, E.M., et al.: A genetic approach to automatic bias generation for based random instruction generation. In: Proceedings of Congress on Evolutionary Computation, Seoul, pp. 442–448 (2001)
11. Wang, S., Huang, K., Yan, X.: Coverage directed test generation based on genetic algorithm. J. Zhejiang Univ. (Eng. Sci.) **50**(3), 581–588 (2016)
12. Shen, H., Wang, P., et al.: A coverage directed test generation platform for microprocessors using genetic approach. J. Comput. Res. Dev. **46**(10), 1612–1625 (2009)
13. Nagamani, A.N., et al.: A genetic algorithm-based heuristic method for test set generation in reversible circuits. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **37**(2), 324–335 (2018)
14. Ros, A., Acacio, M.E., García, J.M.: A scalable organization for distributed directories. J. Syst. Archit. **56**(2–3), 77–87 (2010)