Zhang Y, Ji PF, Zhu PW *et al.* Parallel software-based self-testing with bounded model checking for kilo-core networks-on-chip. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 38(2): 405-421 Mar. 2023. DOI: 10.1007/s11390-022-2553-3

Parallel Software-Based Self-Testing with Bounded Model Checking for Kilo-Core Networks-on-Chip

Ying Zhang[1] (张　颖), *Member, CCF, IEEE*, Peng-Fei Ji[1] (季鹏飞), Pan-Wei Zhu[1] (朱潘玮)
Zebo Peng[2], *Senior Member, IEEE*, Hua-Wei Li[3] (李华伟), *Fellow, CCF*, and
Jian-Hui Jiang[1, *] (江建慧), *Senior Member, CCF*

[1] *School of Software Engineering, Tongji University, Shanghai 200092, China*

[2] *Department of Computer and Information Science, Linkoping University, Linkoping 58183, Sweden*

[3] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*

E-mail: yingzhang@tongji.edu.cn; 1831603@tongji.edu.cn; 1950061@tongji.edu.cn; zebo.peng@liu.se
lihuawei@ict.ac.cn; jhjiang@tongji.edu.cn

Received June 1, 2022; accepted December 15, 2022.

Abstract Online testing is critical to ensuring reliable operations of the next generation of supercomputers based on a kilo-core network-on-chip (NoC) interconnection fabric. We present a parallel software-based self-testing (SBST) solution that makes use of the bounded model checking (BMC) technique to generate test sequences and parallel packets. In this method, the parallel SBST with BMC derives the leading sequence for each router's internal function and detects all functionally-testable faults related to the function. A Monte-Carlo simulation algorithm is then used to search for the approximately optimum configuration of the parallel packets, which guarantees the test quality and minimizes the test cost. Finally, a multi-threading technology is used to ensure that the Monte-Carlo simulation can reach the approximately optimum configuration in a large random space and reduce the generating time of the parallel test. Experimental results show that the proposed method achieves a high fault coverage with a reduced test overhead. Moreover, by performing online testing in the functional mode with SBST, it effectively avoids the over-testing problem caused by functionally untestable turns in kilo-core NoCs.

Keywords software-based self-testing (SBST), parallel test, kilo-core networks-on-chip (NoCs), online testing

1 Introduction

Large-scale networks-on-chip[1] (NoCs) have emerged as a promising architecture for supercomputers due to their outstanding parallel communication capability[2]. This architecture integrates many processor cores into a single chip and turns the chip into a small supercomputer. This architecture effectively alleviates the bottleneck faced by the next generation of supercomputers, as most of the communications among chips are moved inside a chip to reduce transmission delays[1]. For example, the Taihu-light supercomputer uses such a large-scale NoC (i.e., the SW26010 processor), which integrates 260 processor cores[1]. According to a report from Oak Ridge National Laboratory[3], the excellent performance of the architecture helped the Taihu-light Supercomputer hold its status as the fastest supercomputer in the world for three years (2016–2018). Currently, many research institutes are trying to design kilo-core NoCs for the next generation of supercomputers.

A kilo-core NoC requires to be robust and fault-tolerant[4, 5] as it is not only used in low-fault situations[1]. Since a chip's fault probability is proportion-

Regular Paper

This paper was supported in part by the National Key Research and Development Program of China under Grant No. 2020YFB 1600201, the National Natural Science Foundation of China (NSFC) under Grant Nos. 61974105, 62090024, and U20A20202, and the Zhejiang Lab under Grant No. 2021KC0AB01.

*Corresponding Author

al to its size, and the size of a kilo-core NoC reaches the scale of the entire wafer[6], ensuring the NoC is fault-free during the manufacturing stage becomes very difficult. If a foundry simply discards a kilo-core NoC that is faulty somewhere in the chip, the chip yield will be very low, and its cost will be impractically high[1]. A faulty NoC should, therefore, be treated as a good chip after its faults have been dealt with. During the operating stage, a kilo-core NoC continuously works in a thermal-intensive environment[7–9], and severe thermal-intensive conditions will exacerbate the aging problem[10] and increase its faulty probability. Long downtimes in replacing the faulty chip would result in unacceptable operating costs for supercomputers. Therefore, a kilo-core NoC must support dynamic reconfiguration to handle its faults online. A feasible solution to this problem is to implement more processor cores on a single chip, discard the faulty cores and links, and treat the faulty chip as a good one as long as the number of its available cores meets the design requirement[1]. To implement this solution, we first need an online test method to discover faults in an NoC.

Although online testing is required to guarantee a kilo-core NoC's reliability, testing such a large NoC remains a great challenge. First, this system contains about 1 000 switchers and nearly 5 000 buffers. Its test packets have to cover all the buffers and all the feasible turns and conflicting scenarios on each switch. In addition, not all turn channels in an NoC can be activated by the routing algorithm in the function mode. If all NoCs with faults on the functionally untestable turns will be discarded, it will lead to yield loss and cause the over-testing problem[11]. Furthermore, the packets for online testing will occupy the high-speed L1 cache (i.e., critical resources) in a kilo-core NoC. In order to avoid the interference of functional packets on the test process, the test packets have to be transmitted in the NoC's idle periods. Hence, a parallel test technique has to minimize its required storage space and executing time. Manually configuring parallel packets to achieve the above two goals will be an arduous task. Therefore, a novel technology is required to automatically configure test packets.

In this paper, we develop a parallel software-based self-testing (SBST) method to form a sequence of optimized packet configurations to test a kilo-core NoC in parallel with the minimal test overhead. We applied SBST with bounded model checking (BMC)[12, 13] in our previous work[2] aiming at generating test packets for a single buffer or switch. In this work, we generate the optimal configuration of such test packets for the kilo-core NoCs using a Monte-Carlo simulation algorithm. Furthermore, we employ a multi-threading technology to evaluate the performance of the configurations of these packets in parallel. The key contributions are as follows.

1) The developed SBST with BMC[12, 13] can derive the leading sequence for each internal function and detect all functionally testable faults related to the different functions of the NoC.

2) A Monte-Carlo simulation algorithm is developed to search for the approximately optimum configuration of the test packets, which not only guarantees the test quality but also minimizes the test overhead.

3) The multi-threading technology is used to facilitate the Monte-Carlo simulation to search for the approximately optimum configuration in a large random space and reduce the generating time of the parallel tests.

The rest of the paper is organized as follows. Section 2 describes the background of implementing SBST on an NoC. In Section 3, we describe the SBST with BMC for testing the NoC buffer or switch presented in our previous work[2]. The parallel SBST technique is then presented in detail in Section 4. Section 5 describes and explains the experimental results. Finally, we conclude the paper in Section 6.

2 Background

Many NoC testing techniques have been published in the literature. In early work, Cota *et al.* used an NoC as a test access mechanism for manufacturing tests[14, 15], while Richter and Chakrabarty[16] further optimized the number of test pins and minimized the test application time. Although these methods can be used for manufacturing tests, they require external automatic test equipment (ATE) and are therefore not suitable for online tests. The built-in self-test (BIST) was also used to test an NoC[17, 18]. Although BIST reduces the need for external ATE, it cannot be used for online testing when the system is in operation, because it requires the system to be switched to an extra test mode. Besides, its area overhead is reported to be as high as 21% of the original design area[17]. Researchers also proposed boundary scans for NoC testing, by inserting design-for-test hardware into an NoC router's ports[19, 20]. This method can effectively test the router's data path but only achieves limited coverage on its control logic. The effective functional test for the sequential circuit corresponding to the control logic remains an open problem.

Software-based self-testing (SBST) is a promising online testing method[21] as it executes native instructions in the functional mode and can potentially lead to high fault coverage[2]. The method is also capable of online testing an NoC by transmitting packets through its on-chip network[2]. Specifically, when the SBST program loads or stores data (i.e., packets) from/to the remote core, the NoC transmits these packets from the source core to the target core through the on-chip network, and the transmitted packets can be used to test the functional and structural faults in the on-chip network. Besides, the SBST program can also generate "stalls" between flits by inserting NOP instructions (or interrupts); these stalls can activate the router's internal functions.

Collet *et al.* proposed an SBST method for NoC-based multi-core arrays[22], where the SBST program tests the processors but requires auxiliary circuits to test the on-chip network. Although this method requires low area overhead, it has to switch the system from the functional mode to the test mode; therefore, it is not suitable for the online testing scenarios. Dalirsani *et al.* developed a structural SBST method for NoCs[23]. The method applies a timeframe extension to the router and generates test patterns using the SAT solver. However, automatically extracting input sequences is still an open issue. The method given in [23] manually generates test sequences within a small sequence depth but has its limitations for circuits with a large sequence depth. Therefore, a novel SBST method is required to automatically generate test packets and test the hard-to-access faults under a large sequence depth.

3 SBST with BMC for Sequential Circuits

In this work, we develop an SBST method based on BMC (i.e., BSBST)[2] to online test the sequential control logic and target hard-to-detect faults. The method uses an extended finite-state machine (EFSM)[24] to model the sequential control logic. The EFSM can be automatically extracted from an RTL (register-transfer level) description[24].

We define an EFSM as an eight-tuple (I, O, S, R, T, E, F, W), where $I(O)$ denotes the set of input (output) symbols, corresponding to, e.g., input (output) packets. S represents the state set, e.g., the states of the handshake FSM, and R denotes the variable set, e.g., data registers in buffers. T is the transition set, and W is the set of internal wires. A given transition t_a is controlled by an enable function e_a and an update function f_a. The sets of enable functions and update functions are denoted by E and F, respectively. Finally, we denote the property set to test the faults on the wires corresponding to E and the set corresponding to F as P_E and P_F, respectively.

Definition 1. *A functional test on a sequential circuit is a test that detects structural faults that are excited during every internal state transition.*

As some structural faults never lead to errors in a sequential circuit during normal operation[25], a functional test is used to target the testable faults in the functional mode. In this work, we consider the stuck-at fault model. We assume that a sequential circuit is effectively tested if every transition t_a is activated, and all wires in the subset of W corresponding to t_a are controllable and observable.

3.1 Implementation of BMC for Testing

In this subsection, we apply BMC to excite the target function, denoted by a given property p. Let us assume that the model takes b timeframes from the initial state to arrive at the state where the target function is activated. Let B refer to an upper bound on b. BMC efficiently searches all the paths from the initial state within B timeframes to check the property $!p$ (the complement of p). Once a counterexample that violates $!p$ emerges, it activates the target function p.

The BMC tool can also output the input signal sequence and the internal variable sequence associated with the counterexample. We combine these sequences to form the leading sequence. Let I^1, I^2, ..., I^b be the input signals, and R^1, R^2, ..., R^b be the internal variables at these b timeframes; then the leading sequence is defined as $((I^1, R^1), (I^2, R^2), ..., (I^b, R^b))$.

Definition 2. *A leading sequence that excites the property p, referred to as $LS(p)$, is the combination of the input signal sequence and the internal variable sequence from the initial state in the counterexample.*

A leading sequence is used to facilitate the test of structural faults in a sequential circuit. For example, if a stuck-at 0 fault on a wire w in the NoC is under test, we can set "w is equal to 1 and w is observable on output signals" as the target function p and load this function into the BMC tool. Once the BMC tool derives the leading sequence, the input signal sequence in $LS(p)$ will then be transformed into the test sequence.

3.2 Flowchart of SBST with BMC

We present a flowchart for the SBST algorithm with BMC (i.e., BSBST) in Fig.1[2] for sequential circuits. First, the algorithm automatically extracts the EFSM from the design[24] and stores the sets of properties P_E and P_F in a property database. Second, it checks if an unchecked property exists in the database. If so, the algorithm takes one property from the database and goes to the next step; otherwise, the algorithm terminates. Third, the algorithm makes use of a slicing technique to reduce the model size in order to alleviate the state-space explosion problem. Fourth, it uses a BMC tool to check the property. If the tool fails to derive the leading sequence, the property is not testable in the functional mode, and the algorithm returns to the second step; otherwise, the algorithm goes to the next step. Fifth, if the property is in P_E, the input signal sequence in the leading sequence is assumed as the test sequence; otherwise, the algorithm goes to the next step. Sixth, the leading sequence guides the design to the timeframe when the given update function happens, and then a constrained automatic test pattern generation (constrained ATPG) procedure will be used to generate test patterns for the faults that are excited by the update function. Finally, the algorithm translates the test patterns into test sequences and returns to the second step for the next property.

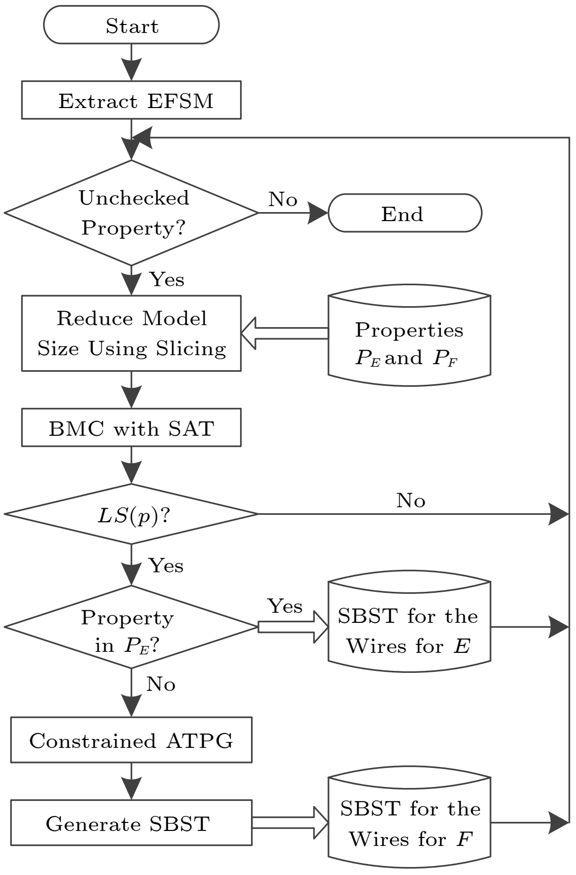

Fig.1. Flowchart of SBST with BMC[2].

In the BMC process, the state-space explosion problem can become serious if the BMC tool directly loads the complete design, which is usually very large. We use two methods to reduce the size of the model. First, we exploit the slicing method[26] to reduce the size of the model by removing non-contributive transitions and variables. The non-contributive transitions/variables are those that do not affect the variables in the target property. Second, we remove the data variables that simply store data information instead of activating any transitions, and thereby significantly reduce the size of the design.

3.3 Sets of Properties in BMC

In this subsection, we describe the sets of properties P_E and P_F used in BMC.

First, let I_{ea}, R_{ea}, and W_{ea} be the subsets of I, R, and W corresponding to a given enable function e_a, respectively. Let $v(R_{ea})$ refer to one possible assignment of the variables in R_{ea}. As R_{ea} contains only a few registers, a small group of $v(R_{ea})$ exists. We enumerate all possible assignments of the variables in R_{ea} to test faults on the wires in W_{ea}. The property requires four conditions: 1) the current state is $cs(t_a)$, 2) the enable function e_a is true, 3) the variables in R_{ea} are equal to the values of the given assignment $v(R_{ea})$, and 4) the elements of R_{ea} are observable. Finally, we collect the group corresponding to every enable function and build the property set P_E.

Second, let R_{fa} and W_{fa} be the subsets of R and W corresponding to a given update function f_a respectively. We develop one property to activate the update function f_a in BMC. This property requires three conditions: 1) the current state is $cs(t_a)$, 2) the enable function e_a is true, and 3) the elements of R_{fa} are observable. Finally, we collect the property corresponding to every update function and build the property set P_F.

3.4 Generation of Test Patterns Using Constrained ATPG

We utilize constrained ATPG[27] to generate test patterns for the faults on the wires in W_{fa}. The key idea is to fix the variables in R_{ea} and the input signals I_{ea} in the leading sequence and to detect the testable faults on the wires in W_{fa} by assigning the remaining variables and input signals.

Fig.2 presents the steps needed to implement constrained ATPG for the faults on the wires in W_{fa}. First, the method transforms the synthesized design into a combinational circuit by removing all internal registers and setting these registers' outputs as pseudo-primary inputs (PPIs) of the combinational circuit and their inputs as pseudo-primary outputs (PPOs). Second, the values of R_{ea} in R^b of the leading sequence are mapped to PPIs corresponding to R_{ea}, while the values of I_{ea} in I^b are mapped to inputs corresponding to I_{ea}. If the update function f_a does not update some variables, such as the register r_2, the method makes the PPO corresponding to r_2 non-observable. Third, ATPG is used to target faults on the wires in W_{fa} by assigning values to unspecified inputs and PPIs and thereby generating compact and effective test patterns. Finally, the method uses the mapping technique from [28] to translate test patterns into SBST programs.

3.5 Implementation of SBST with BMC on an NoC Router

In this subsection, we implement the SBST with the BMC method on the critical components, such as a buffer or a switch. The buffer is often used for handshaking with an adjacent node. Fig.3 presents the handshake state machine of a buffer in the well-

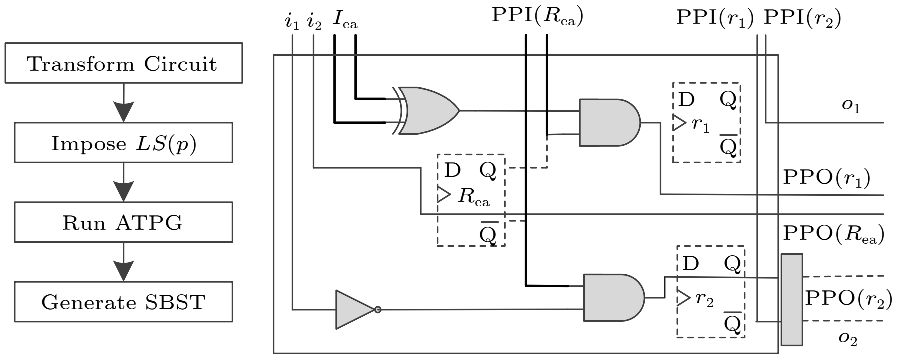

Fig.2. Constrained ATPG on a sequential circuit.

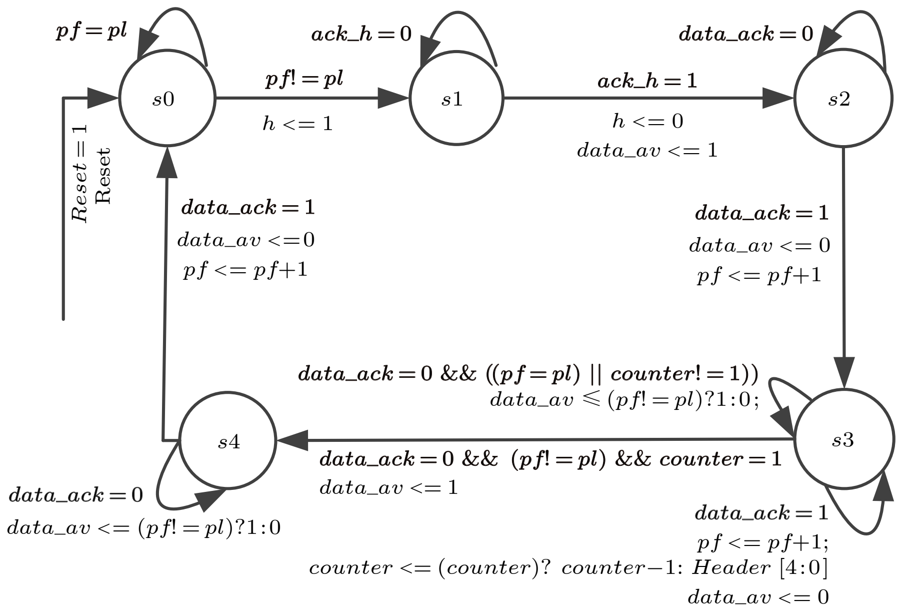

Fig.3. Handshake state machine in a buffer.

known Herms routers[1], where the bold text on the transition arc refers to an enable function, and the normal text corresponds to an update function. Note that the SBST with BMC is a general test generation technique that can be applied to NoCs with other routing algorithms. We first prepare the sets of properties P_E and P_F according to the EFSM corresponding to the buffer respectively. Each property p in P_E should activate a transition t_a and assign a given $v(R_{ea})$ to the variables corresponding to the enable function e_a. If the BMC tool derives a leading sequence $LS(p)$ for property p, the input sequence of $LS(p)$ is considered as the test sequence. Each property p in P_F activates the update function f_a. If the BMC tool derives a leading sequence $LP(p)$, this sequence can trigger the update function. Subsequently, the method runs a constrained ATPG algorithm on the transformed buffer for faults that are sensitized by the update function. Finally, the method translates these sequences into test packets for the buffer.

In our model-checking flow, the BMC tool NuSMV[2] is used by us. At first, it loads in a 16-bit buffer's model whose sequential depth is set to 31. Next, it loads the sets of properties P_E and P_F and derives the leading sequences for these properties. It loads in 25 properties in total and takes only 1.19 s of CPU time to derive the leading sequences for all the properties on the sliced EFSM.

The switch is responsible for forwarding the packets in the input buffer to the correct output port and arbitrating input requests if multiple requests exist. Fig.4 presents its arbitrating state machine. The method sets the condition of activating each transmission turn as a property for the following step. In the model checking, the slicing method[26] greatly reduces the size of the original design, which, in this case, is a 16-bit router with five buffers and one switch. In Table 1, the number of variables and wires is reduced to 3.8% and 7.4%, respectively, of the original design. The BMC tool takes only 1.29 s to verify 32 properties and generate all the leading sequences. Finally, the method translates these sequences into test packets for the switch.

4 Parallel SBST for Kilo-Core NoCs

4.1 Framework of Parallel SBST

In this work, we develop a parallel SBST (i.e., PSBST) based on the Monte-Carlo simulation technique[29] to automatically generate configurations of test packets that concurrently emerge on different NoC nodes and test the NoC in parallel with the minimal test overhead. Since the NoC mapping problem is an NP-complete problem, and the scale of a kilo-core NoC is huge, it is infeasible to find the optimum configuration of test packets. Instead, we apply the Monte-Carlo simulation to find an approximately optimum solution. Fig.5 presents the framework of the PSBST algorithm. First, the algorithm generates a set of random configurations of test packets at the current time T_s. Then, it simulates each configuration till its termination time T_e when all packets arrive at their destination cores. It also counts the packet number and the contribution to the fault coverage. Later, the algorithm comprehensively evaluates these configurations and obtains an optimized configuration. The algorithm then generates another set of configura-

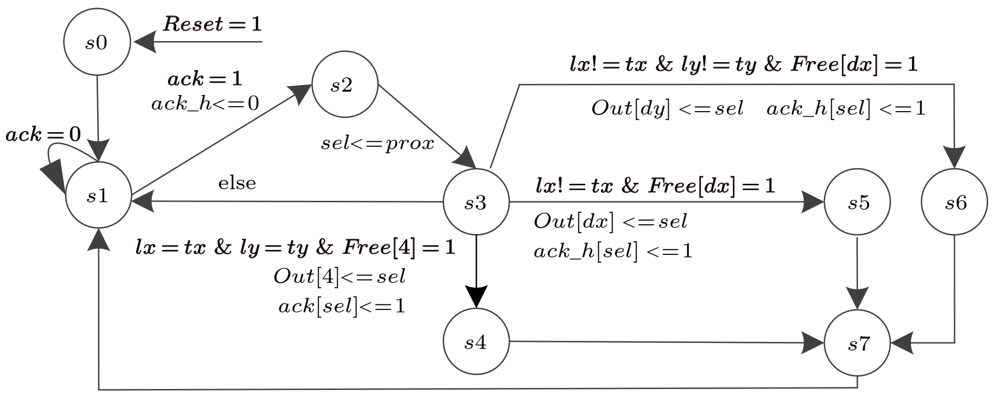

Fig.4. Arbitrating state machine in a switch.

[1]de Mello A V, Möller L H. Hermes NoC. https://www.inf.pucrs.br/~calazans/research/Projects/Hermes/Hermes.html, Mar. 2023.

[2]Cimatti A, Roveri M, Cavada R. NuSMV. http://nusmv.fbk.eu/, Mar. 2023.

Table 1. BMC Results for the Buffer and the Arbitrator

	Number of Variables	Number of Wires	Max Timeframes	Number of Properties	CPU Time (s)
Normal buffer	275	490	23	25	1.19
Sliced buffer	19	114	23	25	1.19
Normal arbitrator	1 429	2 961	9	32	1.29
Sliced arbitrator	54	218	9	32	1.29

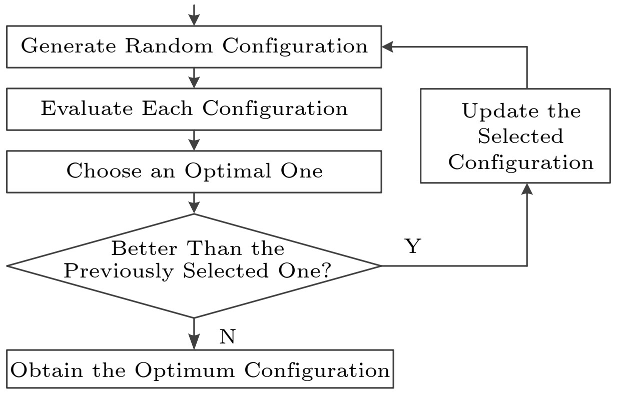
Generate Random Configuration
Evaluate Each Configuration
Choose an Optimal One
Better Than the Previously Selected One?
Y
N
Update the Selected Configuration
Obtain the Optimum Configuration

Fig.5. Framework of the parallel SBST.

tions, derives another optimized configuration, and checks whether the new configuration improves the previous one. If so, the algorithm updates the selected configuration, and the Monte-Carlo simulation continues. Otherwise, the configuration is finally selected. With this approach, the final configuration is derived from a huge amount of random configurations, and the optimum one is not guaranteed. Hence the final selected configuration is only an approximately optimum solution based on the principle of the Monte-Carlo simulation[29].

4.2 Network-on-Chip Modeling

Modeling a kilo-core NoC at the high level is required to speed up packet simulation, as simulating a kilo-core NoC at the gate level faces many problems. First, the gate-level netlist contains too many details, and a kilo-core NoC is huge in size, and thus the gate-level simulation is very time-consuming. Second, the simulation of massive configurations cannot be accelerated by parallel computing as the number of parallel simulators is strictly limited by commercial tools. Hence a high-level and cycle-accurate simulator is required for the parallel SBST solution.

We have developed a high-level simulator using C language for kilo-core NoCs. The simulator preserves the necessary variables of the NoC using the Herms routers[③], omits many low-level details, and thereby ensures its efficiency. Also, we can design high-level simulators for NoCs with other routing algorithms. The simulator also completes the functions of the buffers and the switches[③], and its results maintain cycle accuracy with that of the gate-level simulator. Fig.6 presents the simulator's workflow. First, the simulator puts the given packets onto the NoC's input ports. Then, it updates the state of each switch (i.e., $state(x_i, y_i)$) according to the arbitrating state machine in Fig.4. If the state is $s1$, the simulator checks if each input buffer has a packet request and sends these requests to the switch. If the state is $s2$, the simulator selects a packet request according to the round strategy[③]. If the state is $s3$, the simulator checks whether the next buffer in an adjacent router for the selected request is not full. If it is full, the state returns to $s1$. Otherwise, the simulator determines the transmission direction. If the state is $s4$, $s5$, or $s6$, the simulator sends the head flit in the selected packet to the next buffer and reduces the buffer's flit number by 1.

Then, the simulator checks the tail flit in each buffer according to the handshake state machine[③] in Fig.3. If the head flit from the same packet exists in the buffer, the tail flit has to wait. Otherwise, the simulator checks if the next buffer for the tail flit is full. If it is full, the tail flit also has to wait. Otherwise, the flit number in the packet state (i.e., *pnum*) is reduced by 1. Once the flit number in the packet state is reduced to 0, the tail flit enters its next buffer. This simulation preserves the critical functions of the NoC and eliminates many control signals. Therefore, this high-level simulator can quickly simulate the NoC and keep cycle accuracy with the gate-level simulator.

4.3 Test Objectives and Problem Modeling

Simulating faults directly on the netlist is not applicable to the Monte-Carlo simulation either. A kilo-core NoC contains tens of millions of faults, and fault simulation is extremely time-consuming (e.g., hundreds of hours per configuration). Moreover, our Monte-Carlo simulation involves thousands of configurations. Hence, it requires a novel method to evalu-

[③]de Mello A V, Möller L H. Hermes NoC. https://www.inf.pucrs.br/~calazans/research/Projects/Hermes/Hermes.html, Mar. 2023.

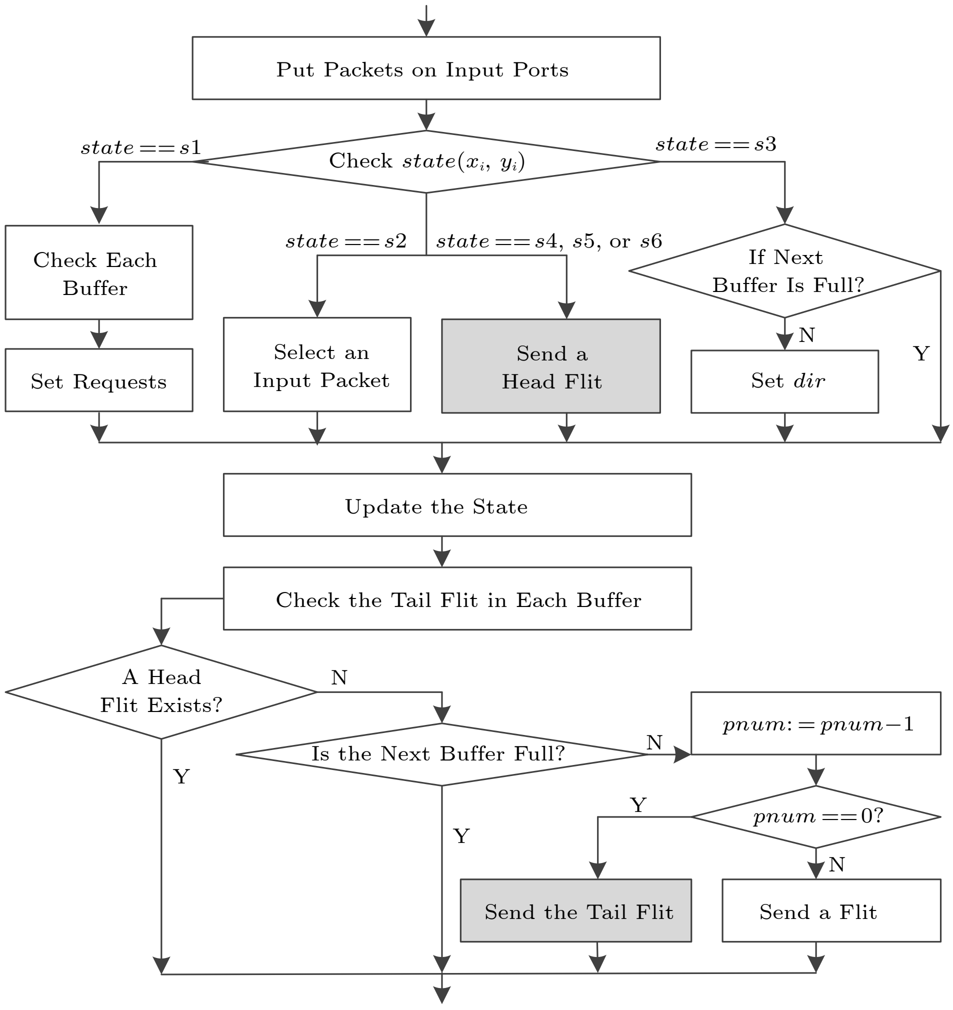

Fig.6. Workflow of the high-level NoC simulator.

ate the test quality of a given configuration of test packets efficiently.

The proposed parallel test method abstracts the test packets generated in Section 4 as test scenarios. First, since the BSBST method generates four test packets for a buffer, the parallel test has to consider four test scenarios corresponding to these packets for each buffer. Second, properties in the BMC cover all transmission turns from an input port to each output port in a switch. However, some turns are not testable due to the XY algorithm used for routing packets through the network[2]. The method will therefore consider only the packets corresponding to these testable turns as test scenarios. Third, the method also abstracts the test packets for the arbitrator as test scenarios. The parallel test evaluates the test coverage of a configuration by counting these test scenarios on every router. (1) presents the test coverage (i.e., cov), where N_{scen} and sum_{scen} denote the amount of the newly detected scenarios and the total scenarios, respectively. Once a test scenario mentioned above firstly appears on a buffer or a switch, this method increases N_{scen} by 1.

$$cov = \frac{N_{\text{scen}}}{sum_{\text{scen}}}, \tag{1}$$

$$objective = \frac{cov}{\sqrt{num_{\text{flit}}} \times \dfrac{time_{\text{cur}}}{time_{\text{pre}}}}. \tag{2}$$

Furthermore, (2) presents the objective formula to evaluate the current configuration. The numerator part is the test coverage, indicating the number of scenarios newly detected by the configuration. The denominator contains two parts. The first is the square root of the flit number (num_{flit}) in the configuration. The square root operation can avoid a small packet covering a few test scenarios to obtain a large objective value. The second is the ratio of the completing time after adding the configuration (i.e., $time_{\text{cur}}$)

to the originally completing time (i.e., $time_{\text{pre}}$). Initially, we assume that the original completing time is the longest transmission time of a single packet. This objective formula, therefore, not only evaluates the test coverage without applying the time-consuming fault simulation but also considers the flit number and the test time.

4.4 Multi-Threading Test Generation

Generating the packet configurations using the Monte-Carlo simulation remains time-consuming, even with the high-level simulator and quick evaluating method. On the other hand, the simulation of different configurations is independent of each other, and the current computing platform often supports running multi-threading programs. Therefore, the proposed PSBST technique uses the multi-threading technology to generate parallel packets for testing kilo-core NoCs more efficiently.

Fig.7 presents the multi-threading algorithm to generate a sequence of packet configurations. This algorithm in Fig.7(a) contains four types of threads: main threads, configuration threads, simulation threads, and statistics threads. Fig.7(b) presents the process of a configuration thread. First, the thread

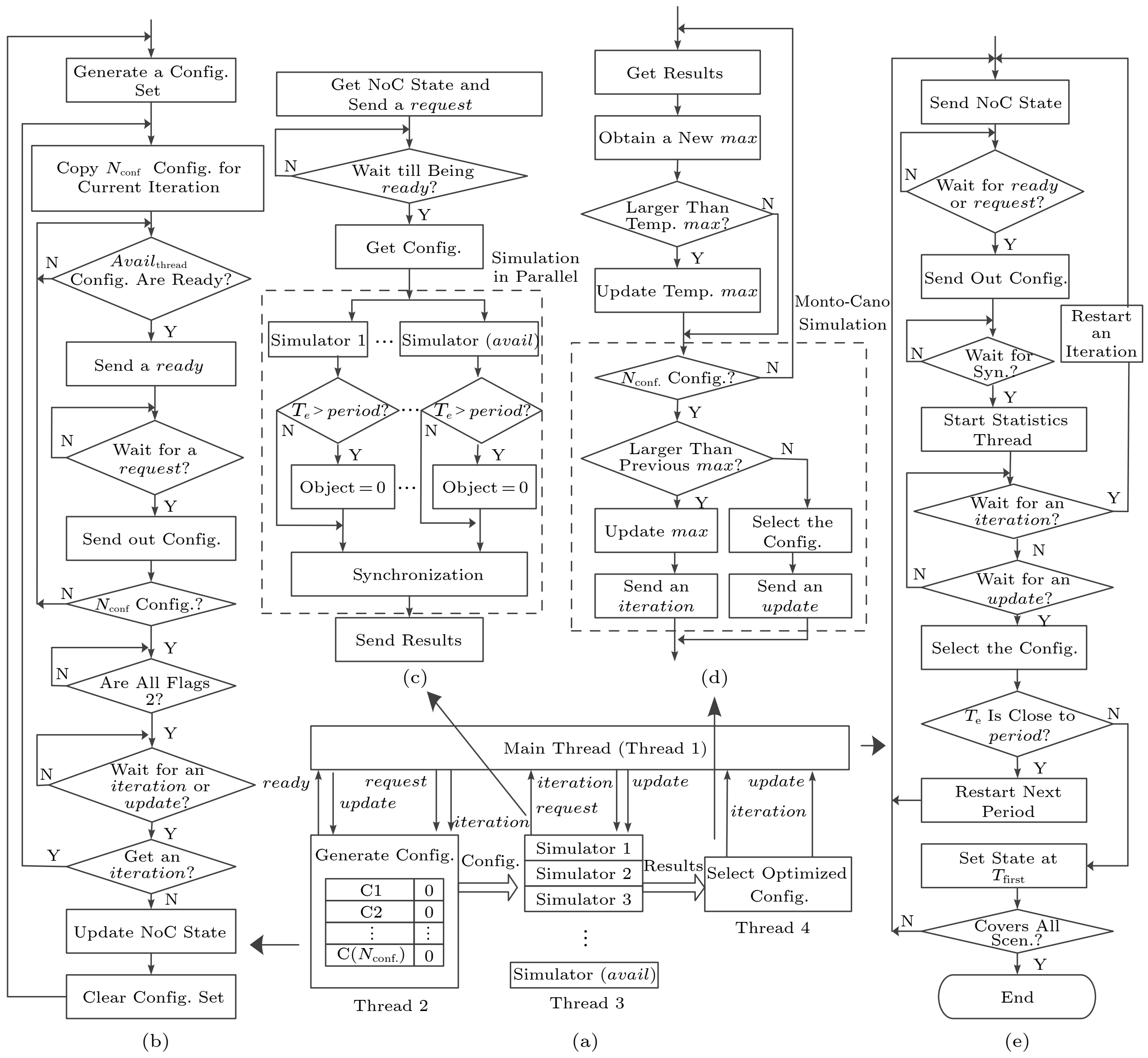

Fig.7. Multi-threading algorithm for generation parallel packets. (a), (b), (c), (d), and (e) are for the algorithm mainframe, configuration thread flowchart (thread 2), simulation thread flowchart (thread 3), statistics thread flowchart (thread 4), and main thread flowchart (thread 1), respectively. Config. means configuration; Temp. means temporary; Scen. means scenarios.

has generated a large number of different configurations of test packets in the temporary storage. Specifically, the thread counts free input ports under the current NoC state and selects $N_{\rm port}$ ports from these free ports. Then it randomly chooses the test packets as discussed in Subsection 3.5, whose source and destination addresses are also random, for these selected ports, i.e., a configuration of packets. Later, the thread copies $N_{\rm conf}$ configurations into a table for one Monte-Carlo iteration and individually sets the flag of a ready configuration as 1. When the number of ready configurations reaches the number of available threads supported by the computing platform (i.e., $avail_{\rm thread}$), the thread sends a *ready* message to the main thread. Once a *request* message arrives, the thread sends $avail_{\rm thread}$ configurations to the simulation threads and sets their flags to 2. Once the thread generates $N_{\rm conf}$ configurations whose flags become 2, it empties the table. If the thread receives an *iteration* message, it copies other $N_{\rm conf}$ configurations from the temporary storage. If the thread receives an *update* message, it updates these free input ports according to the updated NoC state and returns to the first step.

Fig.7(c) presents the process of simulation threads, which are no more than the available threads supported by the computing platform. These threads first obtain the current NoC state from the main thread and send a *request* message to the main thread. Once a *ready* message arrives, the simulation threads can obtain $avail_{\rm thread}$ configurations. Then each simulation thread simulates a configuration on the high-level simulator. If the completing time (i.e., T_e) is within the idle period of the kilo-core NoC, the thread derives the objective value of the configuration. Otherwise, the thread abandons the configuration. The thread also adds a synchronization operation after itself. When all simulation threads are completed, they send their results to the statistics thread.

Fig.7(d) presents the process of the statistical thread. First, this thread receives the results submitted by the simulation threads and selects the maximum objective value and its corresponding configuration from these results. Then, the thread compares this value with the previous maximum value in the temporary storage. If the new value is greater than the previous one, the thread saves this new value in the temporary storage. Next, the thread checks whether the number of simulated configurations reaches $N_{\rm conf}$. If so, the thread compares the current maximum values with the one achieved in the previous Monte-Carlo iteration (if it exists). If the current one is larger, the thread selects the current configuration and sends an *iteration* message to the main thread. Then the algorithm starts another Monte-Carlo iteration. Otherwise, it obtains the optimum configuration and sends an *update* message to the main thread.

Fig.7(e) presents the process of the main thread. The thread first starts the configuration thread and transmits the NoC state to the simulation threads. If the main thread receives a *ready* message and a *request* message, it imposes the generated configurations on the simulation threads. If all synchronization operations in the simulation threads are triggered, the main thread activates the statistics thread. If an *iteration* message emerges, the main thread starts another Monte-Carlo iteration. If an *update* message arrives, the main thread selects the optimum configuration and applies it to the NoC. If the configuration's completing time is very close to the idle period, the thread stores the NoC's state at the completing time and clears all temporary variables. The thread then restarts from the stored NoC state to generate packet configurations for the next idle period. Otherwise, the main thread simulates this configuration until a packet arrives at its destination core for the first time (i.e., $T_{\rm first}$). It updates the NoC's state with that at $T_{\rm first}$ and continues to test the undetected scenarios. The thread is terminated when it covers all test scenarios. This multi-threading technology ensures that the Monte-Carlo simulation can derive the approximately optimum configuration in a large random space and reduce the generating time.

5 Experimental Results

In this section, we use the parallel SBST to test kilo-core NoCs and evaluate the fault coverage and the test cost of this approach, where the NoC used in all experiments is made up of Hermes routers[④]. First, we develop the SBST with BMC to test the Hermes router[④]. Since the fault simulation of a kilo-core NoC is very time-consuming, we set the flit width to 16. Hence the fault simulation can be completed within reasonable time (e.g., a month). Besides, we will use

[④]de Mello A V, Möller L H. Hermes NoC.https://www.inf.pucrs.br/~calazans/research/Projects/Hermes/Hermes.html, Mar. 2023.

the existing full-scan method (i.e., FScan[14]), a functional testing method (i.e., FTest)[20], and the structural SBST (i.e., SSBST) on the router for comparisons[23].

Second, we perform experiments with the proposed parallel algorithm to generate a sequence of packet configurations by Monte-Carlo simulation, where the test packets generated in the previous step are transformed into test scenarios. In the test generation, N_{conf} is a critical parameter. If this parameter is set to be too large, since it is a high-order factor in the time complexity of this algorithm, the algorithm will be difficult to terminate due to the long executing time; if this parameter is too small, too few configurations in each iteration will be generated and it may affect the quality of the generated solution. In this work, we have gotten a good trade-off between the quality of the Monte-Carlo simulation and our platform's computing power, and set N_{conf} to 1 000, based on several experiments.

Third, we test the kilo-core NoC with the Hermes router using the configurations of parallel packets and evaluate their test performance and cost, and we also use the full-scan test and the random test as references. Since the fault simulation on this kilo-core NoC's netlist is extremely time-consuming, we have to reduce the buffer size from 16 to 4, and the maximum flit number per packet from 31 to 15. Next, we synthesize the kilo-core NoC into the netlist using the 45 nm technology node, where this netlist occupies 5.31 million μm^2 area and has about 17.5 million stuck-at faults. Then, we simulate the sequence of packet configurations on the netlist to derive the VCDE file, and finally evaluate the fault coverage of the kilo-core NoC. Furthermore, the proposed multi-threading program runs on a workstation with AMD R5 5600x CPU and 64 G memory, which supports 12 threads, and the EDA platform for fault simulation runs on a server with Intel Xeon 6148 and 256 G memory (supporting 160 threads).

5.1 Multi-Threading Test Generation

The Monte-Carlo simulation can find an approximately optimum packet configuration. The configuration not only covers test scenarios as many as possible but also minimizes the test cost. Fig.8 presents the objective values achieved by each Monte-Carlo iteration starting from the NoC's initial state. Although the Monte-Carlo simulation algorithm selects

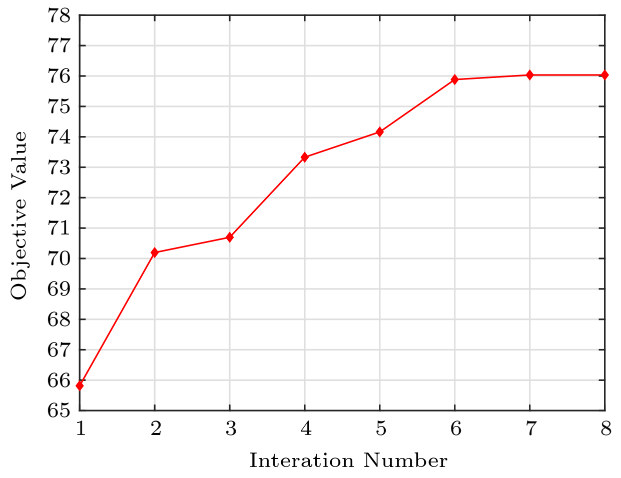

Fig.8. Objective values of Monte-Carlo iterations.

an optimized configuration from 1 000 different configurations, in each iteration, the proportion of these configurations in the entire configuration set is still very low, and another configuration may get a better objective value. Hence the parallel test starts another Monte-Carlo iteration to check if a better configuration exists. The algorithm will continue the Monte-Carlo iterations until no more optimized configuration emerges. In Fig.8, the maximum objective value increases from 66 to 76 after eight Monte-Carlo iterations and no longer grows. In this way, the algorithm obtains an approximately optimum configuration. This configuration is not necessarily the optimum one, but most configurations cannot achieve the objective value comparable to that of this configuration.

The high-level simulator and the multi-threading technology effectively support the Monte-Carlo simulation. The histogram in Fig.9 presents the Monte-

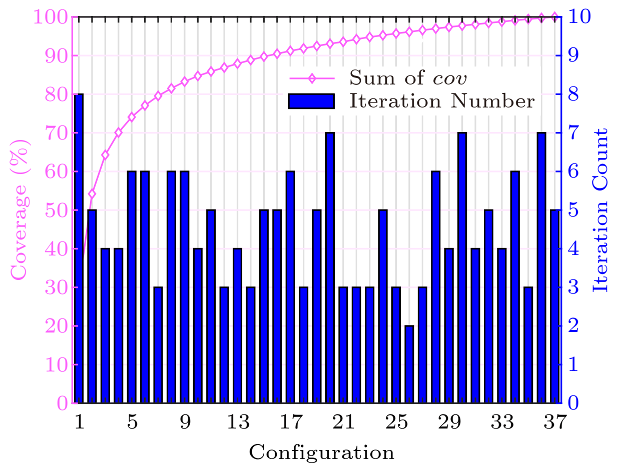

Fig.9. Iteration number and the sum of *cov* for the configuration sequence. *cov* means the coverage rate.

Carlo iteration number that the parallel test performs each time when the algorithm generates the new configuration of packets to be injected into the NoC. The iteration number ranges from 2 and 8. Since the parallel test evaluates 1 000 configurations per iteration, it has to simulate at least 2 000 configurations to derive an optimum one. The gate-level simulator runs slowly, and its commercial license strictly limits its running in parallel. As a comparison, the proposed high-level simulator significantly reduces the simulation time for each packet configuration and maintains cycle accuracy with the gate-level simulator. It thus underpins the Monte-Carlo simulation as an appropriate evaluation technique. Furthermore, although a large number of configurations have to be evaluated, these configurations are independent of each other. In this case, the multi-threading technology can take advantage of the many-core hardware platforms used to run the simulation program and reduce the overall simulation time significantly. The technology thus supports the Monte-Carlo simulation to search for the optimum configurations of test packets efficiently.

Finally, the high-level evaluation assists the parallel algorithm in covering all test scenarios. In the objective formula, the number of newly activated scenarios is the numerator. If a configuration does not activate any new scenario and its value becomes 0, it will be eliminated. This evaluation that keeps purging out configurations with value 0 will guide the Monte-Carlo simulation in covering new scenarios and eventually activate all test scenarios. The curve in Fig.9 shows that the cumulative coverage ratio of tested scenarios increases until it reaches 100%. Besides, the high-level evaluation greatly reduces the evaluating time of each configuration. As a comparison, the gate-level fault simulation on this kilo-core NoC requires hundreds of hours for one configuration. Finally, the evaluation counts the newly-activated scenarios on the entire NoC. Once a new scenario is activated, the evaluation removes it from the untested scenario set. In this way, the parallel test would not select the configuration that repeatedly activates tested scenarios and thereby reduces the overhead of online testing.

5.2 Multi-Threading Test Generation

The SBST with BMC effectively tests the routers in the functional mode. This method leads to extremely high fault coverage (i.e., 98%), which is very close to that obtained with the full-scan method. In particular, the proposed method is highly effective for testing the data registers in the buffers. In Table 2, the fault coverage for the buffers equals that obtained using the full-scan test. Although the full-scan test leads to the highest fault coverage, it cannot test the NoC in the functional mode. Compared with functional testing[20], the BSBST is more effective for testing the sequential unit. Although the insertion of testability logic at the ports[20] can increase the fault coverage for the datapath of an NoC, a major problem still lies in the testing of the sequential units. Table 2 shows that the fault coverage provided by functional testing[20] for the buffer and the arbitrator is far less than that provided by the proposed method. Compared with structural SBST[23] (i.e., SSBST), the SBST with BMC is more attractive for circuits with large sequential depth. For example, although the activation of the counter in the buffer requires more than 10 timeframes, the BSBST still leads to 100% fault coverage on that unit regardless of its large sequential depth. In the literature, structural SBST considers usually only a small number of timeframes (3 or 5), and thus it may not be able to test such a control unit with large sequential depth. Finally, we conclude that the proposed SBST method with BMC outperforms competing methods for the NoC router.

Table 2. Fault Coverage (%) on the Router Using Different Methods

	Buffer	Arbitrator	Router
FScan (16 bits)	--	--	100.0
FTest[20] (32 bits)	88.9	76.5	85.0
SSBST[23] (8 bits)	--	--	96.5
BSBST (16 bits)	99.3	89.4	98.0

Note: --: the fault coverage is not known for prior methods.

The parallel SBST effectively tests the kilo-core NoC used in our experiments. As shown in Table 3, the PSBST method achieves a total of 94.08% fault coverage on the kilo-core NoC. This method abstracts the test packets generated by BSBST into test scenarios and then uses the Monte-Carlo simulation to ensure that all test scenarios emerge on each router. Since these test packets have an excellent test performance and have been successfully imposed on each router by the parallel method, the PSBST test achieves a high fault coverage on the NoC. We have changed the buffer size from 16 to 4 due to the time-consuming fault simulation. As a result, the proportion of faults related to external signals that cannot

Table 3. Comparison Between Different Test Methods on the Kilo-Core NoC in Terms of Test Cost and Overheads

		FC. (%)	Area (%)	Gen. Time (s)	App. Time (cycle)	Data Vol. (KB)
Fscan		100.00	17.14	2 242.5	8 130 000 000	992 000.00
Random	Buf.	64.17				
	Swit.	53.61	0.00	804.0	97 430	1 980.00
	All	57.27				
PSBST	Buf.	98.17				
	Swit.	87.73	0.00	59 883.0	81 815	402.25
	All	94.08				

Note: FC.: fault coverage; Gen. Time: generating time; APP. Time: application time; vol: volume.

be directly tested by SBST rises up. Hence, the buffer's fault coverage has decreased compared with that of the BSBST method in Table 2. The Herms router contains all turn channels from one input port to any output port. However, many turn channels cannot be activated under the XY routing algorithm. Even if the circuit corresponding to such a turn has a fault, the NoC function will not have any impact. Therefore, considering the over-testing problem, the PSBST with a lower fault coverage has actually a higher test quality than the full-scan method.

The existing methods cannot meet the online test requirements of kilo-core NoCs. First, the full-scan test is a robust method for testing kilo-core NoCs. In Table 3, the method achieves 100% fault coverage. However, the full-scan method requires the use of external test equipment to inject test data and a dedicated test mode beyond the normal operation. Therefore, this method is only suitable for manufacturing testing but not for online self-testing. Second, the random test approach hardly achieves any reasonable fault coverage. Random packets cannot have a high test quality as they often repeatedly test easy-to-test faults. Furthermore, the scale of a kilo-core NoC is huge, and the packet configurations generated with the random test approach correspond only to a small proportion of the entire legal configurations. Fig.10 presents the fault coverages that a random test approach achieves with multiple times of the packet amount used in PSBST. As the packet amount increases from one time (e.g., 1x) to four times (e.g., 4x), the growth rate of fault coverage gradually slows down, and the fault coverage reaches 57.27% at most. Therefore, the random test cannot meet the online test requirement of kilo-core NoCs due to its low fault coverage.

Next, we use a set of heatmaps to illustrate how the PSBST covers the buffers and routers at different

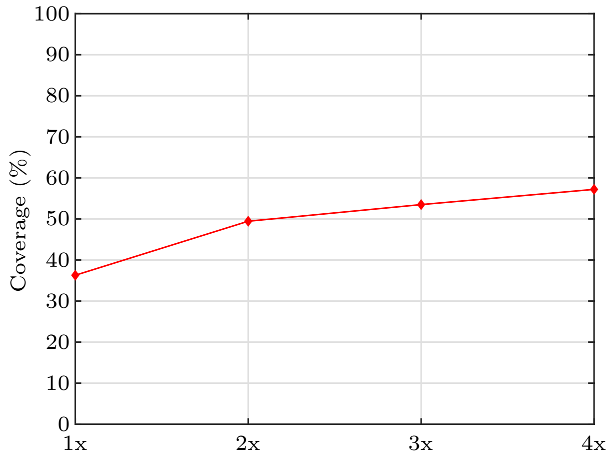

Fig.10. Fault coverage using different data volumes.

locations of the kilo-core (i.e., 32×32) NoC, where the value of the horizontal and vertical axes represent the X and Y coordinates of a node respectively. The heatmaps in Fig.11(a) and Fig.11(b) present the fault coverage of the buffers (the average coverage of five buffers) in each router achieved by PSBST and the random test, respectively. PSBST achieves a higher fault coverage on every buffer of the kilo-core NoC under test. That is because the packets generated by BSBST have an excellent test performance, and the parallel test ensures that these packets appear on every buffer. As a comparison, the random test has a much lower fault coverage on most buffers, and the fault coverage of the internal routers is even lower. That is because many cells in the buffers are not being completely triggered by random packets. Besides, the routers at the periphery of the kilo-core NoC under test are usually connected to the outside through the boundary scan or JTAG (Joint Test Action Group), while internal routers are difficult to access.

PSBST effectively tests the routers anywhere in the kilo-core NoC while avoiding the over-testing problem at the same time. First, many transmission turns in the internal router will not be triggered in the functional mode, i.e., they are functionally untestable. For example, packets from the eastern input port of an internal router cannot be forwarded to the eastern output port. According to the XY routing algorithm, the X coordinate of the destination node of the packet should not be more than the X coordinate of the current router. However, forwarding to the eastern output port requires the X coordinate of the destination node to be greater than that of the current router. Furthermore, the packet from the southern input port cannot be forwarded to the east-

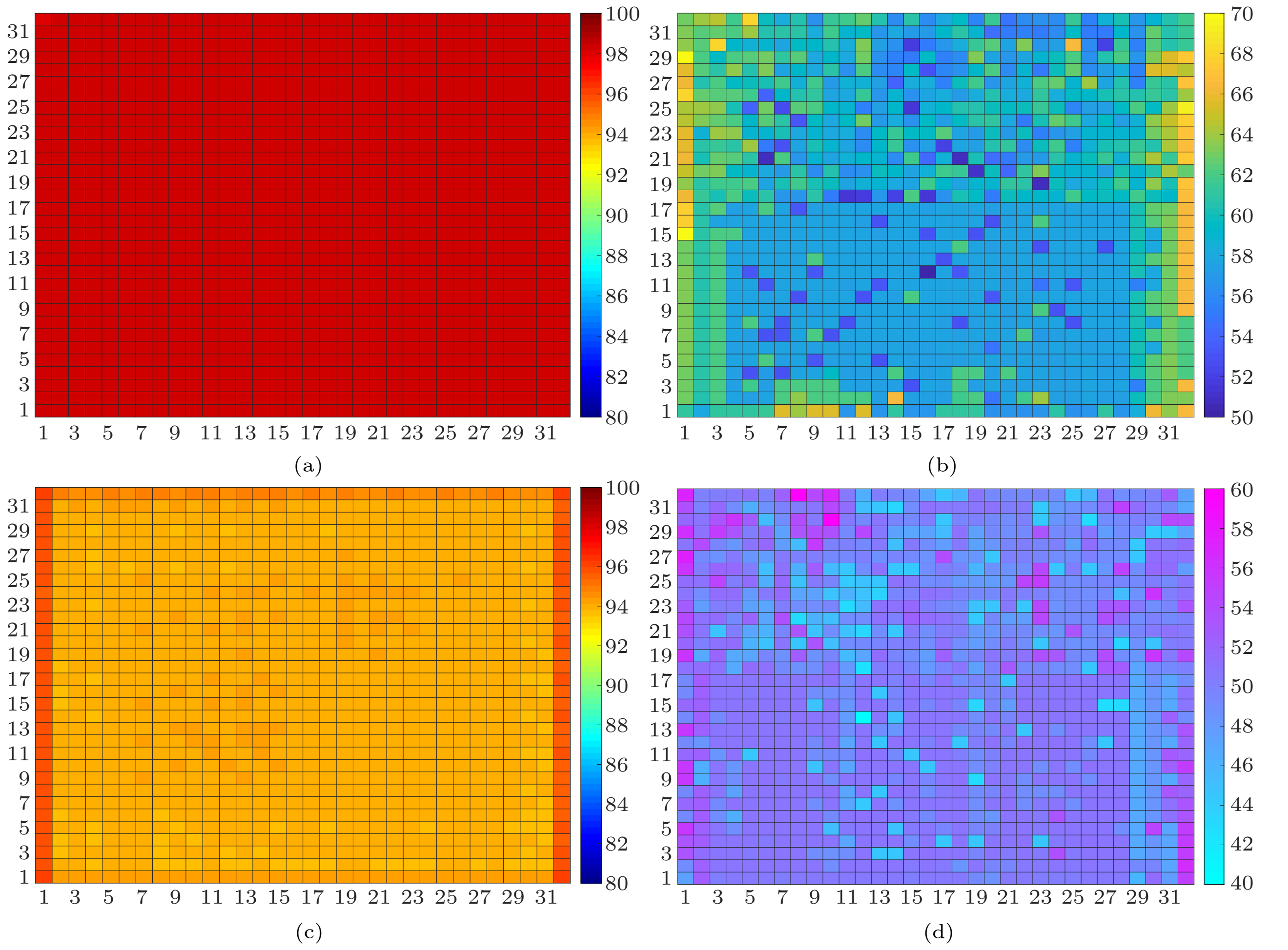

Fig.11. Heatmaps for fault coverage on different positions using PSBST and the random test. (a) Fault coverage of buffers using PSBST. (b) Fault coverage of buffers using random test. (c) Fault coverage of routers using PSBST. (d) Fault coverage of routers using random test.

ern output port. This is because the XY routing algorithm needs to prioritize forwarding packets along the X direction instead of the Y direction. In this case, an internal router contains 25 turn channels, but only 68% (=(25–4–4)/25) of them are functionally testable. PSBST will achieve higher fault coverage on routers if other routing algorithms have fewer untestable turns.

Meanwhile, the arbitrating circuit is also difficult to test as it requires multiple packets to simultaneously arrive at the router. This requirement is difficult to meet in a kilo-core NoC, especially for the switch in an internal router. PSBST sets all feasible turns in a router as test scenarios, and the Monte-Carlo simulation will not terminate until it covers all these scenarios. The high-level simulator also maintains cycle accuracy with the gate-level simulator. This ensures test packets arrive at the router under test on time and thus effectively test the arbitrating circuit. The heatmap in Fig.11(c) presents the fault coverage of each router achieved by PSBST. PSBST achieves a higher fault coverage at each router. Since the edge nodes of the kilo-core NoC under test can obtain packets from the external boundary scan or JTAG, they have more testable turns and thus higher fault coverages. As a comparison, the fault coverages of the random test are generally low and greatly fluctuate among different routers in the heatmap, as shown in Fig.11(d). We thereby conclude that the parallel SBST method is effective in testing kilo-core NoCs.

5.3 Test Cost on the Kilo-Core NoC

Another reason why PSBST is desirable for online testing lies in its negligible cost. Table 3 shows also the test cost of the proposed method, where the full-scan technique and the random test are used as references. Since the PSBST method tests an NoC by transmitting packets among the NoC routers, it does not require any intrusive hardware or any area overhead. In contrast, the full-scan technique introduces 17.14% (i.e., 0.91 million μm^2) of the total area, which is very large due to the large scale of the kilo-core NoC. Besides, the PSBST program is executed in the functional mode; hence it does not require any expensive external ATE.

Table 3 also shows the test software generating time for the PSBST method. First, the generating time (i.e., Gen. time) includes the solving time of BMC. Since the slicing technique significantly compresses the state machines of the buffers and NoC, the entire process only takes 2.48 seconds. Second, the generating time also involves the executing time of the Monte-Carlo simulation. Since each iteration of the Monte-Carlo simulation has to simulate 1 000 different configurations of packets, and PSBST requires multiple iterations to derive an approximately optimum configuration, the Monte-Carlo simulation would be time-consuming. However, PSBST uses multi-threading technology, as discussed in Subsection 5.1, which significantly reduces the execution time. Fig.12 shows the generating time for each configuration, which is at an acceptable level. The executing time can be further reduced if a computing platform with more processor cores is employed.

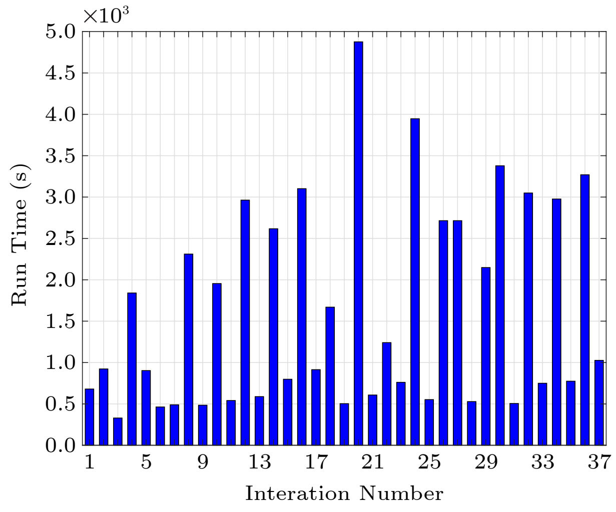

Fig.12. Iteration numbers and the sum of *cov* for the configuration sequence.

Finally, we note that the test-data volume (i.e., Data vol.) and the test application time (i.e., App time) for the proposed PSBST method are reasonable. Since the high-level simulator counts the number of detected scenarios in the entire kilo-core NoC, PSBST can preferentially select the test packet that covers multiple test scenarios. Once PSBST selects a test packet, it will delete the scenarios that the packet covers. In this way, the packet that triggers the tested scenarios again will have a minimum objective value (i.e., 0). PSBST will not select that packet. Furthermore, PSBST uses the Monte-Carlo simulation to minimize the test cost. As a comparison, the random test fails to effectively test routers and repeatedly tests the easy-to-test regions of the kilo-core NoC. Therefore, although the random test uses almost four times the number of test packets of PSBST, it achieves only 57.27% fault coverage. The original full-scan test does not exploit the NoC's characteristics to carry out parallel testing. It needs to connect all internal registers into a scan chain, resulting in extremely large test patterns. It needs also to shift these very large patterns into (out of) the NoC through the scan chain, causing a very large test application time. In this way, the full-scan test is orders of magnitude higher than PSBST in terms of test-data volume and test application time. Therefore, the experimental results demonstrate that PSBST is efficient and effective for the online testing of the kilo-core NoC in the functional mode.

6 Conclusions

We developed a parallel SBST technique to support the efficient online detection of faults in kilo-core networks-on-chip. The proposed PSBST method can automatically generate a configuration sequence of parallel packets and effectively and efficiently test a large NoC in the functional mode. First, the proposed SBST technique makes use of a BMC algorithm to derive the leading sequence for each router's internal function and detect all functional-testable faults corresponding to each router's internal function. Second, a Monte-Carlo simulation algorithm ensures the proposed technique to search for the optimum configuration of the parallel packets, which not only guarantees the test quality but also minimizes the test cost. Finally, the multi-threading technology is used to ensure that the Monte-Carlo simulation can find the approximately optimum configuration in a large random space and reduce the generating time of the parallel tests. Experimental results showed that the proposed method leads to a high test quality and avoids the over-testing problem caused by functionally untestable turns in the functional mode.

References

[1] Zhang Y, Hong X P, Chen Z S, Peng Z B, Jiang J H. A deterministic-path routing algorithm for tolerating many faults on very-large-scale network-on-chip. *ACM Trans. Design Automation of Electronic Systems*, 2021, 26(1): Article No. 8. DOI: 10.1145/3414060.

[2] Zhang Y, Chakrabarty K, Li H W, Jiang J H. Software-based online self-testing of network-on-chip using bounded model checking. In *Proc. the 2017 IEEE International Test Conference*, Oct. 31–Nov. 2, 2017. DOI: 10.1109/TEST.2017.8242037.

[3] Dongarra J. Report on the Sunway TaihuLight system. Tech Report UT-EECS-16-742, Oak Ridge National Laboratory, 2016. https://netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016-old.pdf, Mar. 2023.

[4] Zhang L, Han Y H, Xu Q, Li X W, Li H W. On topology reconfiguration for defect-tolerant NoC-based homogeneous manycore systems. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2009, 17(9): 1173–1186. DOI: 10.1109/TVLSI.2008.2002108.

[5] Ouyang Y M, Da J, Wang X M, Han Q Q, Liang H G, Du G M. A TSV fault-tolerant scheme based on failure classification in 3D-NoC. *Journal of Circuits, Systems and Computers*, 2017, 26(4): 1750059. DOI: 10.1142/S0218126617500591.

[6] Chen Z S, Zhang Y, Peng Z B, Jiang J H. A deterministic-path routing algorithm for tolerating many faults on wafer-level NoC. In *Proc. the 2019 Design, Automation and Test in Europe Conference & Exhibition*, Mar. 2019, pp.1337–1342. DOI: 10.23919/DATE.2019.8714948.

[7] Lee D, Das S, Doppa J R, Pande P P, Chakrabarty K. Performance and thermal tradeoffs for energy-efficient monolithic 3D network-on-chip. *ACM Trans. Design Automation of Electronic Systems*, 2018, 23(5): Article No. 60. DOI: 10.1145/3223046.

[8] Liu W C, Yang L, Jiang W W, Feng L, Guan N, Zhang W, Dutt N. Thermal-aware task mapping on dynamically reconfigurable network-on-chip based multiprocessor system-on-chip. *IEEE Trans. Computers*, 2018, 67(12): 1818–1834. DOI: 10.1109/TC.2018.2844365.

[9] Xiang D, Chakrabarty K, Fujiwara H. Multicast-based testing and thermal-aware test scheduling for 3D ICs with a stacked network-on-chip. *IEEE Trans. Computers*, 2016, 65(9): 2767–2779. DOI: 10.1109/TC.2015.2493548.

[10] Wang L, Wang X H, Leung H F, Mak T. A non-minimal routing algorithm for aging mitigation in 2D-mesh NoCs. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2019, 38(7): 1373–1377. DOI: 10.1109/TCAD.2018.2855149.

[11] Xiang D, Shen K L, Bhattacharya B B, Wen X Q, Lin X J. Thermal-aware small-delay defect testing in integrated circuits for mitigating overkill. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2016, 35(3): 499–512. DOI: 10.1109/TCAD.2015.2474365.

[12] Biere A, Cimatti A, Clarke E M, Strichman O, Zhu Y S. Bounded model checking. *Advances in Computers*, 2003, 58: 117–148. DOI: 10.1016/S0065-2458(03)58003-2.

[13] Clarke E, Biere A, Raimi R, Zhu Y S. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 2001, 19(1): 7–34. DOI: 10.1023/A:1011276507260.

[14] Cota E, Liu C. Constraint-driven test scheduling for NoC-based systems. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2006, 25(11): 2465–2478. DOI: 10.1109/TCAD.2006.881331.

[15] Yuan F, Huang L, Xu Q. Re-examining the use of network-on-chip as test access mechanism. In *Proc. the 2008 Conference on Design, Automation and Test in Europe*, Mar. 2018, pp.808–811. DOI: 10.1145/1403375.1403571.

[16] Richter M, Chakrabarty K. Optimization of test pin-count, test scheduling, and test access for NoC-based multicore SoCs. *IEEE Trans. Computers*, 2014, 63(3): 691–702. DOI: 10.1109/TC.2013.82.

[17] Strano A, Gómez C, Ludovici D, Favalli M, Gómez M E, Bertozzi D. Exploiting network-on-chip structural redundancy for a cooperative and scalable built-in self-test architecture. In *Proc. the 2011 Design, Automation and Test in Europe*, Mar. 2011. DOI: 10.1109/DATE.2011.5763109.

[18] Wang J S, Ebrahimi M, Huang L T, Xie X, Li Q, Li G J, Jantsch A. Efficient design-for-test approach for networks-on-chip. *IEEE Trans. Computers*, 2019, 68(2): 198–213. DOI: 10.1109/TC.2018.2865948.

[19] Herve M B, Cota E, Kastensmidt F L, Lubaszewski M. NoC interconnection functional testing: Using boundary-scan to reduce the overall testing time. In *Proc. the 10th Latin American Test Workshop*, Mar. 2009. DOI: 10.1109/LATW.2009.4813801.

[20] Kakoee M R, Bertacco V, Benini L. At-speed distributed functional testing to detect logic and delay faults in NoCs. *IEEE Trans. Computers*, 2014, 63(3): 703–717. DOI: 10.1109/TC.2013.202.

[21] Kranitis N, Merentitis A, Theodorou G, Paschalis A, Gizopoulos D. Hybrid-SBST methodology for efficient testing of processor cores. *IEEE Design & Test of Computers*, 2008, 25(1): 64–75. DOI: 10.1109/MDT.2008.15.

[22] Collet J H, Zajac P, Psarakis M, Gizopoulos D. Chip self-organization and fault tolerance in massively defective multicore arrays. *IEEE Trans. Dependable and Secure Computing*, 2011, 8(2): 207–217. DOI: 10.1109/TDSC.2009.53.

[23] Dalirsani A, Imhof M E, Wunderlich H J. Structural software-based self-test of network-on-chip. In *Proc. the 32nd VLSI Test Symposium*, Apr. 2014. DOI: 10.1109/VTS.2014.6818754.

[24] Cheng K T, Krishnakumar A S. Automatic functional test generation using the extended finite state machine model. In *Proc. the 30th International Design Automation Conference*, June 1993, pp.86–91. DOI: 10.1145/157485.164585.

[25] Psarakis M, Gizopoulos D, Sanchez E, Reorda M S. Microprocessor software-based self-testing. *IEEE Design & Test of Computers*, 2010, 27(3): 4–19. DOI: 10.1109/

MDT.2010.5.

[26] Ganai M K, Gupta A. Accelerating high-level bounded model checking. In *Proc. the 2006 International Conference on Computer Aided Design*, Nov. 2006, pp.794–801. DOI: 10.1109/ICCAD.2006.320122.

[27] Tupuri R S, Abraham J A. A novel functional test generation method for processors using commercial ATPG. In *Proc. the 1997 International Test Conference*, Nov. 1997, pp.743–752. DOI: 10.1109/TEST.1997.639687.

[28] Zhang Y, Li H W, Li X W. Automatic test program generation using executing-trace-based constraint extraction for embedded processors. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2013, 27(7): 1220–1233. DOI: 10.1109/TVLSI.2012.2208130.

[29] Negro V C, Goldstein L H. The generation of correlated multivariate samples for Monte Carlo simulation. *IEEE Spectrum*, 1968, 5(2): 5. DOI: 10.1109/MSPEC.1968.5214753.

Ying Zhang received his B.S. degree in computer science from Harbin Engineering University, Harbin, in 2006, and his Ph.D. degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2011. Now, he is an associate professor at Tongji University, Shanghai. He is a member of CCF and IEEE. His research interests include signal integrity, reliable design of network-on-chip, and software-based self-testing.

Peng-Fei Ji received his Master's degree from School of Software Engineering, Tongji University, Shanghai. His research interests include kilo-core networks-on-chip online testing and software-based self-testing.

Pan-Wei Zhu is an undergraduate student of the School of Software Engineering, Tongji University, Shanghai. His current research interests include routing algorithms of network-on-chip, data mining, and reinforcement learning.

Zebo Peng received his Ph.D. degree in computer science from Linköping University, Linköping, in 1987. He is currently a professor of computer systems, the Director of the Embedded Systems Laboratory, and the Vice-Chairman of the Department of Computer and Information Science, Linköping University, Linköping. He has published over 350 technical papers and five books in various topics related to embedded and cyber-physical systems.

Hua-Wei Li received her B.S. degree in computer science from Xiangtan University, Xiangtan, in 1996, and her M.S. and Ph.D. degrees in computer architecture from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 1999 and 2001, respectively. She has been a professor with ICT, CAS and CAS since 2008. Her current research interests include testing of very large-scale integration/SoC circuits, approximate computing architecture and machine learning accelerators.

Jian-Hui Jiang received his B.E., M.E. and Ph.D. degrees in traffic information engineering in Shanghai Railway University, Shanghai, in 1985, 1988, and 1999, respectively. He is currently a full professor of software engineering and a vice dean of the School of Software Engineering at Tongji University, Shanghai. His current research interests include dependable systems and networks, software reliability engineering, VLSI/SoC testing and fault-tolerance.