Integration, the VLSI Journal 67 (2019) 134–143

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Testing aware dynamic mapping for path-centric network-on-chip test

Shuyan Jiang [a], Qiong Wu [a], Shuyu Chen [a], Junkai Zhan [a], Junshi Wang [a], Masoumeh Ebrahimi [b], Letian Huang [a,*]

[a] University of Electronic Science and Technology of China, Chengdu, China
[b] Royal Institute of Technology (KTH), Sweden

ARTICLE INFO

Keywords:
Network-on-Chip
Mapping algorithm
Intermittent fault
On-line testing

ABSTRACT

With the aggressive scaling of submicron technology, intermittent faults are becoming one of the limiting factors in achieving high reliability in Network-on-Chip (NoC). Increasing test frequency is necessary to detect intermittent faults, which in turn interrupts the execution of applications. On the other hand, the primary goal of traditional mapping algorithms is to allocate applications to the NoC platform, ignoring the test requirement. In this paper, we propose a novel testing-aware mapping algorithm (TAMA) for NoC, targeting intermittent faults on the paths between crossbars. In this approach, the idle paths are identified, and the components between two crossbars are tested when the application is mapped to the platform. The components can be tested if there is enough time from the time when the application leaves the platform to the time when a new application enters it. The mapping algorithm is tuned to give a higher priority to the tested paths in the next application mapping, which leaves enough time to test the links and the belonging components that have not been tested in the expected time. Experiment results show that the proposed testing-aware mapping algorithm leads to a significant improvement over FF(Fiexitrst Free), NN(Nearest Neighbor), CoNA(Contiguous Neighborhood Allocation), and WeNA(Weighted-based Neighborhood Allocation).

1. Introduction

With the development of fabrication process technology, the density of integrated circuits increases continuously. A single chip can integrate more function units than ever before, and the era of many-core has come. Many-core System-on-Chip (SoC) usually integrates different function units (cores mostly) to provide strong function capacity as well as Network-on-Chip (NoC) to provide high-bandwidth, low-latency and flexibility communication [1]. According to the communication characteristics, many-core SoCs can be divided into Chip multiprocessors(CMP) and Multi-Processors System on Chip(MPSoC) [2]. The communication method of CMP and MPSoC is different. Communication in CMP architecture is generated through data access and cache coherency. Meanwhile, MPSoC needs peer-to-peer communication driven by the application. With the help of programming models and frameworks, MPSoC is widely used in embedded systems [3].

To execute in parallel, the application is divided into several tasks with data exchanges. The tasks should execute on different cores while the data is exchanged through communication between cores. The procedure to assign tasks to cores is called the task mapping [4]. For MPSoC, since the application initiates the communication, the inter-task communication is determined before the task mapping, which is usually presented by the communication graph. Thus, mapping does not only assign the tasks but also assigns the communication traffic between tasks to specified links.

A mapping algorithm is designed to guide task mapping [5]. The primary goal of task mapping algorithms is to reduce the execution time of application. The execution time of the application is severely affected by the communication between tasks. The increase of communication latency will extend the execution of application because tasks have to wait for data for forwarding execution. Not only network capacity but also the reliability is crucial for system performance [6]. To guarantee the right results, error data must be corrected through fault-tolerant methods, like retransmission, which introduces significant latency. Avoiding using fault links could avoid error communication.

Fig. 1 shows one example of the mapping algorithm. Task 1–3 is mapped to a 4 × 4 mesh MPSoC according to CoNA mapping algorithm. It is obvious that not all link is occupied after task mapping. These idle

* Corresponding author.
E-mail address: huanglt@uestc.edu.cn (L. Huang).

https://doi.org/10.1016/j.vlsi.2018.11.009
Received 15 May 2018; Received in revised form 12 September 2018; Accepted 20 November 2018
Available online 11 December 2018

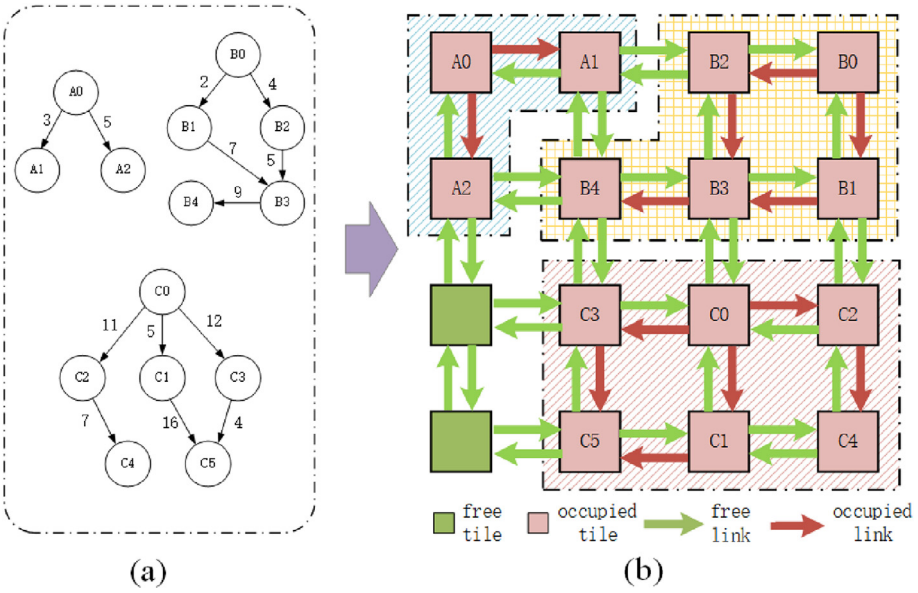

Fig. 1. The Router utilization picture.

links can occur within the same application as well as between different applications. Because the route is determined under deterministic routing algorithms, packets deliver through one of the shortest paths, so not all links within the application are occupied. Because there is no data exchange between applications, links between application are idle mostly. While the applications are executing, these idle links can be used to optimize other features besides communication, such as reliability.

As the integration process progresses, reliability becomes a critical design challenge [7]. More physical failures generate high risks to communication and execution. The reliability of NoCs directly influences the reliability of SoCs and the execution time of applications. Broken links can break up the connections between cores [8], and error packets have to be corrected at the cost of longer execution time [9].

Faults can be divided into transient faults, intermittent faults, and permanent faults [10]. Transient faults occur randomly and recover very quickly. Intermittent faults occur at same devices repeatedly. Permanent faults would not recover once occurred. Intermittent faults are often caused by aging problems and can become permanent faults over time.

Fault-tolerant approaches can detect intermittent faults. After detecting faults, fault-tolerant methods can be triggered to protect communication. It is proved that increasing the test frequency can improve reliability by detecting intermittent faults earlier.

As shown in Fig. 2 from Ref. [11], there are two scenarios examined. The fault rate is about 4.6% for FM1 and 2.2% for FM2 while without any test. By including BIST, the fault rate reduces when shortening the test period (increasing the test frequency). With the test period of 20 K cycles, the fault rates have been reduced by two orders of magnitude, which are around 0.03% for both scenarios. Therefore, increasing the test frequency helps in detecting faults faster and thus reducing the fault rate and increasing system reliability. Therefore, early detection of intermittent faults can trigger fault-tolerant methods early, reducing the impact on the system while extending age. By improving the test frequency, the components are tested more times in a period, once the intermittent fault generated, it can detect the fault in a short time, and repair it immediately, make sure the system reliability not be affected.

Increasing the test frequency is good for reliability, but it is not good for performance. The test methods will interrupt the normal communication. The circuit under test must be isolated from the entire network.

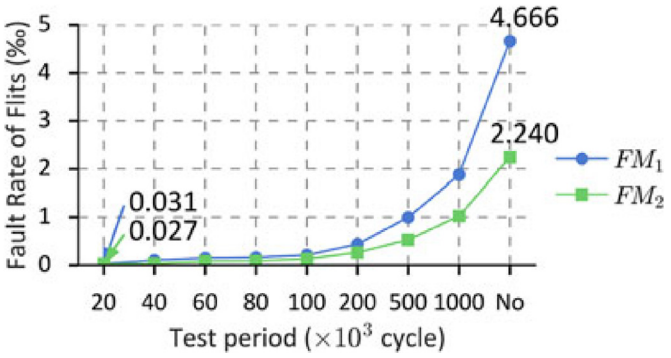

Fig. 2. Flits fault rate under periodic BIST with different test periods [11].

The links under test cannot deliver any packets. Packets have to be blocked in the routers until the test finishes. The application has to wait for a longer time before got the necessary data and slows down. As shown in Ref. [12], the execution time increases up to 4× when the test frequency increases.

On the other side, idle links in the NoC can be used for testing without interrupt normal traffic. Testing idle links does not introduce extra latency. If the mapping algorithm can assign communications to the tested link and leave the untested link free, All links can be tested in a short period. Thus, the mapping algorithms can be an important role in increasing testing frequency in NoC.

In order to improve the test frequency and reduce the impact on the system, we propose a novel testing-aware mapping algorithm (TAMA) to help test with idle links. The contribution of this work includes:

1. The mapping algorithm considers the reliability of the link. While mapping applications, it will try to use the links that have been tested, at the same time, avoid using the link used by the previous mapping. Because the link tested and the link used last time may not have a high probability of having faults so that it can shorten the test period of the link, detect intermittent faults early, and improve reliability.
2. Each time after mapping the application, there will be some idle links. This mapping algorithm provide a test strategy for the map-

ping algorithm it will test the idle link; and interrupt the test when the link is used, giving priority to the performance of the application.
3. Finally, a path-based NoC test method matching the Mapping algorithm is provided.

In this paper, the basement of our design is that the NoC must be a regular network and a deterministic route. Since Build-in-Self-Test (BIST) is wildly used in many testing fields, the BIST core is not in the discussion of this paper.

The remainder of this paper is organized as follows: Section 2 presents the related work about the NoC test and mapping algorithms. The mapping problems along with the concept of idle links in mapping are formally modeled in Section 3. The testing method architecture is given in Section 4. In Section 5, we propose an optimized testing-aware mapping algorithm. Section 6 presents and discusses the simulation results. Finally, the conclusion is given in the last section.

2. Related work

The essence of task mapping is to assign applications to different cores to find the optimal solution while satisfying the constraints. A good mapping algorithm can speed up task allocation and increase system frequency, thereby increasing test frequency and improving system reliability.

The mapping algorithm can be roughly divided into two categories: dynamic mapping and static mapping. Static mapping is a design-time optimization that is typically used to optimize a specific-purpose MPSoC architecture for on-chip multi-core systems. Since the optimization of the design stage can allow the optimization algorithm to run for a long time without having to consider the complexity of the algorithm itself, some optimization algorithms such as genetic algorithm and ant colony algorithm are introduced into the static mapping algorithm. The dynamic mapping algorithm is a run-time optimization, which is mainly applied to the MPSoC architecture on-chip multi-core system for general purpose use.

[13] started discussing dynamic mapping algorithms. Several dynamic mapping algorithms proposed in the paper are often used as the baseline for comparison with other mapping algorithms. Based on several typical dynamic mapping algorithms proposed in Ref. [13], several subsequent papers have continuously improved the dynamic mapping algorithm.

FF (first free) is one of the most fundamental algorithms proposed in Ref. [13]. This algorithm maps tasks to the first idle PE searched by the algorithm according to the normal order of the task graph. Search row by row or column by column. The FF algorithm has almost no optimization for the mapping process, so it can be compared as a “worst mapping algorithm” to evaluate other algorithms.

NN (Nearest Neighbor) is another algorithm proposed in Ref. [13]. The most significant improvement of the NN algorithm compared with the FF algorithm is that it searches for the next free PE and preferentially searches for the few PEs that are currently closest to the mapped task. Compared with FF, the NN algorithm considers the correlation between tasks and maps tasks with communication requirements as close as possible to the nearest location.

The CoNA(Contiguous Neighborhood Allocation) [14] algorithm makes a relatively large improvement on the mapping algorithm based on the principle of retaining the basic mapping of NN. The core idea of the CoNA algorithm is to make the tasks in the application be mapped in a region close to a rectangle as much as possible by determining the reasonable mapping order and the initial mapping task, so as to make the tasks with communication requirements as close as possible.

The WeNA(Weighted-based Neighborhood Allocation) [15] algorithm considers the traffic size between different tasks as the sorting weight in task sorting. It sorts the order of all tasks related to the current tasks according to traffic size and records the number of all related tasks. Ensures that tasks with large traffic with the current task can be preferentially mapped.

Above mapping algorithms aim at reducing communication latency and increasing system performance. WeNA has made significant achievement on this goal. As the increase in the size of the NoC chip and the increase in integration, the problem of failure has become more serious, and more and more studies have been made to consider the fault tolerance and reliability of NoC in the mapping process. In Ref. [16], the author proposes a software technology based on task migration to detect faults in the system handling core unit fault during operation. The technology also provides support for a number of load information management strategies and migration activation policies, to avoid task migration overhead too much, thus improve the reliability. The technique presented in Ref. [17] analyzes all scenarios in which the processor may generate fault during compilation and stores all processor unit fault information that occurs when task mapping is performed during runtime. It proposes an efficient encoding scheme of the remapping information with respect to the numbers of processors and tasks and has an acceptable storage overhead even if multiple failures occur [18]. analyzes three techniques in adapting intermittent faults, including 1) using spare cores, 2) pausing execution on a temporarily faulty core without notifying the OS, and 3) asking the OS to reconfigure itself not to use the faulty core. In addition [18], proposes a fourth technique: using a thin hardware/firmware virtualization layer to manage an overcommitted system - one where the OS is configured to use more virtual processors than the number of currently available physical cores.

Testing can be seen as a special kind of “fault”, and the tested path cannot be used during testing. Therefore, these mapping algorithms for improving reliability are useful for our research [19]. concluded the methodologies take necessary measures to optimize reliability or cure the faults after they have been detected in the system. In Ref. [20], the authors present a technique that can preserve system reliability while achieving significant energy savings in real-time multiprocessor systems. By giving managed tasks higher priorities, it will get higher performance and lower energy consumption [21]. propose a fault-tolerant application mapping methodology that optimizes system performance and energy consumption while considering the occurrence of different types of faults in the system. It also explored the spare core placement problem for improving systems reliability [22,23]. target temperature aware mapping that leads to increased performance and lifetime. The [22] uses online learning and expert policy to make a balance between performance and thermal profile. The [23] introduce a proactive temperature management method for MPSoCs, it estimates the hot spots and temperature variations in advance and modifies the job allocation to minimize the adverse effects of temperature [24]. proposes a runtime task mapping subsystem that mitigates faults using a wear-based heuristic.

For different optimization purpose, the manager node needs to get the status of tasks on cores as well as send the control information of tasks to the cores. However, this kind of data exchange can be done through the existed NoC. Control and status information can be delivered on the NoC. The amount of control and status information is very small, and the timing requirement is also loose. Thus, the overhead of control and status information is negligible.

Among different test strategies [25,26], periodic built-in self-test (BIST) is usually applied to detect and diagnose the intermittent faults [27]. However, traditional BIST methods significantly influence the NoCs’ throughput because the circuit under test should be disabled for testing and isolated from the rest of the circuit by wrappers. So that the system application will be interrupted, leading to the increase of system running time. TARRA [28] tries to reduce the negative impact of BIST methods on performance by introducing a reconfigurable router architecture combined with a test strategy. However, TARRA does not take into account the idle time during mapping.

The test infrastructure in this paper is derived from Ref. [29] that presents the on-the-field test and configuration infrastructure for a 2D-

mesh NoCs. Unlike the traditional BIST methods [30,31], a controlled BIST strategy is used to diagnose and locate the faults in the components of the path between two crossbars.

The goal of this paper is to integrate the test procedure in application mapping. In this way, it is possible to ensure the reliability of NoC without interrupting the running applications to perform the test. However, if a link is highly utilized during the application mapping, it should be tested with a higher frequency as it is under stress and thus prone to faults. On the other hand, the under-utilized links should not be tested too frequently in order to avoid wasting resources. Therefore, a proper test scheduler and a reasonable mapping strategy should be designed using the information of link utilization. We propose an efficient method to identify the free links and test the path between two crossbars during the task mapping and give priority to the tested links when mapping the next application.

3. Problem definition

An application includes a set of communication tasks which can form an application graph. The mapping algorithm is a process of mapping an application graph (Fig. 4(a)) to a topology graph (Fig. 4(b)) in an optimal way [32]. In order to simplify the comparison and reduce the problem size, we consider a uniform mesh-based NoC in our definitions and experiment.

3.1. Router and link utilization

When performing multi-application on a NoC-based multi-core system, each one of them is typically divided into multiple associated tasks. Then, these associated tasks are assigned to a set of nodes in the NoC according to certain rules, and this process is called mapping. Mapping is a way of resource scheduling and allocation for multi-core systems. After mapping, there will be lots of idle links and some idle routers.

As the example shown in Fig. 1, three applications with three, five and six tasks respectively are mapped onto a 4 × 4 mesh NoC. As you can see from the graph, only two routers in the network are idle (the green real frame in Figure b), and the rest is in the working state (the red real frame in Figure b). On the other hand, the communication links between the routers occupied by different applications A, B and C are completely in idle mode (the green real line in the virtual frame), and some communication links between the routers occupied by the same application are also in idle state (the green real line between the virtual frames). The results fully illustrate that the router centric test encapsulation method is not suitable for the operation of embedded application systems, unless the test process takes up a working router, but this will have a serious impact on the running application.

3.2. Optimize space

We analyze the link utilization under four mainstream mapping algorithms as FF [13], NN [13], CoNA [14], and WeNA [15], shown in Fig. 3. The analysis is performed under two situations where the experiment environment of situation1 and situation2 are adapted from Refs. [14,15], respectively. Link utilization stands for the number of cycles that links are utilized over all simulation run. The link utilization ($L_{utilization}$) can be modeled as:

$$L_{utilization} = \frac{\sum_{t=1}^{T_s} N_c(t)}{N_l \times T_s} \quad (1)$$

where N_c stands for the number of utilized paths at each simulation cycle; N_l represents the number of paths in the network; and T_s is the total simulation cycles. For all four mapping algorithms, the link utilization is less than 10%, verifying the fact that the paths are mostly in an idle mode. Therefore, an opportunistic online test scheduling method can take advantage of such situations to test the paths when free. A

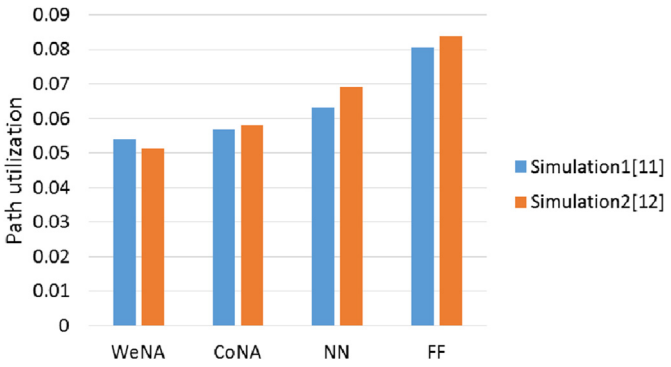

Fig. 3. The link utilization for different mapping algorithms under two different situations in 8 × 8 NoC.

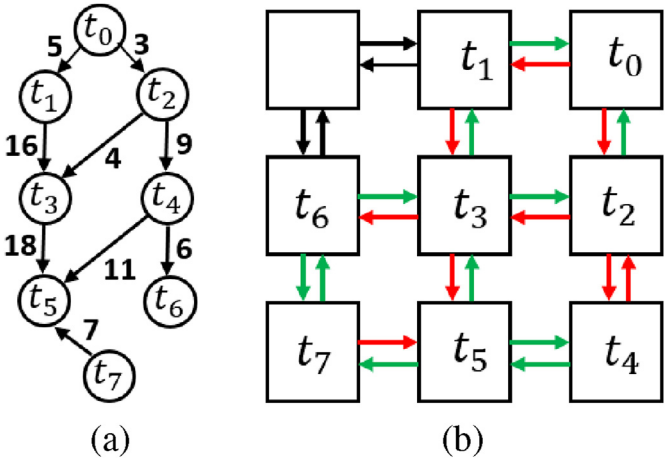

Fig. 4. (a) An application with 8 tasks and 9 edges. (b) free (green) and occupied (red) links. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

reasonable mapping algorithm can provide a balance between test and mapping. It not only improves the link utilization, avoiding the waste of resources but also increases the test frequency.

The integrated system applies the working state of the router and the communication link in the process of mapping. It shows that the communication path and the communication port of its adjacent routers are in a free state for a long time. If these free components are fully utilized, the online barrier detection of these free components is stimulated, and the test frequency is increased, and the test frequency is not interrupted. The intermittent fault on-line detection is completed at the same time of the running application. Based on the above reasons, this paper proposes a new method of BIST test encapsulation. In this method, only a path is adopted, including communication link and router, rather than router or link as the basic unit of testing.

3.3. Testing the paths

The occupied cores with busy and free links can be easily found by considering a uniform mesh-based NoC with the underlying XY routing (as we assumed in this paper). Fig. 4(b) shows a case study that presents the result of mapping an application (Fig. 4(a)) to the cores of NoC using the CoNA mapping algorithm. As can be seen from this figure, the red and green links stand for the occupied and free links, respectively. The free links and the related components can be tested when the application is running without interrupting it or increasing the execution run. The occupied links will be tested as soon as the application exits the

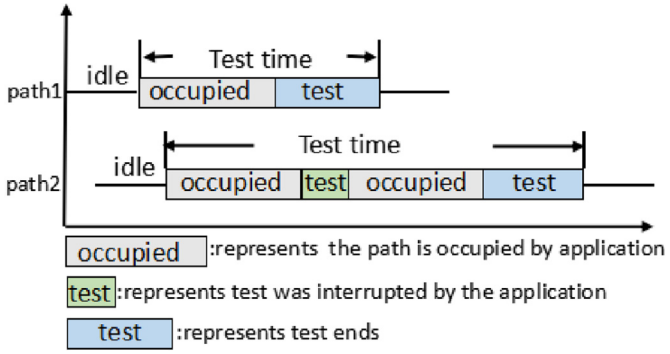

Fig. 5. The path test time under two conditions.

platform. The test lasts until either the test period ends or the paths are needed by a new application. It should be emphasized that the test is immediately stopped when the path is requested by a new application. To reduce these conflicting situations, the ma pping strategy tries to utilize the links that have been recently tested than those of under the test or requiring a test.

3.4. On-the-field test strategy

The test strategy aims at a specific test method to meet the maximum test frequency. Intermittent faults may suffer from the detection delay that is the difference between the time a fault is triggered and the time it is detected. One solution to reduce this delay is to increase the test frequency. On the other hand, the application execution time should not be affected by the test procedure. So, the test scheduling algorithm should be developed by taking into account the three issues:

1. The test start time: one of the test constraints is that it should not affect the application mapping, so once a path is not occupied by any application, the path test starts immediately, like path1 in Fig. 5.
2. Identifying and testing the free paths: we utilize the deterministic routing algorithm XY that enables identifying the free paths as soon as an application is mapped to the platform.
3. Handling the conflicting situation of test and mapping: when an application is entering the platform, the requested paths should be ready, and thus the test should be stopped immediately. In other words, mapping has an absolute priority for the use of paths which ensures that the test does not interrupt or delay the execution of an application, like path2 in Fig. 5.

If a testing link is required by an application, the test procedure is stopped immediately so that the application would not be stopped. In all case, applications have higher priority than test procedures. Test procedure will restart after the application finishes.

3.5. Reliability evaluation metrics

We define a reliability evaluation metric called average test time. The test time refers to the time from when an application task enters the system platform until to the path is tested. Fig. 5 shows the test time of the path in both cases. In the figure, Path1 is occupied when the application enters the system. When the path is idle, it is immediately tested, and the test is completed. For path2, it is immediately tested when it is idle, but in the process of testing, there are new applications entering the system platform. At this time, the test should be immediately interrupted, and the application should be given a place to map. When this path is idle again, the path will be tested again. If there is enough time, this process will be repeated until the path is completed. The maximum test time represents the maximum time required for the path to be tested during the entire application run. It can evaluate the balance of the online test mapping algorithm for the overall scheduling of the online test process. The average test time is the result of averaging the sum of all path times after applying the mapping. A lower average test time means a higher test frequency. By increasing the test frequency, intermittent faults can be better detected, and fault tolerance [28,33] can be properly adopted in a timely manner. Therefore, if a fault can be detected early in the system application run, it can be prevented or fault-tolerated in advance so that the transmission of data packets will not be affected by the fault.

The interrupted test rate ($T_{interrupted}$) can be modeled as:

$$T_{interrupted} = \frac{T_c + T_{ic}}{T_c} \tag{2}$$

where T_c stands for the total number of tested paths during the application mapping overall simulation run. T_{ic} represents the number of tests that are interrupted due to the request on using the path. Since the mapping algorithm proposed in this paper can detect whether the path is tested and the use of the measured path is preferred, this paper tests the interruption rate to evaluate the scheduling performance of the mapping algorithm for the measured path. A lower test interruption rate means that the on-chip system has a higher degree of coordination in the online testing process.

4. Testing method

The data path of a specific port in a router is tightly coupled with part of the control logic, such as the routing calculation unit (RC), virtual channel allocator (VA) and switch allocator (SA), which only work for that port as shown in Fig. 6. Also, the control logic will be suspended while a test task is performing for the data path between two adjacent routers. For more efficient testing, we combine the link and its tightly coupled control logic into a transmission path and divide it into two test groups according to different transmission directions. Each test group contains an input control unit (ICU), an output control unit (OCU), the storage units and the link. The ICU is composed of a finite state machine (FSM), RC, VA and a multiplexer for control signals. Also, the OCU is composed of SA and a multiplexer for data. For decreasing area overhead, the ICU and OCU on the same side share a BIST platform, and they are tested by pseudo-random sequences generated by the BIST platform. For the storage units and the link testing, a predefined test packet is sent from the output register to the input first-input first-output (FIFO) buffer on the other side for functional testing. The test packet is composed of an all-one flit, an all-zero flit and 32 walking-one flits to cover stuck-at and bridging faults.

A typical BIST platform with STUMPS (self-testing using MISR and parallel SRSG) architecture is applied to the ICU and OCU testing as shown in Fig. 7. It uses a linear feedback shift register (LFSR) to generate a number of pseudo-random test sequences using a preset seed. In order to avoid the LFSR being too large, a phase shifter is used to advance the pseudo-random sequence to generate more pseudo-random test vectors. At the capture end of the BIST, the responses including the primary outputs are captured with the at-speed functional mode. Then an X-tolerant test response compactor is leveraged to compress the captured results to reduce the amount of data without severe loss of information. Finally, a distinct signature is generated by a multiple input signature register (MISR) and compared with the desired response in the test response analyzer (TRA) to determine whether there is a fault in the circuit under test. All of the test components mentioned above are controlled by the BIST controller. The BIST controller enables the scan chains and shifter in a given order and controls the timing of the test. The external interface of the BIST platform includes a start signal for triggering, a done signal to indicate if the test is completed, and a pass_fail signal to indicate the test result.

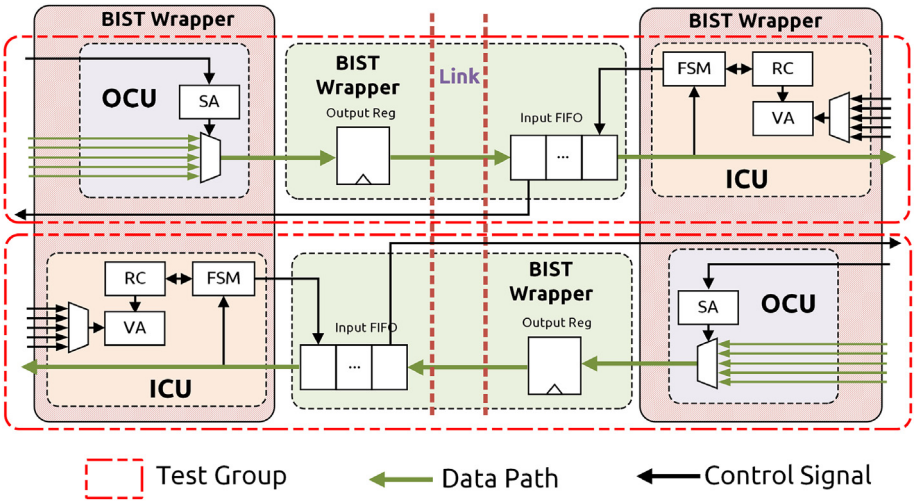

Fig. 6. The path with BIST capacity can be divided into two test groups.

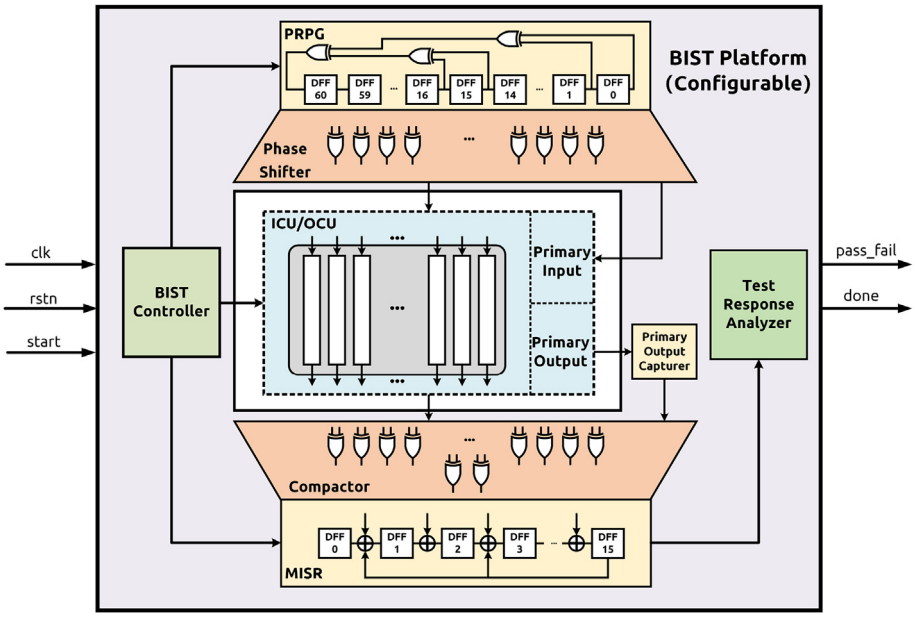

Fig. 7. The BIST platform based on the typical STUMPS architecture.

According to the test results, the physical fault can be located in the control unit or data path of the test group. The location of the faults could be reported to the mapping manager. Mapping manager avoids allocating traffic on the faulty links in subsequence mappings.

5. Testing-aware mapping algorithm

We claim that there is a quest for a testing-aware mapping algorithm which can achieve the goal of application mapping while detecting intermittent faults in NoCs. It will not degrade the overall system performance. In other words, the paths can be tested without interrupting the mapping procedure. To realize the testing-aware mapping technique, we optimize the mapping algorithm while satisfying the requirements of online testing.

The proposed mapping algorithm, named testing-aware mapping algorithm (TAMA) takes the status of online testing to exploit an efficient mechanism for mapping. Based on this method, the tasks are first ordered and then mapped to the platform.

5.1. Sorting tasks

Tasks should be ordered before mapping to the platform. The ordering method of TAMA is similar to WeNA [15]. First, the tasks with the largest number of edges are selected as candidates. For example on the task graph of Fig. 8(a), four tasks have three edges (i.e. t2, t3, t4, and t5). Then, the one with the largest number of communication volume is chosen as the first task to map (i.e., t3).

To sort the remaining tasks, we follow the breadth-first traversal technique similar to [15]. In the example of Fig. 8(b), the breadth-first traversal technique starts from the first task (t3) and explores the first-level neighbor tasks (t1, t2, and t5), and sorts them based on their communication volume (t5 (18), t1 (16), and t2 (4)) from the father task, t3. In the next step, the neighbor tasks of t5 are explored and sorted (t4(11) and t7(7)), as shown in Fig. 8(c). The procedure is repeated for t1 (Fig. 8(d)) and t2, and finally the neighbor task of t4 (i.e. t6) is examined in the last level (Fig. 8(e)). As a result, the tasks' order will be t3, t5, t1, t2, t4, t7, t0, and t6.

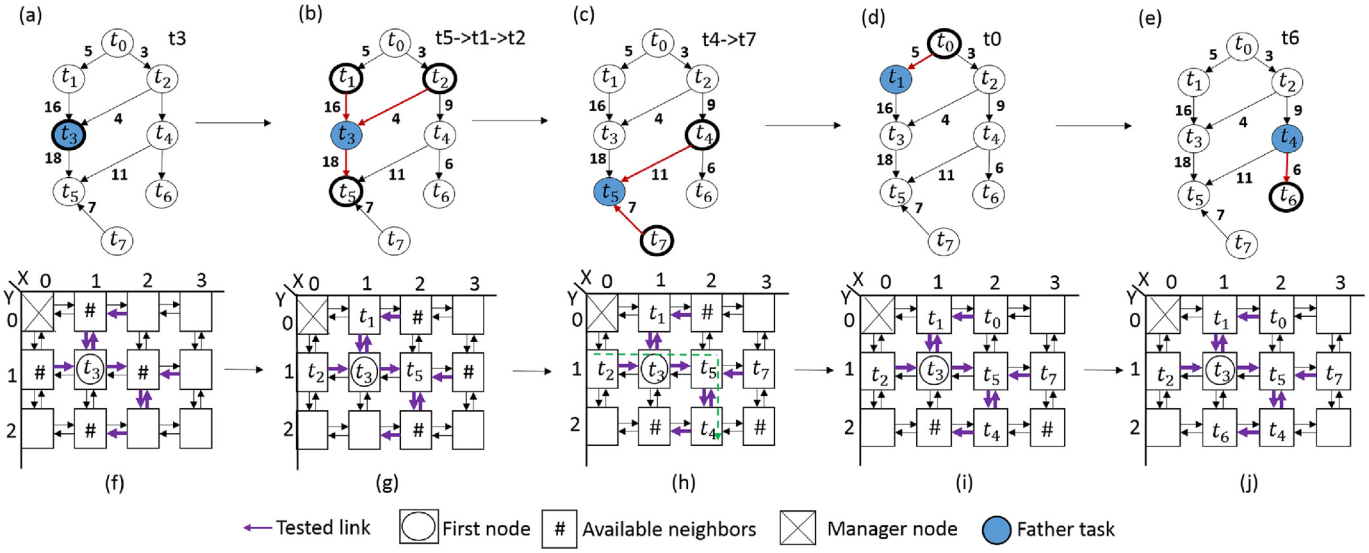

Fig. 8. An example of TAMA mapping algorithm.

5.2. Mapping tasks to nodes

For mapping tasks to the platform, The CoNA and WeNA algorithms start with selecting the first node by taking the situation of the neighboring nodes into account. This prevents the area fragmentation to a large extent while decreasing the congestion probability [14,15]. For the remaining tasks, CoNA randomly maps them to one of the closest neighbors [14]. WeNA, on the hand, takes the communication volume into account [15] in the mapping decision. Similar to other mapping algorithms, TAMA starts by selecting the first node to map and then continue with the rest. The first node selection of TAMA is based on three consecutive steps, given in Algorithm 1:

Step1: the nodes with the maximum number of free neighbors are selected as candidates. This selection prevents area fragmentation and decreases the congestion probability. Fig. 8(f) shows an example of the TAMA mapping algorithm where an application with 8 tasks and 9 edges is going to be mapped to a 3 × 4 mesh NoC with the node $\widetilde{n}_{0,0}$ as the manager. There are two reasons for placing the manager code on the corner. The first is that the contiguity of the mapped region has a significant impact on the performance of the system after mapping. The corner is a good place to keep the contiguity of mapped region. We should not make an obstacle by ourselves. The second is that the traffic between the manager node and task node is not too much so that the communication latency between the manager node and task nodes will not impact the overall system performance. So it is a good choice to place manager node on the node $\widetilde{n}_{0,0}$. We can also see the manager node location in the paper [13–15]we can see that the FF, CoNA and WeNA algorithms all placed manager node on the margin. So it is a good choice to place manager node on the corner. The recently-tested nodes are identified by highlighted arrows. By following Step 1, the maximum number of available neighbors belongs to the nodes $\widetilde{n}_{1,1}$ and $\widetilde{n}_{2,1}$, and thus selected as candidates.

Step2: from the candidates, the ones with the maximum number of tested links are chosen as new candidates. This avoids selecting the non-tested links as much as possible to give them free time to test in the next round. It can also balance the link utilization and improve the reliability by choosing the links that are recently tested. In the given example, since both nodes have four tested links, both of them are chosen again.

Step3: from the new candidates, the node that is closer to the manager ($\widetilde{n}_{0,0}$) is selected as the first node. This is to reduce the system latency by reducing the network latency of data package. If there is more than one candidate, a node is chosen randomly. Since the node $\widetilde{n}_{1,1}$ is closer to the manager, and it is selected as the first node to map. Thereby, the first task, t3, is mapped to the node $\widetilde{n}_{1,1}$.

Algorithm 1 Selecting the first node to map.
Input: A_p: The given application task graph;
Output: *FN*: The selected first node;
1: N ← all nodes in NoC except the manager;
2: **for** node i in N **do**
3: *S*1 ← select the nodes with the maximum number of free neighbors;
4: **end for**
5: **for** node i in *S*1 **do**
6: *S*2 ← select the nodes with the largest number of tested links;
7: **end for**
8: **for** node i in *S*2 **do**
9: *FN* ← select the node with the smallest Manhattan distance to node $n_{(0,0)}$;
10: **end for**

The nodes for mapping the remaining tasks are chosen based on the following three steps, given in Algorithm 2:

Step1: the nodes that are closest to the father task node are selected. As shown in Fig. 8(f), there are four available nodes to map t5 ($\widetilde{n}_{0,1}$, $\widetilde{n}_{1,0}$, $\widetilde{n}_{2,1}$, and $\widetilde{n}_{1,2}$).

Step2: the nodes with the maximum number of tested links are chosen with regard to the location of the father node and the data flow direction. Considering the fact that the flow of data is from t3 to t5, then both $\widetilde{n}_{1,0}$, $\widetilde{n}_{2,1}$ are considered as the nodes with 1 tested link and thus suitable to map the task. In this example, we map t5 to the node $\widetilde{n}_{2,1}$, which is chosen randomly. t1 and t2 are also mapped to $\widetilde{n}_{1,0}$ and $\widetilde{n}_{0,1}$, respectively, as shown in Fig. 8(g). If the father node is located some hops away from the candidate nodes, then the XY routing is considered to find the number of tested links on the path.

Step3: the nodes with the maximum number of tested links are chosen considering the location of other mapped tasks and the data flow direction. For mapping t4 in Fig. 8(g), there are three available nodes close to t5 ($\widetilde{n}_{2,0}$, $\widetilde{n}_{3,1}$, and $\widetilde{n}_{2,2}$); two of which with one tested link. On the other hand, as can be seen from the task graph, t4 has data communication with t2 as well which is already mapped to the platform. So, in this step, the number of tested links on the path from t2 to t4 is counted to decide for a better node to map. The path can be found by considering a static routing like XY. From two possible options ($\widetilde{n}_{3,1}$ and $\widetilde{n}_{2,2}$), the node with the maximum number of tested links (i.e. $\widetilde{n}_{2,2}$) is selected to

map t4 (Fig. 8(h)). Following Step1 to Step3, t7, t0, and t6 are mapped to the platform, shown in Fig. 8(h)–(j).

Algorithm 2 Selecting other nodes to map.
Input: A_p: The given application task graph;
TQ: The ordered task queue;
P: The given platform for mapping;
t and n: The task to map and the selected node;
Output: MAP: T → P: Mapping tasks to the platform;
1: N ← all nodes in NoC except the manager;
2: map $n_{alg1} \leftarrow t_1$;//map t1 to the node selected by Alg. 1
3: **for** i = 2 → |TQ| **do**
4: **for** node j in N **do**
5: $S1$ ← select nodes that are closest to father node;
6: **end for**
7: **for** node j in $S1$ **do**
8: $S2$ ← select the nodes with the largest number of tested links to/from the father node;
9: **end for**
10: **for** node j in $S2$ **do**
11: $S3$ ← select node with the largest number of tested links on the path to/from the other nodes;
12: **end for**
13: map $n_{s3} \leftarrow t_i$
14: **end for**

5.3. Algorithm complexity analysis

Mapping algorithms can be divided into dynamic mapping algorithms and static mapping algorithms. Because static mapping algorithms can run offline, there is no timing limitation. Complex heuristic algorithms can be implemented even though these algorithms need unstable execution time. On the other side, dynamic mapping algorithms run online, which means algorithms must give out the assignment of one task within a time limitation. The calculation time of dynamic mapping algorithms for one task should be stable.

The basic operation in our algorithm is to choose nodes satisfying condition from a bigger set and put them to a smaller set (so-called choose-operation in following), as shown Algs. 1 and 2. At last, our algorithm chooses one node from the nodes meeting all conditions. Thus, there must be a result after Algs. 1 and 2. If the execution time of Algs. 1 and 2 is stable, the proposed algorithm can run online.

Algorithm 1 contains three choose-operations. The worst case is that all nodes meet all conditions and we have to loop them in the last step. In the worse case, all N nodes must be looped in each operation. The total iteration time should be not higher than $3N$. Algorithm 2 contains three choose-operations for each task. Thus, the total iteration time should be not greater than $3|TQ|N$. For each task, the maximum iteration time is also $3N$.

To sum up, the iteration time to give out the assignment of one task is not higher than $3N$, in which N is the number of nodes in SoC. N is a predetermined design attribute. The execution time does not show a relationship with any convergence procedure. Therefore, the proposed algorithm can finish one task within a time limitation. The proposed algorithm can be used as a dynamic mapping algorithm.

Table 1
The setup parameters.

Parameters	Values
Network size	8 × 8
Number of tasks	4–20
Max communication volume	10–30
Application injection rate	10
Simulation length	10,000,000
Number of applications	2000
Test time	1000

6. Results and discussion

6.1. Test time and fault coverage

In order to evaluate the feasibility and effectiveness of the proposed test method, we used Verilog HDL to implement a path with BIST capacity. Synopsys Design Compiler and DFT compiler are used for design for test (DFT) and synthesis with TSMC 45 nm technology. Furthermore, we perform simulation on the synthesized design by Synopsys VCS. Then we dumped out the VCDE file from the simulation and imported it into TetraMAX to evaluate the fault coverage for the stuck-at fault.

Fig. 9. depicts the fault coverages of ICU and OCU with various test time ranged from 8 to 16384 clock cycles. It can be observed that the fault coverage increases rapidly in the left half of the graph. In the range that the test time is greater than 512 clock cycles, the fault coverages of ICU and OCU both reach more than 95% and tends to increase slowly after that.

6.2. Experiment setup

We compare TAMA with four different mapping algorithms as First Free (FF), Nearest Neighbor (NN) [13], Contiguous Neighborhood Allocation (CoNA) [14] and Weighted-based Neighborhood Allocation (WeNA) [15]. The algorithms are implemented in the ESY-net simulator [34]. Several sets of applications are generated using TGFF [35] with

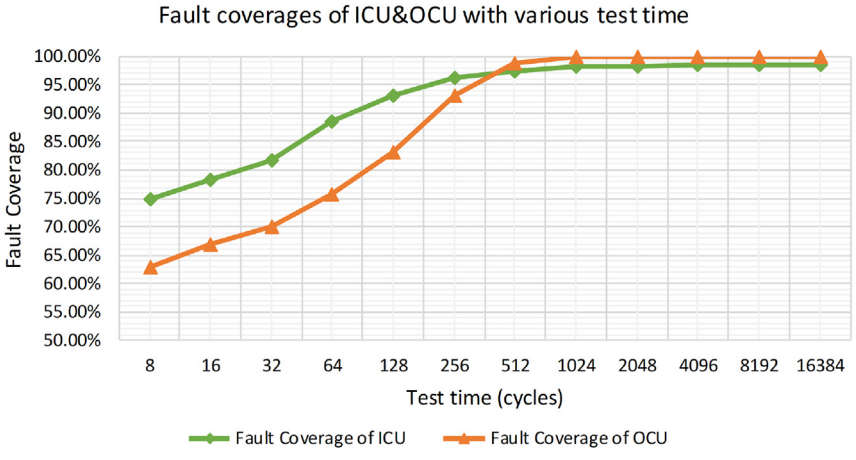

Fig. 9. Stuck-at fault coverage of ICU and OCU for different test time.

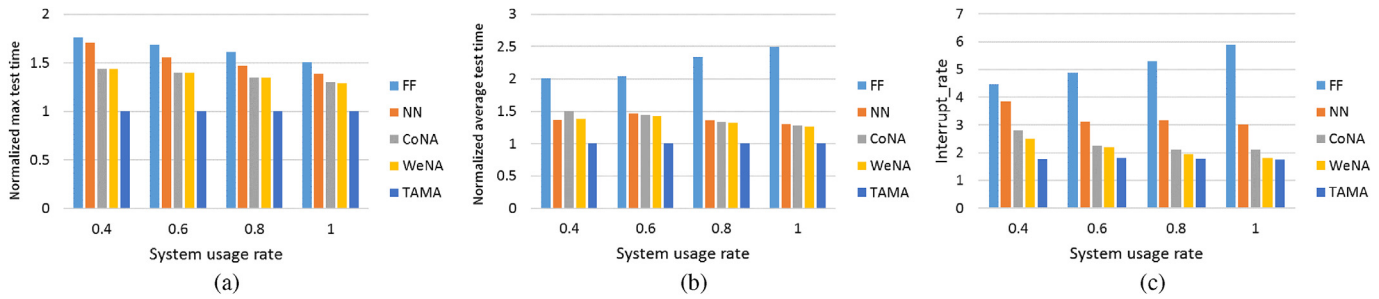

Fig. 10. (a) Max test time (b) Average test time (c) Interrupted test rate.

the parameters listed in Table 1. As for traffic pattern, because we are researching mapping problem, it is more reasonable for us determining traffic by task models. Apart from the parameters listed in this table, we set the system usage rates to 1, 0.8, 0.6, and 0.4. The system usage rate is defined as the percent of cores being occupied by running tasks. By the way, VI.A shows that the test coverage has reached a sufficiently high level for 512 cycles after using BIST. So, in the experiment, the test time is set to be 1000 cycles if considering the time for test preparation, start-up, and exit.

In order to evaluate the algorithm, the examed parameters include maximum test time, average test time, interrupt test rate, average weight Manhattan distance(AWMD), and average latency. Maximum test time, average test time and interrupted test rate are used to evaluate the test frequency and the system reliability. The test time refers to the time from when an application task enters the system platform until to the path is tested. The interrupted rate represents the proportion of times all path tests are interrupted due to mapping requirements throughout the simulation. AWMD and average latency are used to evaluate network communication performance. The average latency represents the average of the sum of the delays of sending packets from the source node to the destination node. (The definition of test timing may not right).

6.3. Maximum test time

Fig. 10(a) shows the maximum test time under different system usage rates for five mapping algorithms. The maximum test time shows the test time in the least ideal case, which can influence the test frequency, thus can increase the system reliability. As can be seen from this figure, TAMA decreases the maximum test time than the WeNA algorithm by more than 28.98%, 34.6%, 39%, 43.8% under system usage rate of 1, 0.8, 0.6 and 0.4, respectively. The benefit of TAMA is more significant in lower usage rates that is because of testing paths at idle times and thus shortening the test time.

6.4. Average test time

From the average test time, we can evaluate the overall performance in increasing system reliability of the algorithm. The average test time is evaluated and compared in Fig. 10(b). As can be seen from this figure, TAMA leads to the lowest average test time which enables a higher test frequency and thus better reliability against intermittent faults. If a faulty path is detected, methods can be applied to either tolerate or fix it, which is out of the scope of this paper. Similar to the analysis of the maximum test time, TAMA is more advantages in lower usage rates with the maximum increase of 38.1% against WeNA when the system usage rate is 0.4.

6.5. Interrupted test rate

The interrupted test rate also shows the performance of the algorithm, lower the interrupted test rate is, less test time it will cost. So that the maximum test time and average test time will decrease more. Fig. 10(c) illustrates the interrupted test rate on all paths during the whole execution time.

As can be seen from this figure, the interrupted test rate is lowest in TAMA as compared to other methods under all configurations. This is due to the fact that TAMA identifies the idle paths based on the underlying XY routing algorithm and the location of the mapped application. Thereby the idle paths are tested meanwhile the application runs in the platform. The remaining paths are tested when the application leaves the platform, and thus the interrupted test rate decreases. It should be mentioned that the mapping strategy selects the tested paths with a higher probability than untested ones. This may leave the busy paths unallocated this time and thus allowing them to be tested during the execution of a new application.

6.6. AWMD metric

Fig. 11(a) shows the results of Average Weighted Manhattan Distance (AWMD) metric under different system usage rates. The AWMD of WeNA is the lowest under four different system usage rate. As for TAMA, the AWMD is a little higher than WeNA, and it is lower than

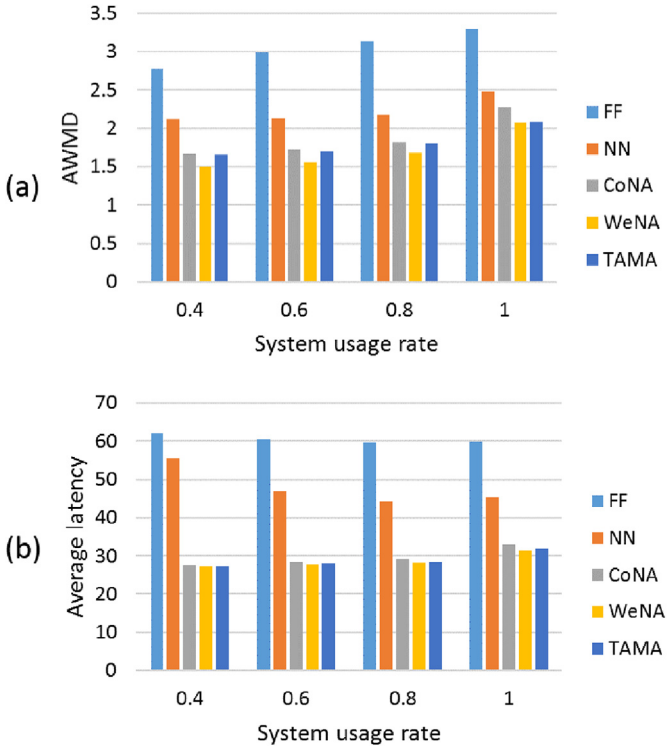

Fig. 11. (a) AWMD metric (b) Average latency evaluation.

other examined methods under system usage rate of 1, 0.8, 0.6 and 0.4. When the system usage rate is 1, and the AWMD of WeNA is by 0.00085 lower than TAMA, nearly the same. Though the AWMD of TAMA is not the lowest, this is to get the advantage in reliability, and the price must be paid.

6.7. Average latency evaluation

The average latency follows the same trend as AWMD. As shown in Fig. 11(b), the average latency of TAMA, CoNA, and WeNA are nearly the same while it is significantly lower than FF and NN under different system usage rates. In sum, TAMA can reduce the maximum test time, average test time, and interrupt rate at no compromise of AWMD and average latency. Which means it can increase the system reliability without sacrificing the Communication performance.

7. Conclusion

In this paper, we proposed a combined approach of mapping algorithm and on-line testing, called testing aware mapping algorithm (TAMA). TAMA targets at increasing test frequency by taking advantage of free time slots during the application execution for testing. First, On-the-field test infrastructure and strategy are proposed, which guarantee the test program cannot affect the process of application mapping by making use of free paths in mapping. Second, tasks are ordered, and a network node which has a maximum number of available neighbors and the largest number of tested paths is selected as the first node to map the first task. It can significantly improve the reliability of the system, decrease the congestion probability and prevent area fragmentation. As the third part, TAMA maps the remaining tasks according to the number of tested paths and nearest neighborhood, trying to form the most tested and contiguous region. Experiment results showed that TAMA leads to significant improvement in test frequency and reliability over traditional mapping algorithms. Although our work is currently focused on regular NoC and deterministic routing, in the future, we will work on irregular NoC or non-deterministic routing with this method.

Acknowledgment

This paper was supported by the National Natural Science Foundation of China under grant (NSFC) No. 61534002, No. 61471407, No.61701095, the Fundamental Research Funds for the Central Universities ZYGX2016J042.

References

[1] L. Benini, G. De Micheli, Networks on chip: a new paradigm for systems on chip design, in: Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, IEEE, 2002, pp. 418–419.
[2] W.H. Wolf, A.A. Jerraya, G. Martin, Multiprocessor system-on-chip (mpsoc) technology, IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 27 (10) (2008) 1701–1713.
[3] W.H. Wolf, Multiprocessor system-on-chip technology, IEEE Signal Process. Mag. 26 (6) (2009) 50–54.
[4] F. Wang, Y. Chen, C. Nicopoulos, X. Wu, Y. Xie, N. Vijaykrishnan, Variation-aware task and communication mapping for mpsoc architecture, IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 30 (2) (2011) 295–307.
[5] E. Carvalho, N.L.V. Calazans, F.G. Moraes, Dynamic task mapping for mpsocs, IEEE Des. Test Comput. 27 (5) (2010) 26–35.
[6] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, L. Peh, Research challenges for on-chip interconnection networks, Int. Symp. Microarchitect. 27 (5) (2007) 96–108.
[7] V. Huard, F. Cacho, X. Federspiel, W. Arfaoui, M. Saliva, D. Angot, Technology Scaling and Reliability: Challenges and Opportunities, 2015.
[8] A.S. Hartman, D.E. Thomas, Lifetime Improvement through Runtime Wear-based Task Mapping, 2012, pp. 13–22.
[9] S. Murali, T. Theocharides, N. Vijaykrishnan, M.J. Irwin, L. Benini, G. De Micheli, Analysis of error recovery schemes for networks on chips, IEEE Des. Test Comput. 22 (5) (2005) 434–442.
[10] C. Constantinescu, Trends and challenges in vlsi circuit reliability, Int. Symp. Microarchitect. 23 (4) (2003) 14–19.
[11] J. Wang, M. Ebrahimi, L. Huang, X. Xie, Q. Li, G. Li, A. Jantsch, Efficient design-for-test approach for networks-on-chip, IEEE Trans. Comput. (2018) 1.
[12] M.R. Kakoee, V. Bertacco, L. Benini, At-speed distributed functional testing to detect logic and delay faults in nocs, IEEE Trans. Comput. 63 (3) (2014) 703–717.
[13] E. Carvalho, N. Calazans, F. Moraes, Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs, in: Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on, IEEE, 2007, pp. 34–40.
[14] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, J. Plosila, Cona: dynamic application mapping for congestion reduction in many-core systems, in: Computer Design (ICCD), 2012 IEEE 30th International Conference on, IEEE, 2012, pp. 364–370.
[15] L.-T. Huang, H. Dong, J.-S. Wang, M. Daneshtalab, G.-J. Li, Wena: deterministic run-time task mapping for performance improvement in many-core embedded systems, IEEE Embed. Syst. Lett. 7 (4) (2015) 93–96.
[16] S. Bertozzi, A. Acquaviva, D. Bertozzi, A. Poggiali, Supporting task migration in multi-processor systems-on-chip: a feasibility study, in: Proceedings of the Design Automation Test in Europe Conference, vol. 1, March 2006, pp. 1–6.
[17] C. Lee, H. Kim, H. Park, S. Kim, H. Oh, S. Ha, A task remapping technique for reliable multi-core embedded systems, in: 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS), Oct 2010, pp. 307–316.
[18] P.M. Wells, K. Chakraborty, G.S. Sohi, Adapting to intermittent faults in future multicore systems, in: 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007), Sept 2007, p. 431.
[19] A.K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on Multi/many-core Systems: Survey of Current and Emerging Trends, 2013, pp. 1–10.
[20] X. Qi, D. Zhu, H. Aydin, Global reliability-aware power management for multiprocessor real-time systems, in: 2010 IEEE 16th International Conference on Embedded and Real-time Computing Systems and Applications, Aug 2010, pp. 183–192.
[21] C. Chou, R. Marculescu, Farm: fault-aware resource management in noc-based multiprocessor platforms, in: 2011 Design, Automation Test in Europe, March 2011, pp. 1–6.
[22] A.K. Coskun, T.S. Rosing, K.C. Gross, Temperature management in multiprocessor socs using online learning, in: 2008 45th ACM/IEEE Design Automation Conference, June 2008, pp. 890–893.
[23] A.K. Coskun, T.S. Rosing, K.C. Gross, Utilizing predictors for efficient thermal management in multiprocessor socs, IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 28 (10) (Oct 2009) 1503–1516.
[24] A.S. Hartman, D.E. Thomas, Lifetime improvement through runtime wear-based task mapping, in: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Ser. CODES + ISSS '12, ACM, New York, NY, USA, 2012, pp. 13–22 [Online]. Available:, http://doi.acm.org/10.1145/2380445.2380455.
[25] X. Chen, Z. Lu, Y. Lei, Y. Wang, S. Chen, Multi-bit transient fault control for noc links using 2d fault coding method, in: 2016 Tenth IEEE/ACM International Symposium on Networks-on-chip (NOCS), Aug 2016, pp. 1–8.
[26] Q. Yu, P. Ampadu, A dual-layer method for transient and permanent error co-management in noc links, IEEE Trans. Circ. Syst. Expr. Brief. 58 (1) (2011) 36–40.
[27] H. Al-Asaad, B.T. Murray, J.P. Hayes, Online bist for embedded systems, IEEE Des. Test Comput. 15 (4) (Oct 1998) 17–24.
[28] L. Huang, J. Wang, M. Ebrahimi, M. Daneshtalab, X. Zhang, G. Li, A. Jantsch, Non-blocking testing for network-on-chip, IEEE Trans. Comput. 65 (3) (2016) 679–692.
[29] Z. Zhang, D. Refauvelet, A. Greiner, M. Benabdenbi, F. Pecheux, On-the-field test and configuration infrastructure for 2-d-mesh nocs in shared-memory many-core architectures, IEEE Trans. Very Large Scale Integr. Syst. 22 (6) (2014) 1364–1376.
[30] S.-Y. Lin, W.-C. Shen, C.-C. Hsu, C.-H. Chao, A.-Y. Wu, Fault-tolerant router with built-in self-test/self-diagnosis and fault-isolation circuits for 2d-mesh based chip multiprocessor systems, in: VLSI Design, Automation and Test, 2009. VLSI-DAT'09. International Symposium on, IEEE, 2009, pp. 72–75.
[31] C. Grecu, P. Pande, A. Ivanov, R. Saleh, Bist for network-on-chip interconnect infrastructures, in: VLSI Test Symposium, 2006. Proceedings. 24th IEEE, IEEE, 2006, p. 6.
[32] P.K. Sahu, S. Chattopadhyay, A survey on application mapping strategies for network-on-chip design, J. Syst. Architect. 59 (1) (2013) 60–76.
[33] M.R. Kakoee, V. Bertacco, L. Benini, At-speed distributed functional testing to detect logic and delay faults in nocs, IEEE Trans. Comput. 63 (3) (2014) 703–717.
[34] J. Wang, Y. Huang, M. Ebrahimi, L. Huang, Q. Li, A. Jantsch, G. Li, Visualnoc: a visualization and evaluation environment for simulation and mapping, in: Proceedings of the Third ACM International Workshop on Many-core Embedded Systems, ACM, 2016, pp. 18–25.
[35] R.P. Dick, D.L. Rhodes, W. Wolf, Tgff: task graphs for free, in: Proceedings of the 6th International Workshop on Hardware/software Codesign, IEEE Computer Society, 1998, pp. 97–101.