Optimizing Dynamic Mapping Techniques for On-Line NoC Test

Shuyan Jiang*, Qiong Wu*, Shuyu Chen*, Junshi Wang*, Masoumeh Ebrahimi†, Letian Huang*, Qiang Li*
*University of Electronic Science and Technology of China, Chengdu, China,
†Royal Institute of Technology (KTH), Sweden
huanglt@uestc.edu.cn

***Abstract*—With the aggressive scaling of submicron technology, intermittent faults are becoming one of the limiting factors in achieving a 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 main goal of traditional mapping algorithms is to allocate applications to the NoC platform, ignoring about 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 links 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 when the application leaves the platform and a new application enters it. The mapping algorithm is tuned to give a higher priority to the tested paths in the next application mapping. This 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, NN, CoNA, and WeNA.**

***Keywords*—Network-on-Chip; mapping algorithm; intermittent fault; on-line testing**

I. Introduction

With the development of fabrication process technology and computing architectures, Multi-Processor Systems-on-Chip (MPSoCs) are likely to have some hundreds of cores integrated into the same chip. However, the performance of a bus-based SoC does not scale with a number of cores. Network-on-Chip (NoC) emerges at a historic moment, which proposes a modular and scalable communication architecture [1], [2].

In such an aggressive trend, reliability has become one of the most important challenges. In fact, the shrunken transistors are prone to the variability phenomena while easily influenced by internal defects, aging process, and wear-out [3]. Furthermore, parameters are more difficult to control when the transistors' size is reduced, which can lead to tiny defects in the process of production [4]. These tiny defects lead to intermittent faults under the combined effect of voltage, temperature, circuit behavior and other factors. The latest research shows that intermittent faults have accounted for more than 40% of the processor's faults, showing the importance of handling them [5].

Mapping algorithms decide the location of application tasks on the NoC-based multi-core system. Different methods are proposed to allocate the tasks onto the cores in an optimal way. Depending on the underlying routing algorithm and the mapping strategy, some links are idle during the execution of an application. Traditional methods usually ignore about these idle times while in this paper we utilize them for the purpose of testing.

Intermittent faults have a distinct characteristic that they occur repeatedly in fixed positions while having a certain randomness on the time of occurrence. One solution to detect intermittent faults is increasing the test frequency. The periodic Built-In Self-Test (BIST) is used to detect and diagnose the intermittent faults [6]. BIST has the advantages of fine-grained diagnosis and high fault coverage. However, increasing the test frequency comes at the cost of interrupting the applications which are not desired. To address this issue, we combine the test and mapping so that the idle times in mapping can be utilized for injecting test vectors.

In this paper, we target the intermittent faults that if not well-treated they may develop to permanent faults. We propose a mapping strategy with embedded testing, called TAMA. The motivation behind this work is that the mapping strategy and the routing algorithm directly affect the link utilization. Thereby, the idle links can be recognized and the components on the idle path can be tested in free times without interrupting the running applications. In short, TAMA tests unused links and components while an application is executing and then tries to map the next application on the tested links/components so that the other idle links can be tested.

The remainder of this paper is organized as follows: Section II 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 III. In Section IV, we propose an optimized testing-aware mapping algorithm. Section V presents and discusses the simulation results. Finally, the conclusion is given in the last section.

II. Related Work

Due to the high pressure to reduce time to market, the high failure rate of modern chips, and the complexity of large multi-core architectures, the testing process of intermittent faults is becoming increasingly important. However, most fault detection methods focus only on permanent or transient faults [7]–[9], ignoring the intermittent faults. On the other hand, increasing the number of processors in a single chip demands an efficient run-time task mapping algorithm. In the case of dynamic mapping, the task assignment and ordering are performed during the execution of an application [10]–[12].

Research has been very limited when considering testing and mapping at the same time. Most researches on mapping algorithms simply focus on improving some essential performance indicators such as traffic and congestion while putting less attention on the system reliability. For example, the CoNA mapping algorithm [10] mainly targets the reduction of internal and external congestion through keeping the mapped region contiguous and placing the communicating tasks in a close neighborhood. The WeNA mapping algorithm [11] primarily aims at decreasing inter-processor communication overhead by arranging the order of task mapping based on the communication volume.

Among different test strategies [7], [9], periodic built-in self-test (BIST) is usually applied to detect and diagnose the intermittent faults [6]. 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. TARRA [13] 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 [14] that presents the on-the-field test and configuration infrastructure for a 2D-mesh NoCs. Unlike the traditional BIST methods [15], [16], 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 identity 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.

III. 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. 1(a)) to a topology graph (Fig. 1(b)) in an optimal way [17]. In order to simplify the comparison and reduce the problem size, we consider a uniform mesh-based NoC in our definitions and experiment.

A. 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. 1(b) shows a case study that presents the result of mapping an application (Fig. 1(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 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 mapping strategy tries to utilize the links that have been recently tested than those of under the test or requiring a test.

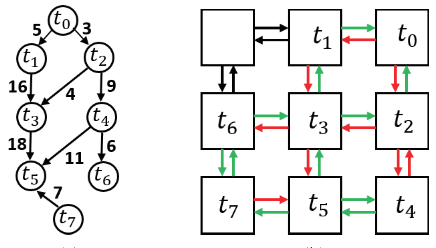


Fig. 1. (a) An application with 8 tasks and 9 edges. (b) free (green) and occupied (red) links.

B. BIST Strucure

Fig. 2 shows the test structure between two routers, which is derived from [14]. Router A generates the test packets in the TPG unit and delivers them toward the TPA unit of Router B for analysis. Test packets pass through the crossbar, output registers, wires and the input buffer controlled by a switch arbiter, routing calculation and state machines. Thus the test procedure is able to capture intermittent faults on data paths as well as the control-units. For simplicity, in this paper we use a *link/path* test that actually means testing all the components between the TPG unit of one router to the TPA unit of the next router.

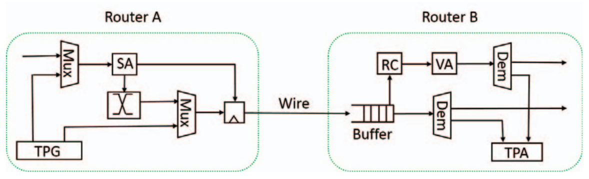


Fig. 2. The BIST structure.

C. 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. 3.
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. 3.

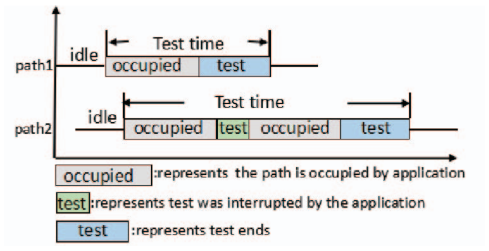


Fig. 3. The path test time under two conditions.

D. Reliability Evaluation Metrics

We define a reliability evaluation metric called average test time. The test time is the time from when an application enters the platform until a path is tested (Fig. 3). The average test time is computed by considering the test time of all paths. Lower average test time means a higher test frequency. We also use average test time to indirectly evaluate system reliability. By increasing the test frequency, intermittent faults can be detected, and appropriate fault-tolerate methods can be adopted on time [18] [13]. Therefore, packets will not be affected by faults if faults are detected and tolerated early.

IV. Testing-Aware Mapping Algorithm

We analyze the link utilization under four mainstream mapping algorithms as FF [12], NN [12], CoNA [10], and WeNA [11], shown in Fig. 4. The analysis is performed under two situations where the experiment environment of situation1 and situation2 are adapted from [10] and [11], 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:

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

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 on-line test scheduling method can take advantage of such situations to test the paths when free. A 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.

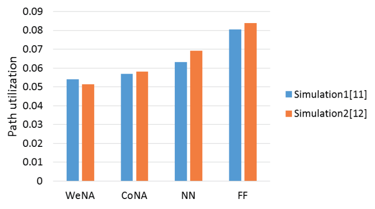


Fig. 4. The link utilization for different mapping algorithms under two different situations in 8×8 NoC.

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 on-line testing to exploit an efficient mechanism for mapping. Based on this method, the tasks are first ordered and then mapped to the platform.

A. Sorting tasks

Tasks should be ordered before mapping to the platform. The ordering method of TAMA is similar to WeNA [11]. First, the tasks with the largest number of edges are selected as candidates. For example on the task graph of Figure 5(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 [11]. In the example of Figure 5(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 Figure 5(c). The procedure is repeated for t1 (Figure (d)) and t2, and finally the neighbor task of t4 (i.e. t6) is examined in the last level (Figure (e)). As a result, the tasks' order will be t3, t5, t1, t2, t4, t7, t0, and t6.

B. Mapping tasks to nodes

For mapping tasks to the platform, The CoNA and WeNA algorithms starts with selecting the first node by taking the situation of the neighboring nodes into account. This prevents the area fragmentation to a large extend while decreasing the congestion probability [10], [11]. For the remaining tasks, CoNA randomly maps them to one of the closest neighbors [10]. WeNA, on the hand, takes the communication

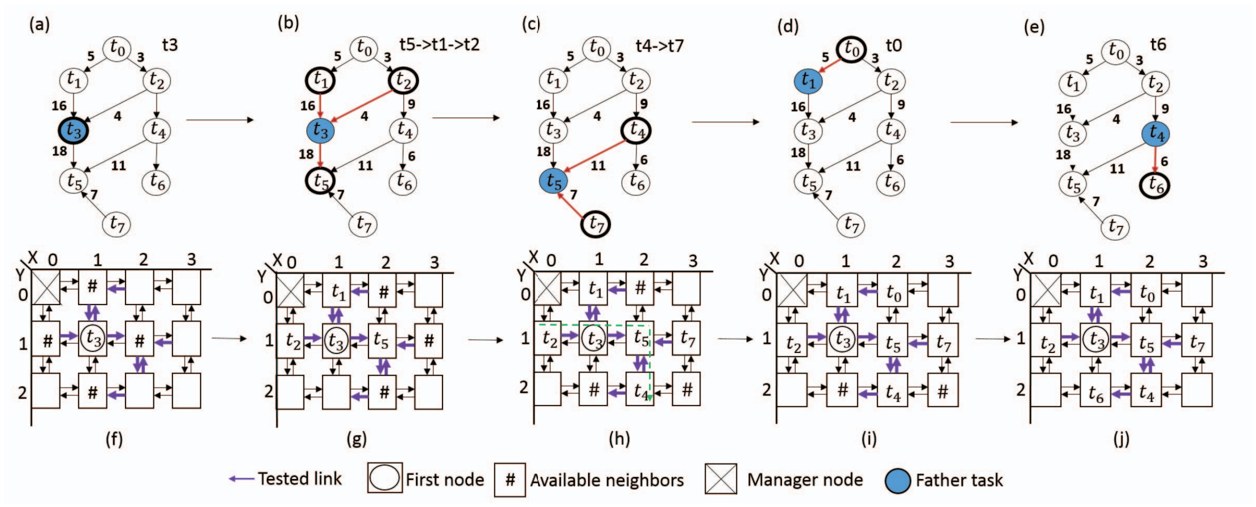


Fig. 5. An example of TAMA mapping algorithm.

volume into account [11] 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 Algorithm1:

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. 5(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 $\tilde{n}_{0,0}$ as the manager. The recently-tested nodes are identified by highlighted arrows. By following Step 1, the maximum number of available neighbors belongs to the nodes $\tilde{n}_{1,1}$ and $\tilde{n}_{2,1}$, 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 ($\tilde{n}_{0,0}$) is selected as the first node. This is to reduce the system latency. If there are more than one candidates, a node is chosen randomly. Since the node $\tilde{n}_{1,1}$ is closer to the manager, it is selected as the first node to map. Thereby, the first task, t3, is mapped to the node $\tilde{n}_{1,1}$.

The nodes for mapping the remaining tasks are chosen based on the following three steps, given in Algorithm2:

Step1: the nodes that are closest to the father task node are selected. As shown in Fig. 5(f), there are four available nodes to map t5 ($\tilde{n}_{0,1}$, $\tilde{n}_{1,0}$, $\tilde{n}_{2,1}$, and $\tilde{n}_{1,2}$).

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 $\tilde{n}_{1,0}$, $\tilde{n}_{2,1}$ are considered as the nodes with 1 tested link and thus suitable to map the

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:     $S1$ ← select the nodes with the maximum number of free neighbors;
4: end for
5: for node i in $S1$ do
6:     $S2$ ← select the nodes with the largest number of tested links;
7: end for
8: for node i in $S2$ do
9:     $FN$ ← select the node with the smallest Manhattan distance to node $n_{(0,0)}$;
10: end for
```

task. In this example we map t5 to the node $\tilde{n}_{2,1}$, which is chosen randomly. t1 and t2 are also mapped to $\tilde{n}_{1,0}$ and $\tilde{n}_{0,1}$, respectively, as shown in Fig. 5(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. 5 (g), there are three available nodes close to t5 ($\tilde{n}_{2,0}$, $\tilde{n}_{3,1}$, and $\tilde{n}_{2,2}$); 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 ($\tilde{n}_{3,1}$ and $\tilde{n}_{2,2}$), the node with the maximum number of tested links (i.e. $\tilde{n}_{2,2}$) is selected to map t4 (Fig. (h)). Following Step1 to Step3, t7, t0, and t6 are mapped to the platform, shown in Fig. (h) to (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 $n_{alg1}$ ← $t_1$; //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 $n_{s3}$ ← $t_i$
14: end for
```

V. Results and Discussion

A. Experiment Setup

We compare TAMA with four different mapping algorithms as First Free (FF), Nearest Neighbor (NN) [12], Contiguous Neighborhood Allocation (CoNA) [10] and Weighted-based Neighborhood Allocation (WeNA) [11]. The algorithms are implemented in the ESY-net simulator [19]. Several sets of applications are generated using TGFF [20] with the parameters listed in Table I. Apart from the parameters listed in this table, we set the system usage rates to 1, 0.8, 0.6, and 0.4.

TABLE I
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	2,000

B. Maximum Test Time

Fig. 6(a) shows the maximum test time under different system usage rates for five mapping algorithms. 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.

C. Average Test Time

The average test time is evaluated and compared in Fig. 6(b). As can be seen from this figure, TAMA leads to the lowest average test time which enables a higher test frequency and thus a 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.

D. Interrupted Test Rate

Fig. 6(c) illustrates the interrupted test rate on all paths during the whole execution time. The interrupted test rate ($T_{interrupted}$) can be modeled as:

$$T_{interrupted} = \frac{T_c + T_{ic}}{T_c} \qquad (2)$$

where T_c stands for the total number of tested paths during the application mapping over all simulation run. T_{ic} represents the number of tests that are interrupted due to the request on using the path. 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.

E. AWMD metric

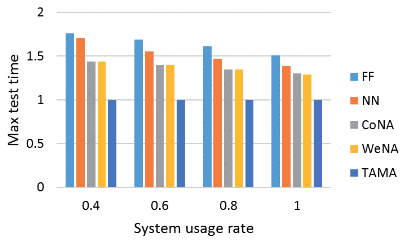
Fig. 7(a) shows the results of Average Weighted Manhattan Distance (AWMD) metric under different system usage rates. The AWMD of TAMA is always lower than all the other examined methods. The only exception is when the system usage rate is 1 and the AWMD of WeNA is by 0.00085 lower than TAMA.

F. Average Latency Evaluation

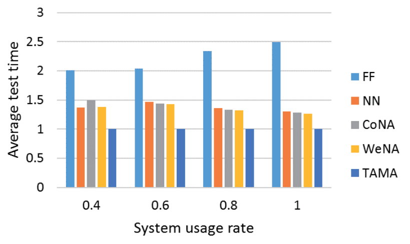
The average latency follows the same trend as AWMD. As shown in Fig. 7(b), the average latency of TAMA, CoNA, and WeNA are nearly the same while it is significantly lower than FF and NN under different system usage rates. In sum, TAMA can reduce the maximum test time, average test time, and interrupt rate at no compromise of AWMD and average latency.

VI. Conclusion

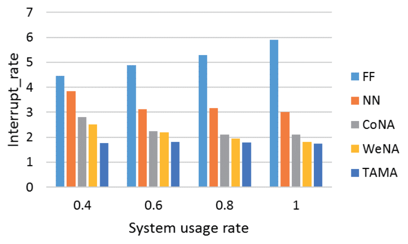
In this paper, we proposed a combined approach of mapping algorithm and on-line testing, called testing aware mapping algorithm (TAMA). TAMA targets at increasing test frequency by taking advantage of free time slots during the application execution for testing. First, On-the-field test infrastructure and strategy are proposed, which guarantee the test program cannot effect the process of application mapping by making use of free paths in mapping. Second, tasks are ordered and



(a)



(b)



(c)

Fig. 6. (a) Max test time (b) Average test time (c) Interrupted test rate.

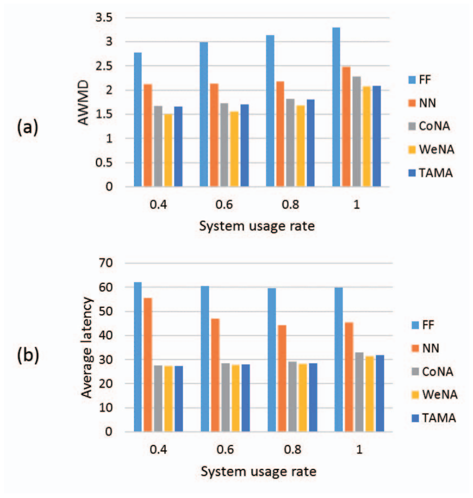


Fig. 7. (a) AWMD metric (b) Average latency evaluation.

a network node which has a maximum number of available neighbors and a largest number of tested paths is selected as the first node to map the first task. It can significantly improve the reliability of system, decrease the congestion probability and prevent area fragmentation. As the third part, TAMA maps the remaining tasks according to the number of tested path and nearest neighborhood, trying to form the most tested and contiguous region. Experiment results showed that TAMA leads to significant improvement on test frequency and reliability over traditional mapping algorithms.

Acknowledgment

This paper was supported by the National Natural Science Foundation of China under grant (NSFC) No.61534002, No.61471407, Central Universities under Grant ZYGX2016J042. This work is also supported by VINNOVA-MarieCurie.

References

[1] L. Benini and 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] M. Palesi and M. Daneshtalab, *Routing Algorithms in Networks-on-Chip*. Springer Publishing Company, Incorporated, 2013.

[3] M. Kaliorakis, M. Psarakis, N. Foutris, and D. Gizopoulos, "Accelerated online error detection in many-core microprocessor architectures," in *VLSI Test Symposium (VTS), 2014 IEEE 32nd*. IEEE, 2014, pp. 1–6.

[4] M. Meijer and J. P. de Gyvez, "Body-bias-driven design strategy for area-and performance-efficient cmos circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 42–51, 2012.

[5] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Characterizing the impact of intermittent hardware faults on programs," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 297–310, 2015.

[6] H. Al-Asaad, B. T. Murray, and J. P. Hayes, "Online bist for embedded systems," *IEEE design & Test of Computers*, vol. 15, no. 4, pp. 17–24, 1998.

[7] Q. Yu and P. Ampadu, "A dual-layer method for transient and permanent error co-management in noc links," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 1, pp. 36–40, 2011.

[8] A. Ghofrani, R. Parikh, S. Shamshiri, A. DeOrio, K.-T. Cheng, and V. Bertacco, "Comprehensive online defect diagnosis in on-chip networks," in *VLSI Test Symposium (VTS), 2012 IEEE 30th*. IEEE, 2012, pp. 44–49.

[9] X. Chen, Z. Lu, Y. Lei, Y. Wang, and S. Chen, "Multi-bit transient fault control for noc links using 2d fault coding method," in *Networks-on-Chip (NOCS), 2016 Tenth IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–8.

[10] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and 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.

[11] L.-T. Huang, H. Dong, J.-S. Wang, M. Daneshtalab, and G.-J. Li, "Wena: Deterministic run-time task mapping for performance improvement in many-core embedded systems," *IEEE Embedded Systems Letters*, vol. 7, no. 4, pp. 93–96, 2015.

[12] E. Carvalho, N. Calazans, and 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.

[13] L. Huang, J. Wang, M. Ebrahimi, M. Daneshtalab, X. Zhang, G. Li, and A. Jantsch, "Non-blocking testing for network-on-chip," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 679–692, 2016.

[14] Z. Zhang, D. Refauvelet, A. Greiner, M. Benabdenbi, and F. Pecheux, "On-the-field test and configuration infrastructure for 2-d-mesh nocs in shared-memory many-core architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 6, pp. 1364–1376, 2014.

[15] S.-Y. Lin, W.-C. Shen, C.-C. Hsu, C.-H. Chao, and 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.

[16] C. Grecu, P. Pande, A. Ivanov, and R. Saleh, "Bist for network-on-chip interconnect infrastructures," in *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*. IEEE, 2006, pp. 6–pp.

[17] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *Journal of Systems Architecture*, vol. 59, no. 1, pp. 60–76, 2013.

[18] M. R. Kakoee, V. Bertacco, and L. Benini, "At-speed distributed functional testing to detect logic and delay faults in nocs," *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 703–717, 2014.

[19] J. Wang, Y. Huang, M. Ebrahimi, L. Huang, Q. Li, A. Jantsch, and 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.

[20] R. P. Dick, D. L. Rhodes, and 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.