A Simple and Effective Evaluation Method for Fault-Tolerant Routing Methods in Network-on-Chips

Yota Kurokawa * and Masaru Fukushi

Graduate School of Sciences and Technology for Innovation, Yamaguchi University, Yamaguchi, 755-8611, Japan;
Email: mfukushi@yamaguchi-u.ac.jp (M.F.)
*Correspondence: ykurokawa@yamaguchi-u.ac.jp (Y.K.)

***Abstract*—This paper proposes a simple and effective evaluation method for fault-tolerant routing methods developed for Network-on-Chip (NoC)-based many-core processors. To cope with faults which significantly degrade the reliability of communication among cores, a variety of fault-tolerant routing methods have been studied. Those methods have been mainly evaluated in terms of communication performance such as latency and throughput by computer simulations of packet routing. However, such evaluations are not practical in that they cannot reveal the performance difference in executing parallel applications with the fault-tolerant routing methods. The proposed method obtains the information of the target parallel application such as task execution time, communication pattern, and communication amount and incorporates it in the conventional packet routing simulations. With the proposed evaluation method, computer simulations have been conducted to evaluate the performance of four famous fault-tolerant routing methods, i.e., Fcube4, Position Route, Passage-Y, and Passage-XY, using NAS Parallel Benchmarks and the performance difference is revealed in executing parallel programs named Integer Sort (IS) and Fast Fourier Transform (FFT). The results show that, Passage-XY outperforms other methods in both IS and FT, and for the case of IS, Passage-XY can reduce the program execution time by up to about 39%, 56%, and 26% compared with Fcube4, Position Route, and Passage-Y, respectively.**

***Keywords*—Network-on-Chip (NoC), fault-tolerant routing, evaluation method, NAS parallel benchmarks**

I. Introduction

In recent years, multi-core and many-core systems have become a mainstream for high-speed processing of wide range of parallel applications, such as multimedia, robotics, machine learning, cloud computing, and edge computing, as well as scientific computing. In these systems, there are two types of methods for connecting cores: a common bus method and a Network-on-Chip (NoC). The common bus, which is the current mainstream, enables data transfer by directly connecting cores to bus lines. However, as the number of cores increases, longer bus lines are required, and transmission delay increases and operating frequency reduces in the systems. In addition, basically, only one-to-one data communication is allowed using bus lines. On the other hand, the NoC, which is a novel connection method for large-scale systems, connects each core to a router configuring a node and communicates each other by sending packets on the network of nodes. Compared to the common bus, the NoC has the advantages of high communication bandwidth, low communication delay, and high scalability for the number of nodes.

In Very Large Scale Integration (VLSI) chips on which the NoC is implemented, the occurrence of faults is inevitable during the system fabrication and run time. Even if only one fault occurs in the network, packets will be dropped or corrupted when passing the faulty node. This significantly degrades the reliability of communication among nodes and eventually results in the malfunction of the whole system. Therefore, fault-tolerant routing methods are essential for avoiding faulty nodes in packet routing.

A variety of fault-tolerant routing methods have been studied for NoCs, e.g., [1–21]. Among various network topologies, a 2D mesh has attracted attention because of ease of implementation. Depending on the approach of designing routing rules, those methods devised for 2D mesh NoCs can be classified into the following five categories: region-based methods [1–3], Virtual Channel (VC)-based methods [4–6], table-based methods [7–10], buffer-less methods [11, 12], and passage-based methods [13, 14]. Despite of the difference of approaches, those methods have been mainly evaluated in terms of communication performance such as latency and throughput by computer simulations of packet routing. Uniform communication patterns are widely employed in the simulations where the source and destination nodes of each packet are decided randomly. Application specific patterns are also employed such as bit-reversal, matrix transpose, hot-spot, and the ones from parallel application benchmarks. However, such evaluations are not practical in that they cannot reveal the performance difference in executing parallel applications with the fault-tolerant

Manuscript received March 31, 2023; revised April 5, 2023; accepted April 19, 2023; published September 6, 2023.

routing methods. Such practical evaluation will require hardware design of not only the developed fault-tolerant routing method but the cores and routers, and also require the development of an operating system to execute parallel applications, i.e., implementation of the prototype entire system [22, 23]. However, as can be easily understood, such prototype implementation is beyond the scope of routing research and thus it is hardly employed in the literature.

Motivated by the need for the practical evaluation, this paper proposes a simple and effective evaluation method for fault-tolerant routing methods developed for NoC-based many-core processors. In contrast to the conventional evaluation methods which at most consider the communication pattern of parallel applications, the proposed method obtains the information of the target parallel application such as task execution time, communication pattern, and communication amount and incorporates it in the conventional packet routing simulations. This method will provide researchers/designers of routing methods with a way of practical evaluation without implementing the prototype system with the routing method.

The remainder of this paper is organized as follows. Section II presents the architecture of an NoC and conventional fault-tolerant routing methods. Section III describes the proposed evaluation method. Section IV evaluates the execution time of parallel programs. Finally, Section V concludes this paper.

II. Architecture of NoC and Fault-Tolerant Routing Methods

A. Architecture of NoC

In NoCs, each core is connected to an on-chip network through a router. The pair of a core and a router is called a node. Various topologies have been studied for the on-chip network, such as 2D mesh/torus, 3D mesh/torus, fat-tree, and hypercubes, as in the basic topology of parallel computers. Among them, 2D mesh is the most popular topology which is suitable for a planar implementation on a VLSI chip.

Fig. 1 shows the architecture of 2D mesh NoC which has nodes of *m* rows and *n* columns. Each node consists of a Central Processing Unit (CPU) core and a router. The CPU core carries out instructions of an assigned computation task, which can be either independent or a part of a parallel program.

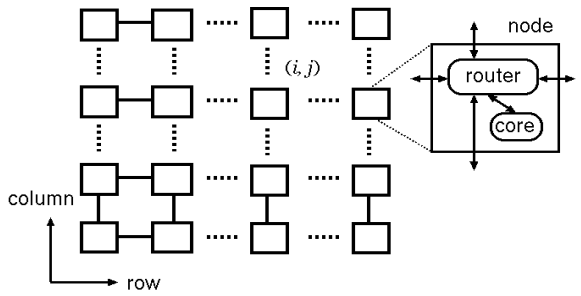

Figure 1. The architeure of 2D mesh NoC.

The configuration of a router used in a 2D mesh NoC is shown in Fig. 2. The router consists of input/output units that store flits (i.e., a small fraction of a packet) for forwarding packets to neighbor routers in the north, south, east, and west; a crossbar switch that connects input/output units; routing circuits that determine output ports; a switch allocator that controls the crossbar switch; and a VC allocator that controls the VC.

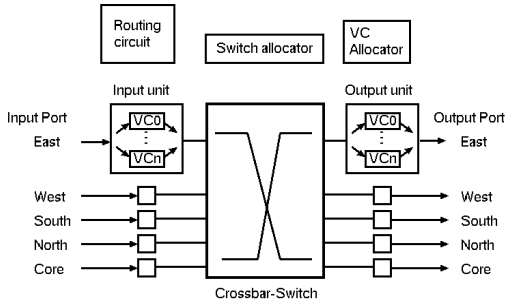

Figure 2. The architecture of router.

B. Packet Routing

Wormhole routing is basically used in NoCs, where each packet is divided into smaller units called flits. Flits are forwarded in a pipeline fashion. There are three types of flits: head flit, which stores destination information, body flit, which divides the data part, and tail flit, which indicates the end of the packet.

When a flit is stored in the buffer of the router's input unit, the following processes are performed [24].

(1) An output port is decided by the routing circuit.

(2) A VC to be used is decided by the VC allocator.

(3) The corresponding input and output units are connected via the crossbar switch with the control of the switch allocator.

(4) The flit in the input unit is sent to the output unit.

(5) The flit is sent to the neighbor node.

These processes in each router are simulated precisely on a cycle basis in conventional simulators.

C. Conventional Fault-Tolerant Routing Methods

In this section, we introduce target fault-tolerant routing methods adopted in performance evaluation. All those methods guarantee packet arrival rate of 100%, thus providing perfect fault-tolerant routing.

1) Fcube4

This study targets the method of Boppana *et al.* [6]. This method creates rectangular faulty regions including faulty nodes and makes packets detour around the faulty regions with VCs.

In this method, detour paths are defined around faulty regions. When a faulty region is created, non-faulty nodes included in the region become unused nodes which are treated similar to faulty nodes. This method defines a clockwise or counterclockwise detour rule for the detour route, depending on the direction of the destination.

Fig. 3 (a) shows a routing example. A source node of a packet is denoted as S and the destination as D. In this example, there are six unused nodes. To select a route closer to the destination, the packet is sent to

counterclockwise around the faulty region and takes the shortest route to the D.

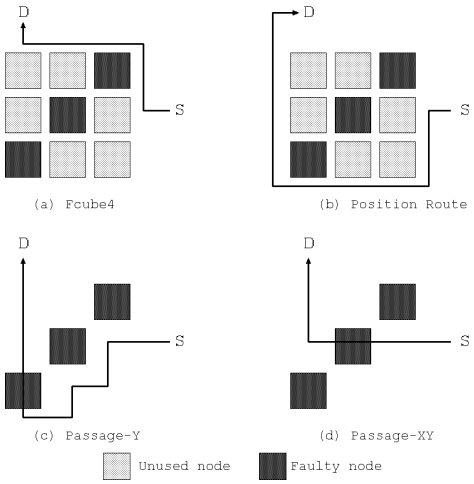

Figure 3. Routing examples of conventional methods.

2) Position route

This method of Fukushima *et al.* [3] creates rectangular faulty regions in the same way as Fcube4. Packets with a message detour around the regions without using VCs.

In this method, detour paths are created. There are three types of detour paths: those for the faulty region on the south edge of the network, those for the fautly region on the west edge, and those for the faulty regions not on the south or west edges. In addition, there are also three types of messages depending on the movement phase: westward, northward or southward, and eastward. This method defines a rule that selects a unique route according to the detour path, the message, and the destination location.

Fig. 3 (b) shows a routing example. In the same way as Fcube4, six unused nodes are generated. When the packet faces the faulty region, it moves clockwise around the region.

3) Passage-Y

Kurokawa *et al.* [13] proposed a method called Passage-Y that detours or passes through faulty nodes without using VCs and creating faulty regions.

The architecture to enable passage of faulty nodes is shown in Fig. 4. It consists of four switches, links, and registers. The possible three states for each switch is shown in Fig. 4. The state of each switch is uniquely decided by a fault flag of the node, and if the node is faulty, packets pass through the faulty nodes vertically and horizontally.

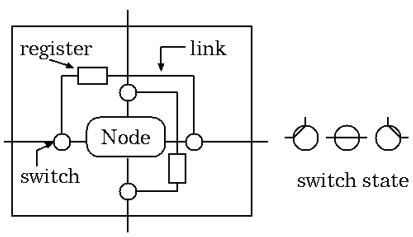

Figure 4. The architecture to allow passage of faulty nodes.

In this method, the neighbors of a faulty node are given information to detour and pass through the faulty node; whether it is on the south edge or not. In this method, when a packet faces a faulty node during the movement of X direction, it detours the north direction if the faulty node is on the south edge. Otherwise, it is detoured in the south direction. If a packet faces a faulty node while moving in the Y direction, it always passes through the faulty node.

Fig. 3 (c) shows a routing example. In this method, no unused nodes are generated. The packet detours the faulty nodes when moving in the X direction and passes through them when moving in the Y direction to proceed to the D.

4) Passage-XY

The method proposed by Kurokawa *et al*. [14], called Passage-XY, is an extension of Passage-Y with two VCs.

In this method, the VC to be used is determined by the location of the D. Similar to Passage-Y, it also chooses whether to detour or pass through faulty nodes based on the information of the faulty node. In addition, this method defines rules that allow passage of faulty nodes in both the X and Y directions.

Fig. 3 (d) shows a routing example. This method also does not generate unused nodes. The packet passes through the faulty node when moving in the X and Y directions, taking the shortest path.

III. Proposed Evaluation Method

A. Basic Approach

As presented in the previous section, a fault-tolerant routing method is employed to realize reliable communication on an on-chip network with faulty nodes. In conventional evaluation methods commonly used in the literature, the developed fault-tolerant routing method is simulated generating packets at random or following a predetermined pattern and compared the communication performance such as latency and throughput with existing methods. Because NoC is a platform for executing parallel applications which generally involve communication among nodes, such conventional evaluations are not practical in that they cannot reveal the performance difference in an practical case where the routing methods are employed.

This paper proposes a simple but effective method for evaluating fault-tolerant routing methods. It takes an approach of utilizing the information of the target parallel application such as task execution time, communication pattern, and communication amount and incorporates it in the conventional packet routing simulations.

First, we explain how to define a model of a parallel program and obtain information from a parallel program. Next, we explain how to reflect the information to the conventional simulations.

B. Getting Information from Parallel Programs

In the proposed method, by acquiring the processing time and communication information of a parallel program *P* and incorporating this information into a conventional routing simulator, it enables to easily

evaluate the target routing methods using an actual parallel program.

Parallel programs are executed with N_p parallelism. First, task processing is performed on a CPU core in an NoC. If a communication process is requested after task processing, the task transfers data to a specific process. In general, parallel programs are executed by repeating these steps. Fig. 5 shows the model of P assumed in this paper. There are N processing blocks (*PB*s) between N-1 communication blocks (*CB*s). To descrie the model, some definitions are given below.

- PB_i: i-th process block (i = 0, 1, …, N)
- CB_i: i-th communication block (i = 0, 1, …, N-1)
- $T_i(PB_j)$: execution time of PB_j in process i
- $CI_i(CB_j)$: communication information CI_i of CB_j in process i, i.e., a pair of a destination process number and communication amount
- $C_i(PB_j)$: the number of cycles for executing PB_j in process i
- $C_i(CI_i(CB_j))$: the number of cycles for performing communication using $CI_i(CB_j)$ in process i
- N_p: the number of processes for executing P

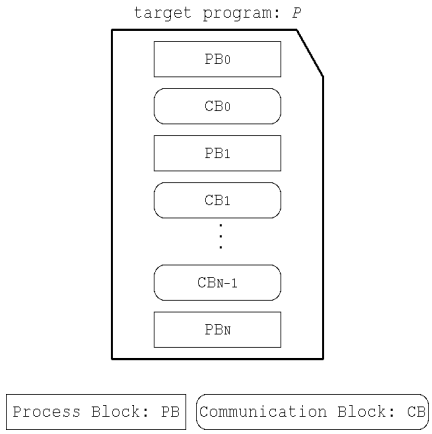

Figure 5. A program mod.

The flow of the proposed method is shown below.

(1) Acquisition of $T_i(PB_j)$

To aquire $T_i(PB_j)$, timer functions are inserted just before and after all *PB*s in the source code of P.

(2) Acquisiton of $CI_i(CB_j)$

To aquire $CI_i(CB_j)$, a function that outputs the destination process number and data size is inserted just before each *CB* in the source code. This function obtains $CI_i(CB_j)$ from the arguments of the communication function in the parallel program. In this paper, MPI, which is commonly used in parallel programs, is assumed.

(3) Execution of P

P is executed with N_p processes to obtain $T_i(PB_j)$ and $CI_i(CB_j)$.

(4) Estimation of $C_i(PB_j)$

To run the simulation with the setting of the assumed NoC, the obtained $T_i(PB_j)$ is converted to $C_i(PB_j)$. Let $T(PB_j) = T_0(PB_j)$ and $C(PB_j) = C_0(PB_j)$ for simplicity. $C(PB_j)$ is calculated by the following equation.

$$C(PB_j) = T(PB_j) \times F_{CPU} \times F_{router} / F_{core}. \quad (1)$$

F_{CPU} is the operating frequency of the CPU in the actual PC that executed the P, and F_{core} and F_{router} are the operating frequencies of the CPU core and router in the assumed NoC, respectively. The $C(PB_j)$ represents the processing time of the assumed CPU core in the NoC, which corresponds to a waiting time in the simulation. Note that one cycle is equivalent to 1 Hz.

(5) Measurement of $C_i(CI_i(CB_j))$

$C_i(CI_i(CB_j))$ is measured by simulating packet routing using $CI_i(CB_j)$. In this simulation, blocking communication is realized such that the next process does not start until it receives a reply packet from the destination node, which is close to the execution on a real computer.

(6) Calculation of the execution time T

Let T_i be the execution time of process i. T_i is given by the following equation.

$$T_i = \Sigma_{0 \le j \le N} C(PB_j) + \Sigma_{0 \le j < N} C_i(CI_i(CB_j)). \quad (2)$$

Then, T is calculated by the following equation.

$$T = \max T_i. \quad (3)$$

From above, the execution time T when P is executed on an assumed NoC can be estimated easily in the conventional simulation.

C. Reflecting Obtained Information in Simulation

This section describes a method to reflect the obtained information in packet routing simulation.

General simulators simulate packet routing generating packets with randomely decided or predetermiend destination nodes in every cycles. Therefore, even if information is acquired by the method in Section III.B, it cannot be directly reflected in the simulator. Therefore, it is necessary to modify a part of the simulator so that it can read the aquired information from a file.

In the proposed method, the execution time of *PB* is also reflected in the simulation by creating a simple task in each process. Each process has the information of $T_i(PB_j)$ and CB_i. Since CB_i contains communication information $CI_i(CB_j)$, the destination and communication amount are stored in each process. If CB_i performs blocking communication, it does not proceed to the next PB_{i+1} until communication of all CB_i has been completed.

Fig. 6 shows an example of how the simulation works. This figure describes the example of running a parallel program P having three *PB*s and two *CB*s in four processes P0, P1, P2 and P3. The simulation is performed using $C(PB_i)$ and $CI_i(CB_j)$ obtained by running P. First, each process is assigned to a node in ascending order as shown in Fig. 6. When the simulation starts, the node to which each process is assigned waits until $C(PB_0)$ cycles have elapsed. Then, it generates packets based on $CI_i(CB_0)$ and routes them until communication of CB_0 is completed. Then, each node waits until $C(PB_1)$ cycles have elapsed. This operation is repeated until PB_2, and the number of cycles to finish all last processes is output as the execution time. Note that the cycle at the end of CB_i is not the same for each process, although each process moves to PB_{i+1} when CB_i communication is completed.

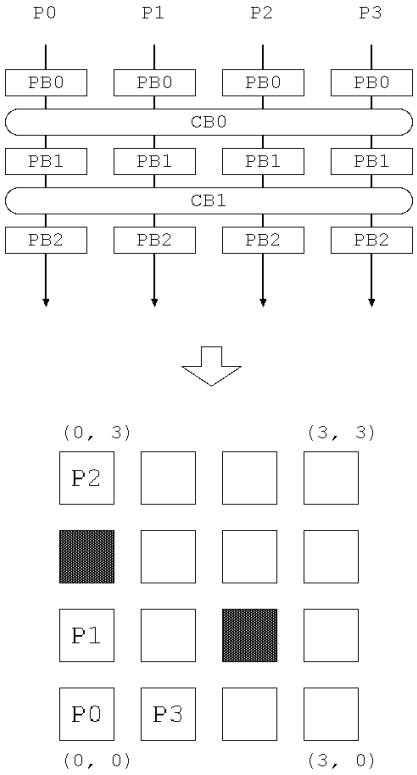

Figure 6. An example of simulation.

IV. Performance Evaluation

To reveal the performance difference of fault-tolerant routing methods with the proposed method, the execution time of NAS Parallel Benchmarks (NPB) is measured for four fault-tolerant routing methods, i.e., Fcube4, Position Route, Passage-Y, and Passage-XY. In this evaluation, two parallel programs are used from NPB: Integer Sort (IS) and Fast Fourier Transform (FFT). Those programs are executed with the setting of class D and 64 parallelism. The environment in which the benchmarks were run is shown in Table I.

TABLE I. Execution Environment

OS	Ubuntu 20.04 (64bit)
CPU	Xeon siliver (2.1GHz)
CPU cores	32
Threads	64
memory	126 GB

A cycle-accurate custom simulator developed in C language was used to simulate packet routing and measure the execution time of the benchmarks for each method. In the simulations, flits are basically transferred to neighbor nodes in the fifth cycle with VCs and the fourth cycle without VCs, as shown in Section II.B. Faulty nodes are randomly generated with a failure rate of f = 2–10%. Other simulation parameters are shown in Table II. Each process was assigned in ascending order by node number, avoiding faulty nodes. The operating frequencies of CPU cores and routers assumed in the simulations are 2 GHz and 200 MHz.

TABLE II. Simulation Parameters

Parameter	Value
Network size	10 × 10
Fault rate (f)	2, 4, 6, 8, 10 %
Input buffer length	8 flits
Output buffer length	1 flit
Packet length	16 flits

Figs. 7–11 show the simulation results for f = 2–10%, respectively. In all cases, execution time of Passage-XY is the shortest, followed by Passage-Y, Fcube4, and Position Route. When f = 2%, Passage-XY reduces execution time for IS by up to about 19%, 42%, and 10% compared with Fcube4, Position Route, and Passage-Y, respectively. For FT, it reduces execution time by up to about 3%, 8%, and 2%, respectively. When f = 10%, it reduces execution time by up to about 39%, 56%, and 26% for IS, and about 7%, 12%, and 3% for FT, respectively. The reduction ratios of IS are larger than those of FT because communication sizes are largely different (IS: several thousand bytes, FT: a few bytes).

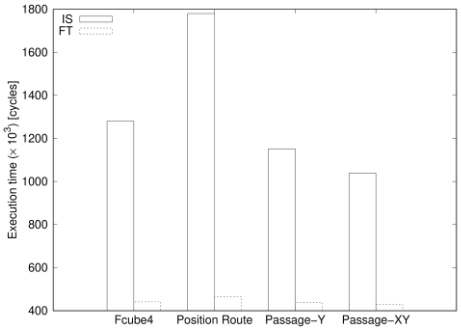

Figure 7. Execution time (f = 2%).

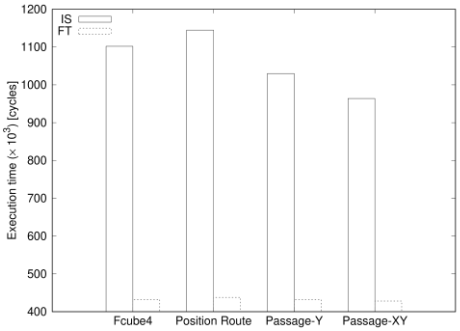

Figure 8. Execution time (f = 4%).

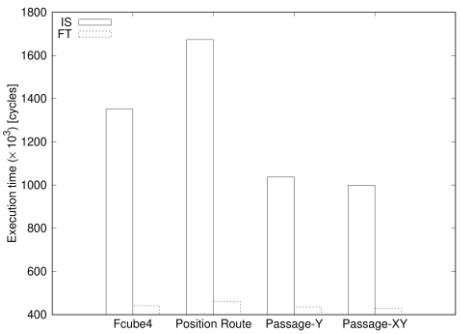

Figure 9. Execution time (f = 6%).

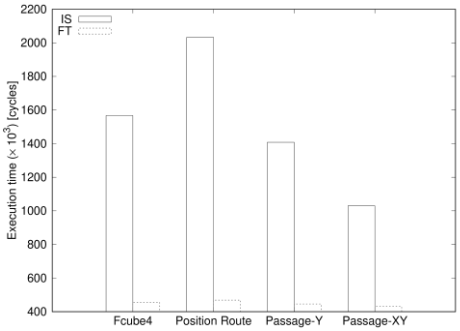

Figure 10. Execution time (f = 8%).

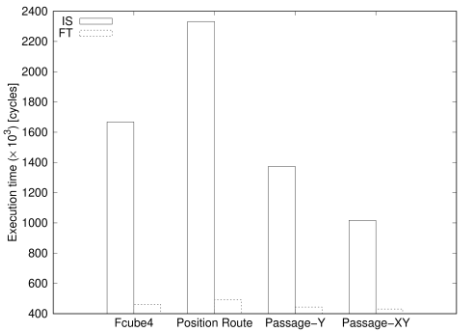

Figure 11. Execution time (f = 10%).

In conventional evaluation methods, communication latency and network throughput are mainly evaluated by changing the packet generation rate. It can only evaluate performance at a specific packet generation rate. However, when an actual parallel program is running on an NoC, the packet generation rate changes dynamically due to various factors, e.g., network congestion, routing method, data size, and program processing time. Fig. 12 shows the packet generation rate for every 10,000 cycles when f = 10% and IS is executed. As shown in this figure, it can be seen that the packet generation rate changes every cycles. In addition, due to the difference in routing methods, the packet generation rate for Passage-XY becomes high and execution time is short, while Position Route has a low packet generation rate and long execution time. Therefore, the proposed method considers the above factors and is a useful method because it can easily evaluate the execution time.

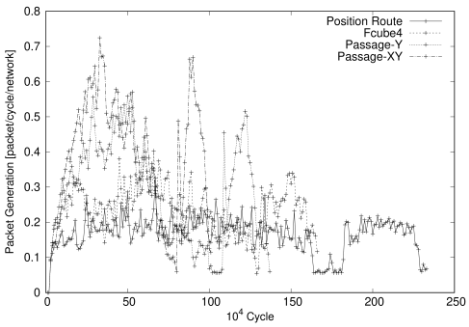

Figure 12. Packet generation rate for IS (f= 10%).

Note that, in the proposed method, the communication time of a parallel program can be obtained with a high accuracy; however, the processing time of the program is converted into cycles from the measured execution time. Hence, the cycle counts may change slightly under an NoC system (CPU, OS, etc.) actually implemented in hardware.

V. Conclusion

This paper proposed a simple and effective evaluation method for fault-tolerant routing methods developed for NoC-based many-core processors. The proposed method obtains the information of a target parallel application such as task execution time, communication pattern, and communication amount and incorporates it in the conventional packet routing simulations. The results show that, for IS, Passage-XY can reduce the program execution time by up to about 39%, 56%, and 26% compared with Fcube4, Position Route, and Passage-Y, respectively.

Conflict of Interest

The authors declare that there is no conflict of interest in this work.

Author Contributions

All authors conceived the idea of the study. Y.K. developed the simulator and measured simulation results. M.F. analyzed the results. Y.K. drafted the original manuscript. M.F. reviewed and revised the manuscript draft. All authors had approved the final version.

Funding

This work was supported by JSPS KAKENHI Grant Number JP21K11810.

References

[1] K. H. Chen and G. M. Chiu, "Fault-tolerant routing algorithm for meshes without using virtual channels," *J. Inf. Sci. Eng.*, vol. 14, pp. 765–783, Feb. 1998.

[2] R. Holsmark and S. Kumar, "Corrections to Chen and Chiu's fault tolerant routing algorithm," *J. Inf. Sci. Eng.*, vol 23, pp. 1649–1662, May 2007.

[3] Y. Fukushima, M. Fukushi, and I. E. Yairi, "A region-based fault-tolerant routing algorithm for 2D irregular mesh network-on-chip," *J. Electron. Test*, vol. 29, no. 3, pp. 415–429, May 2013.

[4] V. Janfaza and E. Baharlouei, "A new fault-tolerant deadlock-free fully adaptive routing in NoC," in *Proc. IEEE East-West Design & Test Symposium (EWDTS)*, Sebia, 2017, pp. 1–6.

[5] D. Sinha, A. Roy, K. V. Kumar *et al*., "Dn-FTR: Fault-tolerant routing algorithm for mesh based network-on-chip," in *Proc. 4th International Conference on Recent Advances in Information Technology (RAIT)*, India, 2018, pp. 1–5.

[6] R. V. Boppana and S. Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," *IEEE Trans. Comput.*, vol. 44, no. 7, pp. 848–864, July 1995.

[7] H. K. Hsin, E. J. Chang, C. A. Lin *et al.*, "Ant colony optimization-based fault-aware routing in mesh-based network-on-chip systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 11, pp. 1693–1705, Nov. 2014.

[8] J. Liu, J. Harkin, Y. Li, and L. P. Maguire, "Fault-tolerant networks-on-chip routing with coarse and fine-grained look-ahead," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 2, pp. 260–273, Feb. 2016.

[9] H. Zhao, N. Bagherzadeh, and J. Wu, "A general fault-tolerant minimal routing for mesh architectures," *IEEE Trans. on Computers*, vol. 66, no. 7, pp. 1240–1246, July 2017.

[10] R. G. Mota, J. Silveira, J. Silveira *et al*., "Efficient routing table minimization for fault-tolerant irregular network-on-chip," in *Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Monaco, 2016, pp. 632–635.
[11] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proc. 36th annual international symposium on Computer architecture*, New York, 2009, pp. 196–207.
[12] Z. Yao, X. Sui, T. Xu *et al*., "QBLESS: A case for QoS-aware bufferless NoCs," in *Proc. IEEE 22nd International Symposium of Quality of Service (IWQoS)*, China, 2014, pp. 93–98.
[13] Y. Kurokawa and M. Fukushi, "Passage of faulty nodes: A novel approach for fault-tolerant routing on NoCs," *IEICE Trans. Fundamentals*, vol. E102-A, no. 12, pp. 1702–1710, Dec. 2019.
[14] Y. Kurokawa and M. Fukushi, "Deterministic/adaptive fault-tolerant routing methods for 2D mesh NoCs based on the passage of faulty nodes," *IEICE Trans. on Information and Systems (Japanese Edition)*, vol. J104-D, no.7, pp. 574–585, July 2021.
[15] Y. Ouyang, Q. Wang, M. Ru *et al.*, "A novel low-latency regional fault-aware fault-tolerant routing algorithm for wireless NoC," *IEEE Access*, vol. 8, pp. 22650–22663, Jan. 2020.
[16] S. Jagadheesh, P. V. Bhanu, J. Soumya *et al.*, "Reinforcement learning based fault-tolerant routing algorithm for mesh based NoC and its FPGA implementation," *IEEE Access*, vol. 10, pp. 44724–44737, April 2022.
[17] J. Samala, H. Takawale, Y. Chokhani *et al*., "Fault-tolerant routing algorithm for mesh based NoC using reinforcement learning," in *Proc. 2020 24th International Symposium on VLSI Design and Test (VDAT)*, India, 2020, pp. 1–6.
[18] Z. Nain, R. Ali, S. Anjum *et al*., "A network adaptive fault-tolerant routing algorithm for demanding latency and throughput applications of network-on-a-chip designs," *Electronics*, vol. 9, no 7, pp. 1–18, 1076, July 2020.
[19] Z. M. Rad and E. Yaghoubi, "ADFT: An adaptive, distributed, fault-tolerant routing algorithm for 3D mesh-based networks-on-chip," *Int. J. Internet Technology and Secured Transactions*, vol. 10, no. 4, pp. 481–490, April 2020.
[20] C. Nehnouh and M. Senouci, "A new fault tolerant routing algorithm for networks on chip," *International Journal of Embedded and Real-Time Communication Systems*, vol. 10, no. 3, pp. 65–85, July 2019.
[21] M. Shah, M. Upadhyay, P. V. Bhanu *et al*., "A novel fault-tolerant routing algorithm for mesh-of-tree based network-on-chips," in *Proc. International Symposium on VLSI Design and Test, Singapore*, 2019, pp. 446–459.
[22] S. R. Vangal, J. Howard, G. Ruhl *et al.*, "An 80-Tile Sub-100-W TeraFLOPS processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.
[23] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel *et al*., "KiloCore: A 32-nm 1000-processor computational array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, April 2017.
[24] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufman Publishers, 2004.

Copyright © 2023 by the authors. This is an open access article distributed under the Creative Commons Attribution License (CC BY-NC-ND 4.0), which permits use, distribution and reproduction in any medium, provided that the article is properly cited, the use is non-commercial and no modifications or adaptations are made.