Communication Driven Remapping of Processing Element (PE) in Fault-tolerant NoC-based MPSoCs

Chia-Ling Chen, Yen-Hao Chen and TingTing Hwang
Department of Computer Science, National Tsing Hua University, Taiwan
s102062567@m102.nthu.edu.tw, yhchen@cs.nthu.edu.tw and tingting@cs.nthu.edu.tw

ABSTRACT

We propose a remapping algorithm to tolerate the failures of Processing Elements (PEs) on Multiprocessor System-on-Chip. A new graph modeling is proposed to precisely define the increase of communication cost among PEs after remapping. Our method can be used not only to repair faults but also to improve the communication cost of given initial mapping results. Experimental results show that under multiple failures, the communication cost by our method is 43.59% less on average compared with that by previous work [1] using the same number of spare PEs. Moreover, the communication cost is further reduced by 4.16% after applying our method to initial mappings produced by NMAP [2].

I. INTRODUCTION

With the advance in semiconductor technology, a large number of Processing Elements (PEs) can be integrated on a single chip. Such implementation is commonly known as Multi-processor System-on-Chips (MPSoCs). Applications, consisting of several tasks, are mapped onto PEs of a chip to perform various functions in high performance. As the demand for high bandwidth and scalability increases, Network-on-Chip (NoC) is considered a feasible communication infrastructure among PEs [3]. The NoC is composed of routers connecting other routers, and each PE is attached to its individual router. With the number of PEs integrated on the MPSoC increasing, reliability has become an important design concern [4]. Fault tolerant schemes are required to maintain the functionality of a system when some of its components break down.

Faults in NoC-based MPSoCs may occur in the communication network (i.e., links, network interfaces and routers) and the PEs. In this paper, we focus on the failures in PEs, and assume that the communication infrastructure can be handled by other techniques. There has been several previous work targeting on solution of tolerating faulty PEs. One solution is to move tasks of application running on a failed PE to other normal ones when a failure is detected. C. Ababei and R. Katti [5] proposed a 2-step heuristic approach to address single and multiple PE failures. This approach searches a new mapping region which is fault-free in first step, and then remaps the tasks on faulty cores onto this new region in second step. It tries to minimize overall migration of each task. O. Derin et al. [6] proposed a fault-tolerant task mapping which is formulated as an Integer Linear Programming (ILP) problem. Optimal mappings for all single-fault scenarios are found at design time, which are used by the online task remapping heuristic with the objective of minimizing communication traffic in the system and total execution time of the application. The above approaches inevitably cause the degradation of the system performance due to extra workload added onto the remaining fault-free PEs.

A more practical solution to tolerate failures is to use hardware redundancy. When an element of system fails, redundant element can take over the role assigned to the failed one and the whole system will run normally. The idea of providing redundant resources to improve reliability is widely applied in various hardware components, including PE, physical links, via, router, etc. In the work of L. Liu, et al. [1], spare PEs are provided to construct a reconfigurable architecture. The topology reconfiguration problem is converted into flow problem in graph theory, and a repair approach is proposed to tolerate PE failures with minimal impact on area, throughput and delay. This method reassigns tasks based on the current locations. Hence, the reconfiguration cost is very low. Moreover, the modeling is very effective and produces results of high repair rate in polynomial time.

In this paper, the topology reconfiguration is adopted to repair PE failures. We propose a remapping algorithm for homogeneous NoC-based MPSoCs with spare PEs. Our algorithm minimizes the total communication cost and achieves 100% repair rate. Moreover, our method can be applied not only to repairing the faults but also improving the performance of initial mapping.

The rest of the paper is organized as follows. Chapter II describes the motivation for this work. Chapter III presents our proposed remapping approach in detail. Chapter IV shows the extension of the algorithm to produce initial mapping. Experimental results are reported in Chapter V, followed by conclusions in Chapter VI.

II. MOTIVATION

Communication cost is an important metric to measure the performance of NoC-based systems. A commonly used metric [2], [7], [8] is given by:

$$commcost = \sum_{i \in C} (amount_i \times hopcount_i) \quad (1)$$

where C is the set of the communication between pairs of tasks, $amount_i$ is the transferred data bytes of communication i, $i \in C$, and $hopcount_i$ is the minimum number of hops between the mapped PE locations of source task and target task of communication i.

L. Liu, et al. [1] has proposed an elegant modeling of repairing failed PE. A non-spare faulty PE at location (x, y) is replaced by a healthy PE at location (x', y'), and then the PE at location (x', y') is replaced by other healthy PE, and so on, until the replacement ends at a spare one. Such a replacement chain is defined as a repairing path. To find a repairing path, a mesh architecture is represented as a topology graph, a directed graph, where each node denotes a PE on the mesh and each directed edge connecting two nodes denotes the replacement of two PEs. To ensure the minimal impacts on communication distance ($hopcount$), a PE can only be replaced by its neighbour PEs. However, the cost of the amount of data transmission is defined on each node rather than edge, which takes the communication cost into consideration indirectly. An example is shown in Figure 1. In Figure 1(a), a 3×4 architecture with application tasks mapped onto its non-spare PEs ($R1$-$R3$, $R5$-$R7$, $R9$-$R11$), and the weight on edge between a pair of tasks denotes the corresponding communication amount. According to the above modeling, it is converted into the topology graph, as illustrated in Figure 1(b). Note that the communication cost is accumulated on each node. Let $R5$ be a faulty PE. A repairing path $R5 \rightarrow R1 \rightarrow R2 \rightarrow R3 \rightarrow R4$ can be seen as a unit flow starting from a source node—faulty PE $R5$ and ending to a target node—spare PE $R4$. The repair method using minimum cost maximum flow (MCMF) algorithm is used to repair multiple faults. The modeling is very effective and produces results of high repair rate in polynomial time.

However, there are two problems in this approach. One is that the communication cost modeled on node by [1] is not able to describe the actual communication cost and thus unsatisfactory results may be produced. Figure 2(a) shows a 3×4 architecture with one faulty PE to be repaired. Figure 2(b) shows that a minimum cost repairing path based on the cost defined by [1] is obtained. After task remapping, as shown in Figure 2(c), the total communication cost becomes $commcost = (6\times2+30\times2+10\times1+5\times1+5\times2+24\times1+30\times1+5\times2) = 161$. But choosing another repairing path, as shown in Figure 2(d), less amount of communication cost after remapping can be obtained as shown in Figure

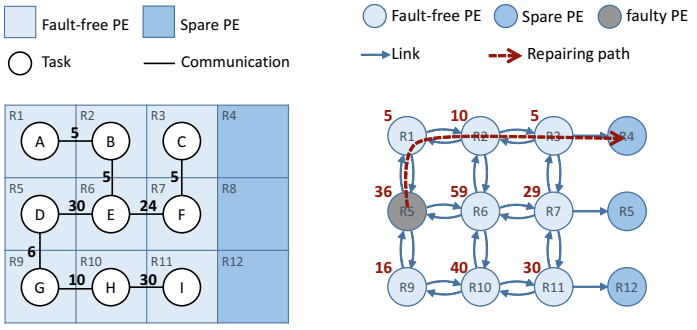

Fig. 1. (a)Task mapping on 3x4 architecture (b)Topology graph modeled by [1].

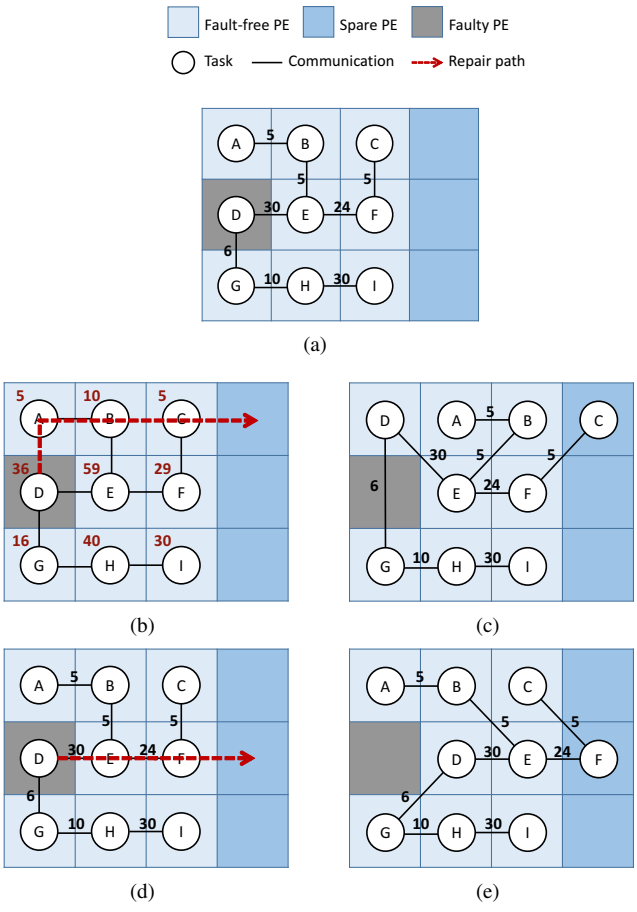

Fig. 2. (a)Initial mapping with 1 faulty PE. (b)The minimum cost repairing path obtained by [1]. (c)Remapping result from [1], $commcost = 161$. (d)Another repairing path. (e)Remapping result with lower communication cost, $commcost = 131$.

2(e), where $commcost = 131$. Clearly, the defined cost in [1] is not precise enough to represent the communication cost.

The second problem is that previous work [1] does not guarantee to fully repair a faulty NoC-based chip even when there are enough spare PEs. The reason is that each PE can be replaced by its neighbour PE only, which means a repairing path should be a continuous replacing chain. Figure 3 shows examples of 3×4 and 4×5 architecture with non-repaired faulty PE and un-used spare PE. In Figure 3(a), the faulty PEs are $R1, R2$ and $R5$. $R2$ and $R5$ can be repaired by the replacing chains $R2 \rightarrow R3 \rightarrow R4$ and $R5 \rightarrow R6 \rightarrow R7 \rightarrow R8$ respectively. Yet $R1$ cannot be repaired since all the neighbour PEs of $R1$ are faulty. Similar situation occurs in another example shown in Figure 3(b) with faulty PEs $R6, R11, R12$ and $R17$. In this case, $R11$ is blocked by other 3 faulty PEs and cannot find out a continuous replacing chain to a spare PE, $R5$.

Motivated by the above two observations, we propose a communication driven remapping algorithm to repair faulty PEs. We consider

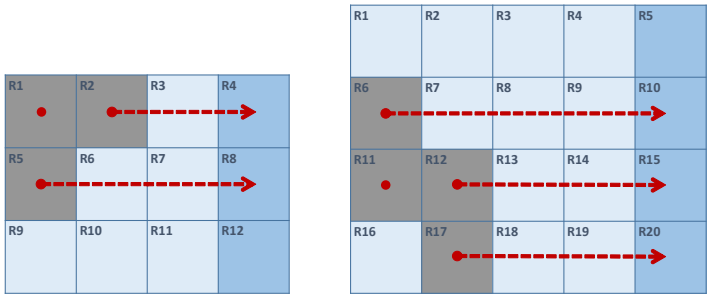

Fig. 3. Examples of unfully repaired system.

the communication demands between pairs PEs directly to minimize the total communication cost. Moreover, we allow each PE can be replaced by PE hops away from it, and guarantee 100% repair rate when the number of faulty PEs are not more than that of spare ones. Lastly, in this paper we assume faults occur one by one rather than simultaneously. Our assumption is more reasonable than previous work [1], since the probability of PEs simultaneously breaking down is extremely low.

III. Repairing Algorithm by Remapping

Our objective is to tolerate the PE failures by remapping while minimizing the communication cost. We firstly present the problem formulation of the remapping problem in Section III-A. Then, we describe the proposed repairing algorithm in detail in Section III-B.

A. A New Graph Modeling

We follow the idea of repairing path presented in previous work [1]. A repairing path can be seen as a flow starting from a faulty PE and ending at a spare PE, and the problem of obtaining the optimal repairing path can be converted into a minimum cost flow (MCF) problem.

As mentioned in Chapter II, the communication cost can be expressed as $commcost$ in Equation (1). For precisely modeling the impact on $commcost$, we will introduce a new topology graph. Our core idea is to model the variation of $commcost$ for all possible task re-allocation scenarios in the graph. The new topology graph is formally defined as following.

The topology graph is a directed graph, $G = (V, E, Cost)$. Each vertex $v_i \in V$ represents a PE in the NoC-based MPSoC, and each directed edge $v_i \rightarrow v_j$ denoted as $e_{i,j} \in E$, represents the PE replacement of v_i by v_j, which means the task on v_i is remapped onto v_j. The cost of each edge $e_{i,j}$, denoted as $cost_{i,j} \in Cost$, represents the **increment** in $commcost$, $\Delta commcost$, when the task on v_i is remapped onto v_j assuming unchanged locations of other tasks. Figure 4(a) shows an example of a 2×3 architecture with initial mapping. If we remap task B from $R2$ onto $R3$, the $commcost$ will increase 15, i.e., $(10\times2+5\times3)-(10\times1+5\times2) = 15$. Hence, $cost_{R2,R3}$ on edge $R2 \rightarrow R3$ is 15 in the topology graph as shown in Figure 4(b). Similarly, the $commcost$ will decrease 10 if we remap task C from $R4$ onto $R2$. Hence, $cost_{R4,R5}$ is -10. A repairing path in the topology graph is composed of a set of continuous directed edges, the total cost of these edges is exactly the $\Delta commcost$, which is the $commcost$ after remapping compared to the $commcost$ before mapping. Although our cost definition of each edge is modeled in the local view assuming unchanged locations of other tasks, global optimal solution can be obtained by accumulating cost of edges in repairing path. We will explain it later.

In order to solve the non-fully repairing problem of previous work [1] mentioned in Chapter II, we allow a PE can be replaced by non-neighbour PE. That is, an edge $e_{i,j}$ is constructed between v_i and v_j even when v_i and v_j are not adjacent to each other. However, in order to model cost on edges accurately, not all pairs of vertices $\in V$ have edges between them. Consider an example shown in Figure 5. Figure 5(a) shows a 2×3 architecture. $R1, R2, R4$ and $R5$ are non-spare PEs, while $R3$ and $R6$ are spare ones. One fault occurs in $R1$. The $commcost$ of initial mapping is 50. The complete topology graph is constructed as shown in Figure

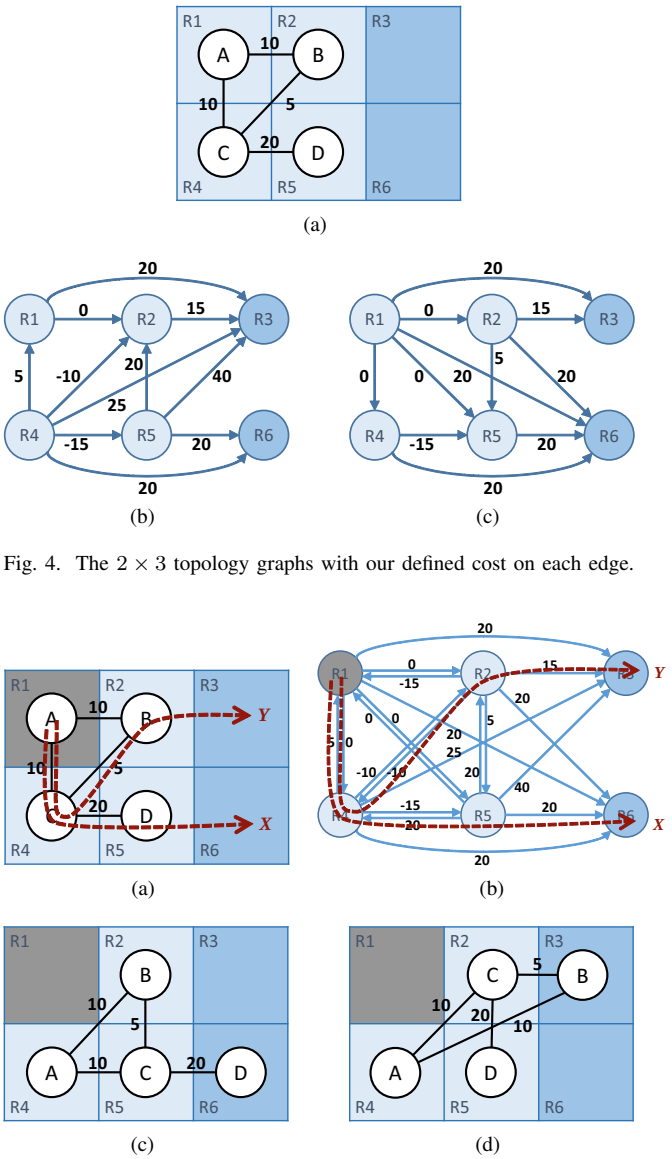

Fig. 4. The 2×3 topology graphs with our defined cost on each edge.

Fig. 5. (a)Initial mapping. (b)Complete topology graph. (c)Remapping result based on repairing path $R1 \rightarrow R4 \rightarrow R5 \rightarrow R6$. (d)Remapping result based on repairing path $R1 \rightarrow R4 \rightarrow R2 \rightarrow R3$.

5(b), where each non-spare node has directed edges going to the other nodes including spare ones. Consider the repairing path X, $R1 \rightarrow R4 \rightarrow R5 \rightarrow R6$, with total cost, $0 + (-15) + 20 = 5$. The remapping result based on the repairing path X is shown in Figure 5(c). The *commcost* becomes 55 from 50 after remapping. $\Delta commcost = 5$ represents the exact cost of repairing path X. However, consider another repairing path Y, $R1 \rightarrow R4 \rightarrow R2 \rightarrow R3$, with total cost, $0 + (-10) + 15 = 5$ in the graph shown in Figure 5(b). The remapping result based on the repairing path Y is shown in Figure 5(d), where *commcost* becomes 75 from 50. Hence the cost of path Y is $\Delta commcost = 25$, rather than the incorrect value $\Delta commcost = 5$. This inconsistent costs between repairing path and actual remapping result is caused by the down-and-up (back-and-forth) moves as shown in Figure 5(a). Consider the communication link where data amount is 10 between task A and task B in Figure 5(a). When only remapping task A from $R1$ to $R4$, the distance between these two tasks become zero hop and hence the communication cost between them decreases 10. When only remapping task C from $R4$ to $R2$, the distance between task A and task B is still one hop, and hence the communication cost between them increases 0. However, the increment of communication cost is not $(-10) + 0$ if we perform both actions, remapping the task A from $R1$ to $R4$ and task C from $R4$ to $R2$, at the same time. The distance between A and C becomes two hops and the communication cost increases 10 actually. When a repairing path is not moving monotonically increasing (decreasing) in x or y-direction, incorrect cost is modeled.

The following Lemma 1 gives conditions to build the topology graph.

Lemma 1. *Let $R_0 \rightarrow R_1 \rightarrow ... \rightarrow R_n$ be a given repairing path and R_k be the k^{th} PE of the path located at (x_k, y_k), where k is a positive integer. To obtain the correct cost, the given repairing path should subject to the following conditions:*

$$\begin{cases} x_0 \leq x_1 \leq ... \leq x_n \quad or \quad x_0 \geq x_1 \geq ... \geq x_n \\ y_0 \leq y_1 \leq ... \leq y_n \quad or \quad y_0 \geq y_1 \geq ... \geq y_n \end{cases}$$

Proof. The communication between pairs of tasks can be classified into three cases based on whether the tasks are on the PEs of repairing path. Three cases are that both tasks are not on the PEs of the path, only one of them is on the PEs of the path, and both are on the PEs of the path. Since we locally define the cost on topology graph assuming that locations of other tasks are unchanged, clearly our cost modeling is correct for communication of the first and second cases.

Let us consider the communication of third case. Assume there is a communication link between task A and task B with communication volume $amount_{A,B}$. Task A is on R_i located at (x_i, y_i) and task B is on R_j located at (x_j, y_j), where $i < j$ and $i, j \in 0, 1, ..., n$. After remapping based on the given repairing path, task A is remapped onto R_{i+1} located at (x_{i+1}, y_{i+1}) and task B is remapped onto R_{j+1} located at (x_{j+1}, y_{j+1}). The number of hops between task A and task B, represented as $hopcount_{A,B}$, is changed to $hopcount'_{A,B}$ after remapping. Hence, communication cost is also changed according to Equation (1). We obtain

$$\begin{aligned} \Delta commcost_{A,B} &= amount_{A,B} \times (hopcount'_{A,B} - hopcount_{A,B}) \\ &= amount_{A,B} \times [(|x_{j+1} - x_{i+1}| + |y_{j+1} - y_{i+1}|) - (|x_j - x_i| + |y_j - y_i|)] \\ &= amount_{A,B} \times [\alpha(x_{j+1} - x_{i+1}) + \beta(y_{j+1} - y_{i+1}) - \alpha(x_j - x_i) - \beta(y_j - y_i)] \\ &= amount_{A,B} \times [\alpha(x_i - x_{i+1} - x_j + x_{j+1}) + \beta(y_i - y_{i+1} - y_j + y_{j+1})] \end{aligned} \tag{2}$$

where

$$\alpha = \begin{cases} 1, & \text{if } x_0 \leq x_1 \leq ... \leq x_n \\ -1, & \text{if } x_0 \geq x_1 \geq ... \geq x_n \end{cases}$$

$$\beta = \begin{cases} 1, & \text{if } y_0 \leq y_1 \leq ... \leq y_n \\ -1, & \text{if } y_0 \geq y_1 \geq ... \geq y_n \end{cases}$$

In our modeling, $\Delta commcost_{A,B}$ on edge $R_i \rightarrow R_{i+1}$ is modeled as $\Delta commcost_{A,B}(R_i \rightarrow R_{i+1}) = amount_{A,B} \times [\alpha(x_i - x_{i+1}) + \beta(y_i - y_{i+1})]$, and $\Delta commcost_{A,B}$ on edge $R_j \rightarrow R_{j+1}$ is modeled as $\Delta commcost_{A,B}(R_j \rightarrow R_{j+1}) = amount_{A,B} \times [\alpha(x_{j+1} - x_j) + \beta(y_{j+1} - y_j)]$. Hence, total $\Delta commcost_{A,B}$ on the given repairing path is

$$\begin{aligned} \Delta commcost_{A,B} &= amount_{A,B} \times [\alpha(x_i - x_{i+1}) + \beta(y_i - y_{i+1})] \\ &\quad + amount_{A,B} \times [\alpha(x_{j+1} - x_j) + \beta(y_{j+1} - y_j)] \\ &= amount_{A,B} \times [\alpha(x_i - x_{i+1} - x_j + x_{j+1}) + \beta(y_i - y_{i+1} - y_j + y_{j+1})] \end{aligned} \tag{3}$$

As we can see, the results of Equation (3.1) and (3.2) are equal, which clearly indicate that our modeling can obtain the solution with correct total communication cost. □

Following Lemma 1, we limit topology graph by removing all the edges that cause down-and-up (back-and-forth) moves. In this paper, the spare PEs are assumed to place at the right side of the architecture. Therefore we firstly construct all the edges going to right in x-direction. Next, up edges and down edges can not coexist in the same topology graph. But if the edges are limited to going only up or down in the y-direction, the solution space will significantly be reduced. Therefore, we construct two topology graphs. They are the right-up one and the right-down one, which include all the possible repairing paths without down-and-up (back-and-forth) moves. An example of the 2×3 architecture shown in Figure 4(a) is modeled to two topology graphs illustrated in Figure 4(b) and (c), where

Figure 4(b) shows the 2×3 right-up topology graph, and Figure 4(c) shows the 2×3 right-down one.

B. *Repairing Algorithm*

The flow of our proposed repairing algorithm is illustrated in Figure 6. Given an initial mapping. When a new faulty PE occurs, we start repairing procedure. First, the corresponding topology graphs are constructed. Then MCF is applied to find a repairing path. Finally, tasks are remapped following the repairing path. We describe each step in detail as follows.

Step 1: Construct the topology graphs

Once a PE failure is detected, the right-up topology graph and the right-down topology graph are constructed. Meanwhile, the cost of each edge is defined according to the communication among PEs, as mentioned in Section III-A.

Step 2: Apply MCF to obtain the repairing path

For each topology graph, define two additional nodes S and T, where S is the source node and T is the target node. Define an edge from S pointing to the faulty PE. For each spare PE, define an edge from it to T. The cost of these edges are defined as 0. Then, apply the minimum-cost-flow algorithm on these two topology graphs respectively. Clearly, we will obtain two flows, one is from the right-up topology graph and the other is from right-down topology graph. Simply choose the one with lower cost as the repairing path.

Step 3: Remap the tasks

Remap the involved tasks based on the repairing path produced by step 2. A new mapping result is produced. Note that if a new faulty PE occurs, the procedure is repeated. New topology graphs are constructed based on the remapping result produced in the last iteration and failed nodes in the previous iteration are removed from topology graphs.

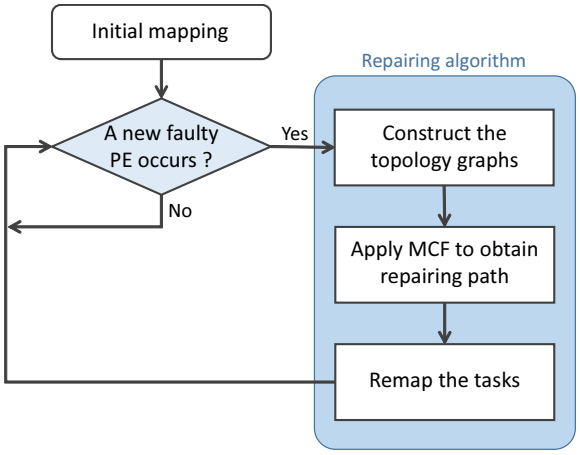

Fig. 6. Flow diagram of the proposed repairing algorithm.

Figure 7 shows an example of proposed repairing algorithm. A fault occurs in PE, $R1$, of the 2×3 architecture, as shown in Figure 7(a). The right-up and right-down topology graphs are constructed and then MCF algorithm is applied. A repairing path $R1 \rightarrow R2 \rightarrow R3$ is obtained from the right-up topology graph illustrated in Figure 7(b) and the total cost is 15, while a repairing path $R1 \rightarrow R4 \rightarrow R5 \rightarrow R6$ is obtained from the right-down topology graph illustrated in Figure 7(c) and the total cost is 5. Therefore, we select the repairing path in right-down topology graph. Finally, the tasks are remapped by the repairing path as shown in Figure 7(d) and $\Delta commcost = 5$. That is, the communication cost of remapping result increases 5 compared to that of initial mapping.

IV. Extension to Initial Mapping

Many task-mapping methods have been proposed [9]. Among them, balance of work load and communication cost are commonly used as mapping metrics. Our remapping technique is to move the complete task assigned to one PE to the other one. Hence, it affects communication cost rather than load balance.

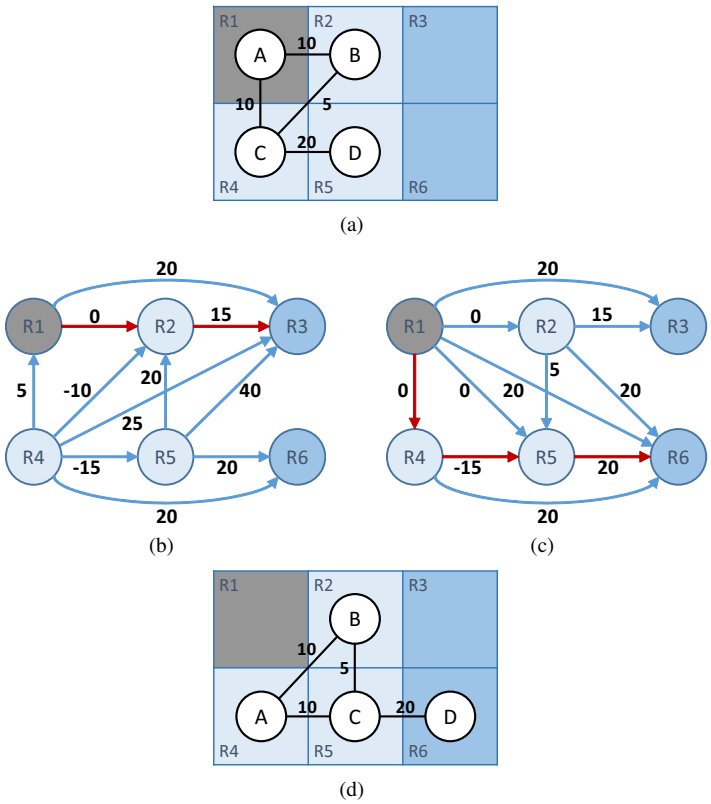

Fig. 7. (a)Initial mapping. (b)A min-cost repairing path obtained from the right-up topology graph. (c)A min-cost repairing path obtained from the right-down topology graph. (d)Remapping result.

Besides repairing faulty PEs, our method can also be applied to further improve the communication cost of a given initial mapping. A two-step algorithm is proposed to perform a re-mapping:

- Step 1: Placement of faulty PEs
- Step 2: Application of our repairing algorithm

To utilize our method to further improve the communication cost of a given initial mapping. The first step is to assume the locations of faulty PEs and spare PEs. Then, the second step is to call our algorithm to repair the faulty PEs by spare PEs.

An example is illustrated in Figure 8. Figure 8(a) is a given initial mapping with 9 tasks mapped onto a 3×3 NoC. First, we assume PEs on the left column are faulty and create a column of spare PEs on the right side, as illustrated in Figure 8(b). Next, the repairing algorithm mentioned in Section 3.2 is applied, and the remapping result is obtained as illustrated in Figure 8(c). Except the column of faulty PEs, we take the relative position of tasks mapped on remaining 3×3 region as the new mapping result. The new initial mapping is shown in 8(d).

Since we assume faults occur one by one in our approach, the specified locations of faults in Figure 8(b) will give 3! remapping results due to different sequence of fault occurrence. Besides faulty PEs on the left side, faulty PEs can also be located on the other three sides, i.e., right, top and bottom sides. Figure 9(a), (b), (c) and (d) shows 3×3 NoC with faults on the right, bottom, top and left respectively, and their spare PEs are added on the opposite side of the faulty PEs. Therefore, at least ($n!\times 4$) mapping results can be obtained for an $n \times n$ architecture. In our experiment, n is less than 10 and hence we could try all possible sequences and select the result with the lowest *commcost* as our new initial mapping result.

V. Experimental Results

In this chapter, we presents the benchmarks, environment setting and experimental results. Our proposed approach is implemented using C and verified on a MPSoC evaluation tool called Transaction Generator (TG) [10]. Eight application models are used in the experiment as shown in Table I. av_bench is from literature [11], while others: RS_enc, RS_dec, H264-720p_dec, H264-1080p_dec, Fpppp, FFT-1024_complex and Sparse

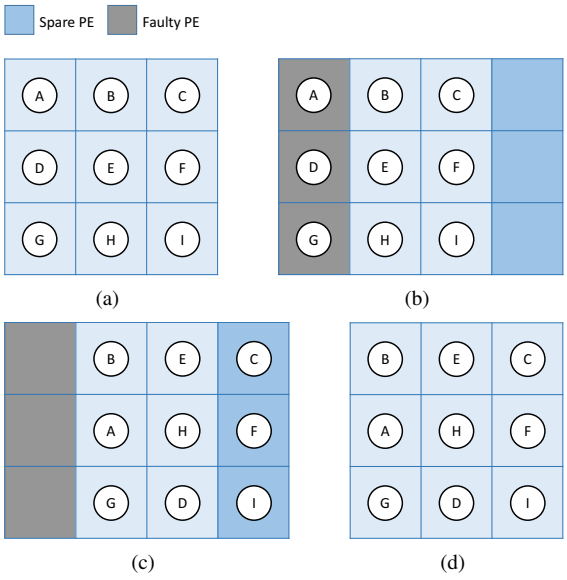

Fig. 8. Extension to initial mapping.

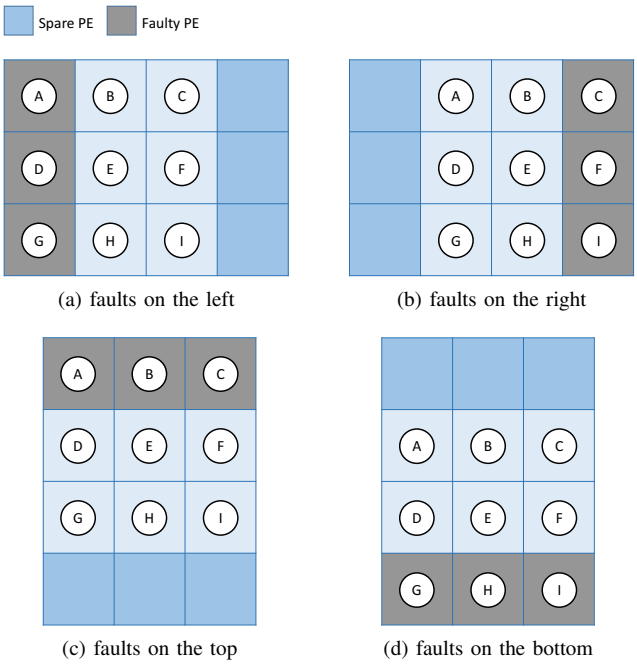

Fig. 9. PE failures are assumed on 4 different sides.

are from the realistic NoC traffic benchmark suite MCSL [12] based on real applications. The mapping of application tasks is on a mesh 4×5 architecture, where 4 PEs on the right most column are spare PEs, and XY deterministic routing is used. Hence, tool in MCSL partitions tasks to 16 groups and each group is mapped to one non-spare PE. As shown in Table I, column labelled $\#CommunicationLinks\ among\ tasks$ is the number of communication links among tasks in the original spec. After mapping, we have column labelled $\#CommunicationLinks\ among\ PEs$ to represent the total number of communication links among PEs which are related to the $commcost$. In the following experiments, we firstly perform the result of repairing PE failures, and then followed by the effect on initial mapping. Finally, the overall impact of applying our approach to both initial mapping and repairing is presented in the last experiment.

A. *Results on Repairing*

In repairing experiments, we present the evaluation results of repair rate and communication cost. Our repairing algorithm, called $Ours$ for short, is compared with Liu's work [1] called $Liu's$. Repair rate is the probability that faulty PEs can be repaired by spare ones. To conduct experiment on repair rate, we randomly generate 50000 faulty patterns for the number of faults varying from one to four on 4×5 and 8×9 mesh NoC respectively. Liu's and our repairing algorithm are then called to repair faulty PEs. The results of repair rates are shown in Figure 10. It shows that as the number of faulty PEs increases, repair rate of $Liu's$ has fallen rapidly in both 4×5 and 8×9 NoC architectures. In contrast, $Ours$ remains 100% repair rate for all cases. For experiment of communication cost, initial mapping is produced by NMAP. 10000 faulty patterns are randomly generated. Different applications have different characteristics, including communication amount and number of communication links among tasks. For fair comparison of different applications, we use average $hopcount$ each data byte traverses as the metric of communication cost instead of the $commcost$. Figure 11 shows the average communication cost of all applications with different number of faulty PEs. Let us take number of faulty PEs = 0 as the base. When number of faulty PEs = 1, communication cost of $Liu's$ raises 0.119 $hopcount/byte$ averagely while that of $Ours$ raises only 0.064 $hopcount/byte$ averagely. That is, $Ours$ is improved by $(0.1199 - 0.0643)/0.1199 = 0.463 = 46.3\%$. We compute the improvement for all cases in different number of faulty PEs and then compute the average. The communication cost of $Ours$ achieves 43.59% less on average than that of $Liu's$.

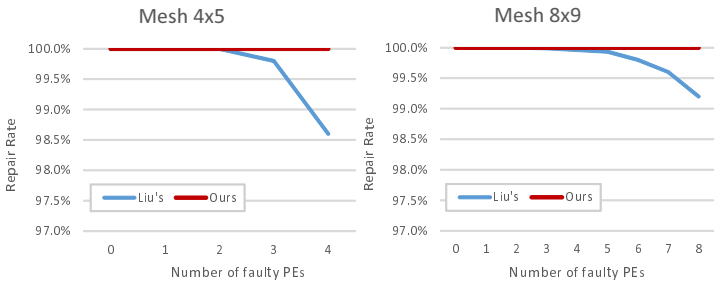

Fig. 10. Comparison of repair rate of Liu's [1] versus Ours under different number of faults.

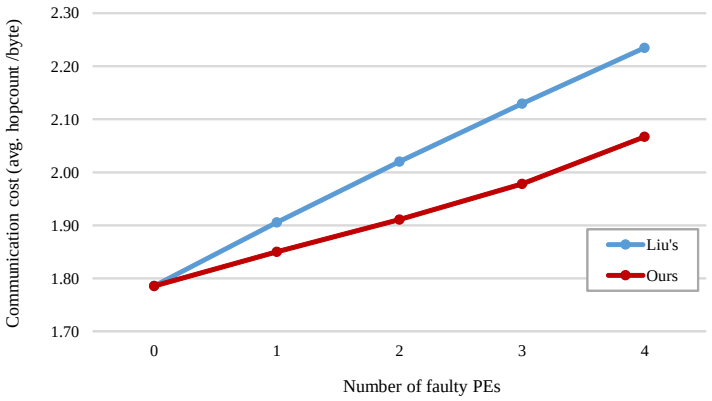

Fig. 11. Comparison of communication cost of $Liu's$ versus $Ours$ under different number of faults.

B. *Results on Initial Mapping*

As mentioned in Chapter IV, our method can be applied to improve initial mappings. We choose two initial mapping results, from Ori and NMAP respectively, as our baseline mappings. Ori is the default mapping of application models provided by TG, while NMAP [2] is a mapping algorithm that minimizes the average communication delay under the bandwidth constraint. With or without applying our proposed remapping algorithm to Ori and NMAP results, we have four combinations as shown in Table II. Figure 12 shows the comparison of communication cost of 4 initial mapping results for each application. Except FFT-1024_complex, either results of Ori or NMAP are improved after applying our remapping algorithm. Average improvement of Ori+Ours over Ori is 10.91% and that of NMAP+Ours over NMAP is 4.16%.

TABLE I
BENCHMARKS.

Application	Description	# Tasks	# CommunicationLinks	
			among tasks	among PEs
av_bench	a video codec pair and an audio codec	40	57	25
RS_enc	Reed-Solomon code encoder	262	348	18
RS_dec	Reed-Solomon code decoder	182	392	71
H264-720p_dec	H.264 video decoder with a resolution of 720p	2311	3461	65
H264-1080p_dec	H.264 video decoder with a resolution of 1080p	5191	7781	65
Fpppp	Chemical program performing multi-electron integral derivatives	334	1145	120
FFT-1024_complex	Fast Fourier transform with 1024 inputs of complex numbers	16384	25600	116
Sparse	Random sparse matrix solver	96	67	34

TABLE II
COMPARISON OF 4 COMBINATIONS FOR INITIAL MAPPING.

Mapping Algorithm	Description
Ori	default mapping that benchmarks provide
Ori + Ours	applying our approach on default mapping
NMAP	a mapping technique proposed in [2]
NMAP + Ours	applying our approach on NMAP result

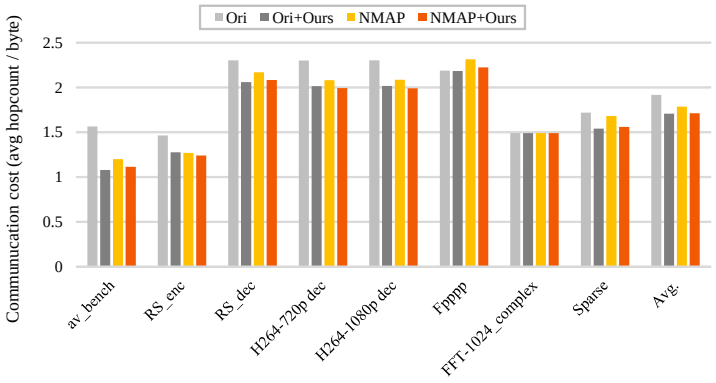

Fig. 12. Comparison of communication cost of 4 methods for initial mapping.

C. *Overall Effect*

Finally, we conduct the experiment to see the overall effect by adopting our approach to both initial mapping and repairing. Three combinations of algorithms for initial mapping and repairing are: Nmap–Liu, NmapOurs–Liu and NmapOurs–Ours. In Nmap–Liu, the initial mapping is produced by NMAP and the repairing by Liu's [1]. In NmapOurs–Liu, the initial mapping is produced by NMAP+Ours and the repairing by Liu's [1]. In NmapOurs–Ours, the initial mapping is produced by NMAP+Ours and the repairing by Ours. The experimental result is shown in Figure 13. By comparing the Nmap–Liu and NmapOurs–Liu, we can observe that communication cost by NmapOurs–Liu decreases an average of 3.94% as compared with that by Nmap–Liu due to applying our method to initial mapping. When comparing the NmapOurs–Liu and NmapOurs–Ours, we can see the increment of communication cost caused by increasing faults is smaller in NmapOurs–Ours than in NmapOurs–Liu. The result of NmapOurs–Ours achieves 39.08% improvement on average compared with the result of NmapOurs–Liu.

VI. CONCLUSIONS

In this paper, we have presented a repairing algorithm to tolerate PE failures by remapping the application tasks. Our proposed graph modeling can precisely describe the the increment of communication cost resulting from task remapping. Our mapping result is 43.59% better than previous work [1]. As long as the number of faulty PEs are not more than that of spare ones, we can achieve 100% repair rate for all cases. Moreover, the proposed algorithm can also be applied to the initial mappings to improve results. We have performed a set of experiments to demonstrate that our proposed algorithm can indeed obtain better results in terms of communication cost.

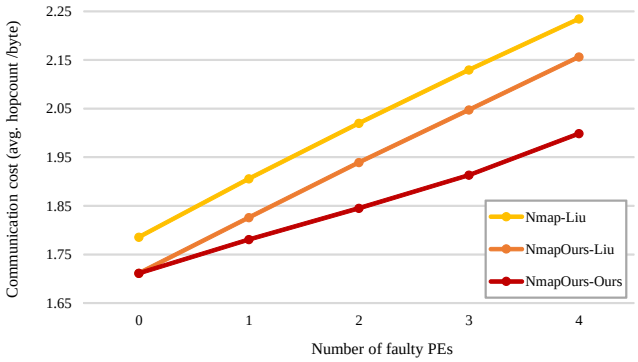

Fig. 13. Comparison of overall effect.

REFERENCES

[1] L. Liu, Y. Ren, C. Deng, S. Yin, S. Wei, and J. Han, "A novel approach using a minimum cost maximum flow algorithm for fault-tolerant topology reconfiguration in noc architectures," in *The 20th Asia and South Pacific Design Automation Conference, ASP-DAC 2015, Chiba, Japan, January 19-22, 2015*, pp. 48–53, IEEE, 2015.

[2] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, p. 20896, IEEE Computer Society, 2004.

[3] J. Henkel, W. H. Wolf, and S. T. Chakradhar, "On-chip networks: A scalable, communication-centric embedded system design paradigm," in *17th International Conference on VLSI Design (VLSI Design 2004), with the 3rd International Conference on Embedded Systems Design, 5-9 January 2004, Mumbai, India*, p. 845, IEEE Computer Society, 2004.

[4] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault tolerance overhead in network-on-chip flow control schemes," in *Proceedings of the 18th Annual Symposium on Integrated Circuits and Systems Design, SBCCI 2005, Florianolpolis, Brazil, September 4-7, 2005* (C. Galup-Montoro, S. Bampi, and A. Orailoglu, eds.), pp. 224–229, ACM, 2005.

[5] C. Ababei and R. S. Katti, "Achieving network on chip fault tolerance by adaptive remapping," in *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pp. 1–4, IEEE, 2009.

[6] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant network-on-chip multiprocessors," in *NOCS 2011, Fifth ACM/IEEE International Symposium on Networks-on-Chip, Pittsburgh, Pennsylvania, USA, May 1-4, 2011*, pp. 129–136, IEEE Computer Society, 2011.

[7] P. K. Sahu, N. Shah, K. Manna, and S. Chattopadhyay, "A new application mapping algorithm for mesh based network-on-chip design," in *India Conference (INDICON), 2010 Annual IEEE*, pp. 1–4, IEEE, 2010.

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

[9] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*, pp. 1:1–1:10, ACM, 2013.

[10] L. Lehtonen, "Transaction generator-tool for network-on-chip benchmarking," in *2010 Norchip Conference*, 2014.

[11] J. Hu, Ü. Y. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific networks-on-chip router design," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2919–2933, 2006.

[12] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A noc traffic suite based on real applications," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2011, 4-6 July 2011, Chennai, India*, pp. 66–71, IEEE Computer Society, 2011.