A Deterministic-Path Routing Algorithm for Tolerating Many Faults on Wafer-Level NoC

Zhongsheng Chen[1] Ying Zhang[1*] Zebo Peng[2] Jianhui Jiang[1]
[1] *School of Software Engineering, Tongji University, China*
[2] *Embedded Systems Lab, Linköping University, Sweden*
13166219587@163.com, yingzhang@tongji.edu.cn, zebo.peng@liu.se, jhjiang@tongji.edu.cn

Abstract—**Wafer-level NoC has emerged as a promising fabric to further improve supercomputer performance, but this new fabric may suffer from the many-fault problem. This paper presents a deterministic-path routing algorithm for tolerating many faults on wafer-level NoCs. The proposed algorithm generates routing tables using a breadth-first traversal strategy, and stores one routing table in each NoC switch. The switch will then transmit packages according to its routing table online. We use the Tarjan algorithm to dynamically reconfigure the routes to avoid the faulty nodes and develop the deprecated link/node rules to ensure deadlock-free communication of the NoCs. Experimental results demonstrate that the proposed algorithm does not only tolerate the effects of many faults, but also maximizes the available nodes in the reconfigured NoC. The performance of the proposed algorithm in terms of average latency, throughput, and energy consumption is also better than those of the existing solutions.**

I. INTRODUCTION

With its demonstrated parallel communication capability, large-scale Network-on-chip (NoC) has emerged as a promising fabric in supercomputers [1]. According to a report from the Oak Ridge National Laboratory [2], the SW26010 processor integrates 260 cores using a NoC architecture. The area of that processor is 25 square centimeters, and the radius of a wafer is usually 7-8 centimeters, hence such a large-scale NoC is referred to as a wafer-level NoC in this work. The outstanding performance of the wafer-level NoC helped Sunway TianHuLight supercomputer to stay as the fastest computer in the world for three years. A wafer-level NoC can move a great deal of parallel communication between chips to inside of a chip, which reduces transmission delay, and thus significantly improves computing performance. At the same time, such a fabric can save supercomputers from the energy wall as the fabric makes a single chip as a small supercomputer. Therefore, developing wafer-level NoCs becomes an efficient measure to further improve supercomputer performance.

Ensuring high yield is a key issue in developing wafer-level NoCs. A large-scale wafer-level NoC will encounter the many-fault problem as the probability of faults on a chip is proportional to the size of the chip. If each faulty wafer-level NoC is directly discarded when there is a fault anywhere in the NoC, the yield will be very low, and the production cost impractically high. A feasible solution to maintain high yield of wafer-level NoCs is implementing more processing routers (i.e., nodes) on the wafer; discarding nodes and links with faults; and treating a faulty chip as a good chip if the number of available nodes on the chip exceeds the design requirement. Fortunately, the NoC communication fabric supports this solution, since it provides naturally many alternative paths for inter-node communication. What we need is a routing algorithm to tolerate the potentially many faults that can occur during the manufacturing and operation phases of NoCs.

A number of techniques have been published [3-9] for fault-tolerant NoCs, but none of them is developed to tolerate many faults. This paper presents a deterministic-path routing algorithm (DPRA) to tolerate many faults on wafer-level NoCs. The algorithm exhaustively searches reachable paths between nodes offline, and provides a routing table for every switch. Then the switch transmits packages according to its routing table online in spite of the existence of many faults. The key contributions of this paper are as follows:

- The proposed routing algorithm can not only tolerate many faults, but also ensure the maximum delay of the switch within a single cycle period;
- The dynamical reconfiguration approach using the Tarjan algorithm [10] maximizes the available nodes on the NoC after reconfiguration;
- The proposed deprecated link/node rules ensues deadlock-free communication of NoCs without virtual channels.

The rest of the paper is organized as follows. Section II describes background material and related works for fault-tolerant NoCs. Section III presents the framework of the deterministic-path routing algorithm. Detailed implementation of the algorithm is presented in Section IV. Experimental results are presented in Section V. Finally, we conclude the paper in Section VI.

II. BACKGROUND

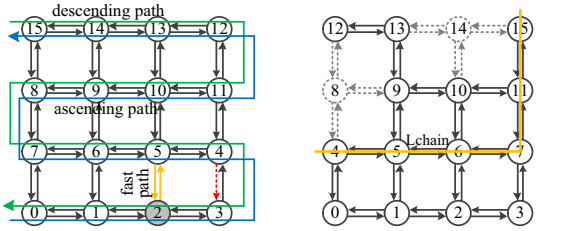
A. Faults and fault tolerance in Network-on-chip

This work only considers permanent faults in a NoC. According to the location of these faults, we classify them as link faults, core faults, and router faults. Every channel in a NoC contains two opposite links. Once a fault emerges on a link, we treat it as a faulty link. If a fault happens on a local core, we treat its input and output link as faulty links. Once a fault occurs on a router, we treats all links connected to it as faulty links. In this way, this high-level link model covers all faults appearing on a NoC, and the routing algorithm is proposed to tolerate these faulty links.

Evaluating the performance of a fault-tolerant NoC is a key issue to consider when designing such NoCs. We consider a set of factors to fully evaluate the NoC performance, including reliability, energy cost, average latency, throughput, maximum delay of the switch, and area overhead. Reliability refers to the ability of a NoC to execute applications without failure within a certain period of time. Energy cost refers to the total energy consumed by all NoC components during the period. The average latency is defined as the average transmission cycles of every packet from its source node to the destination node. The throughput refers to the average number of successfully transmitted packets in a time unit.

Ensuring deadlock-free is another important issue for the fault-

* To whom correspondence should be addressed.
This paper is supported by National Natural Science Foundation of China (NSFC) under grant No. (61432017, 61404092, 61772199).



(a) Fault Configuration for HamFA (b) Fault Configuration for ZoneDefense

Fig. 1 Examples of HamFA and ZoneDefense.

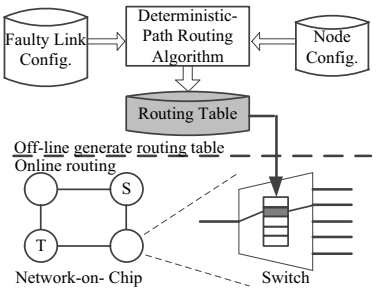


Fig 2 The framework of the deterministic-path routing algorithm.

tolerant NoC, because the packets involved in a deadlock will wait for each other forever and block the transmission of other packets. Designers often adopt virtual channels [11] to avoid deadlocks, but using virtual channels is complicated and expensive. It is, therefore, desirable to avoid deadlocks without virtual channels. In this work, we apply the popular turn model [3] to avoid deadlocks. A deadlock requires that the possible turns of transmission paths form a ring in the NoC. The turn model forbids a specific turn, hence can avoid deadlocks without virtual channels.

B. *Fault-tolerant Algorithms for NoC*

The HamFA algorithm presented in [6] is a very simple routing algorithm to tolerate link faults. The algorithm assigns a HamFA ID to each node of the NoC, as shown in Fig. 1(a). Then it connects the nodes in series based on their HamFA IDs and forms two paths: the ascending path and the descending path. The algorithm also sets the link from the current node (Node n2 in Fig. 1(a)) to the north-neighbor (or south-neighbor) node as a fast path, and the router will prefer to send packets through the fast path. When a fault emerges on the fast path, the router uses the ascending path (or descending path) to send the packet instead, and hence it tolerates the effect of such a single fault. However, when the fast path coincides with the descending (or ascending) path, and a fault just emerges on the corresponding link, the algorithm will fail to tolerate such a fault. For example, even if only a single faulty link from node n4 to node n3 (n4=>n3) exists, node n2 could no longer receive packets from node n4.

ZoneDefense [8] is a powerful algorithm to tolerate multiple node faults. The algorithm first determines unsafe nodes according to the existing faulty nodes. Then, it sets several rectangular zones to cover all unsafe nodes, and such a zone is called unsafe zone. Later, the algorithm designs Lchains (or Fchains) to surround these zones. Finally, even if multiple faulty nodes emerge, the algorithm can send out packets and bypass these faulty zones through the Lchains (or Fchains), and thus tolerate the effects of multiple faults. However, this algorithm sacrifices all original faulty-free nodes enclosed in the unsafe zone. For example, when nodes n8 and n14 are faulty, the algorithm designs an Lchain to surround the unsafe zone, and abandons four fault-free nodes, n12, n13, n9, and n10, in that zone, as shown in Fig. 1(b). Therefore, this algorithm is not applicable for NoCs with many faults as it hardly ensures that a NoC has enough available nodes left to meet design requirement.

III. The Framework of the Deterministic-Path Routing Algorithm (DPRA)

A deterministic-path routing algorithm (DPRA) is proposed to tolerate many faults in wafer-level NoCs. DPRA generates a routing table offline, and then requests the router to forward the packet online according to the routing table. Fig. 2 presents the framework of DPRA. In the offline phase, DPRA first reads in the configuration information about the NoC nodes and faulty links. Then it applies the breadth-first traversal algorithm, and generates a routing table for every node in the NoC. The breadth-first traversal algorithm ensures that all existing transmission paths are searched regardless of the NoC topology and the fault distribution. It is worth emphasizing that the breadth-first traversal algorithm also minimizes the length of the transmission paths.

When faults occur and several nodes become unreachable, DPRA uses the Tarjan algorithm to search the strongly connected components in the NoC topology. Then it replaces the faulty NoC with its maximum strongly connected component. This operation maximizes the number of available nodes in the degraded NoC.

At the beginning of the online phase, DPRA first loads the generated routing tables, using other channels (e.g. test channels), into the switches. Each switch determines then the hop direction for the transmitted packets by reading the table that has been stored locally. The online operation is very simple, and the maximum delay of its hardware will never exceed a single cycle period.

IV. Implementing DPRA on Wafer-Level NoCs

A. *Routing Table Generation*

The detailed implementation of DPRA is as follows. The nodes of a wafer-level NoC are first numbered in a row-first manner from the leftist and lowest corner, and a routing table is generated for one node at a time. The algorithm sets the current node as the starting node, and traverses other nodes in a breadth-first manner. When a node has multiple adjacent nodes, the algorithm traverses the node with the smallest number. The traversal process will generate a traversal tree with the current node as its root, and this tree contains all shortest routing paths from the current node. For example, Fig. 3(a) presents a 4*4 NoC. Assume that node n13 is the current node. Fig. 3(b) shows the traversal tree of n13. If a packet is to be sent from n13 to node n3, the packet can travel through the path from the root to node n3, highlighted in the routing tree. It is worth emphasizing that the breadth-first traversal minimizes the length of the transmission path. For example, the length of the path from n13 to n3 in the routing tree in Fig. 3(b) is just equal to the Manhattan distance between n13 and n3.

The proposed algorithm will then generate a routing table for the current node using the routing tree. This table stores the hop direction from the current node (the root) to each destination node. When a node is selected as the destination node, the algorithm first gets the path from the root to that node, marks the child node of the root on the path, and stores the orientation of the child node at the root node as the hop direction for the destination node. Only four possible values exist for the direction, and the algorithm uses a 2-bit variable to indicate the direction: East (00), South (01), West (10), and North (11). Fig. 3(c) presents the routing table for

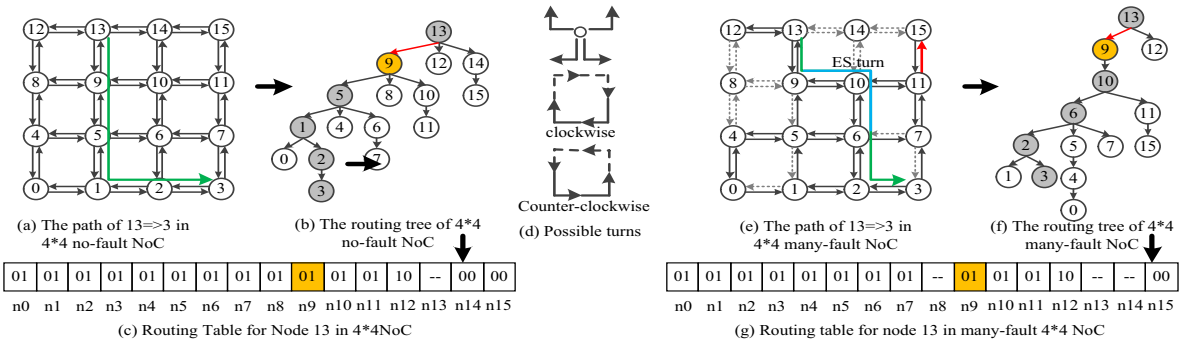


Fig. 3 The examples of generating the routing table using DPRA.

the current node n13, and the table is just a single register array whose cell is 2 bits. The sole online operation is to read these registers and get the hop direction. Hence the maximum delay of its hardware will never exceed a single cycle period.

When no fault exists, the proposed NoC is deadlock-free. As mentioned above, the algorithm will preferentially visit the adjacent node with the smallest number. Due to this rule, if a node forwards a packet to its southeast node (e.g., from n5 to n2), the node will select a path with a south-east (SE) turn (n5=>n1=>n2) instead of that with an east-south (ES) turn (n5=>n6=>n2). Therefore, the transmission path of any packet only contains four possible turns in the case of no faults. Fig. 3(d) presents these turns. The transmission paths cannot form a ring either clockwise or counter-clockwise, thus avoiding deadlocks.

DPRA can tolerate the effects of many faults. Assume that many faulty links occur on a 4*4 NoC, as shown in Fig. 3(e), where the dotted links are faulty links. We just require to remove all faulty links from the NoC topology, and then run DPRA. The breadth-first traversal is capable of searching a path to any destination node if the node is still connected to the current node n13. Moreover, DPRA guarantees to minimize the length of the searched paths in the case of no fault. For example, the traversal tree in Fig. 3(f) contains all nodes except two isolated nodes (n8 and n14), and the path length from n13 to n3 remains the Manhattan distance. The routing table is updated in Fig. 3(g).

Fig. 4 presents the DPRA pseudo code for generating the routing table. We combine the two steps of generating the traversal tree and generating the routing table using that tree together, and directly calculate and store the hop direction into the routing table when traversing the NoC topology.

B. *Reconfiguration using Tarjan Algorithm*

DPRA is also capable of reconfiguring a wafer-level NoC when some of its nodes are no longer connected, due to faults. In that case, DPRA uses the Tarjan algorithm and searches all strongly connected components in the NoC topology. Then the algorithm selects the maximum strongly connected component in the NoC topology as a new working NoC. Afterwards, DPRA sets the nodes outside the maximum strongly connected component as unavailable, and regenerates the routing table for every node. In this way, DPRA successfully degrades the computation capability of the NoC, but maintains fully connectivity of the degraded NoC. Fig. 5 presents the pseudo code for searching the maximum strongly connected component. Because the degraded NoC is the maximum strongly connected component of the original NoC, the algorithm maximizes the available nodes in the new NoC.

DPRA also supports dynamic reconfiguration. When a test technique detects a new fault in the NoC, DPRA first executes the Tarjan algorithm to determine if the NoC remains a strongly connected graph. If yes, the algorithm recalculates the traversal tree and updates each routing table; otherwise, it performs the degradation operation mentioned above, and updates the routing tables. For example, if the link n11=>n15 becomes faulty as highlighted in red color in Fig. 3 (e), DPRA determines that nodes n15 is no longer connected to the NoC. Then it disables n15 in the current NoC topology, and generates a smaller NoC. The degraded NoC contains now only thirteen nodes, and DPRA updates their routing tables. Finally, DPRA updates the routing table in each switch, using other channels (e.g. test channels). In this way, the effects of the new fault is tolerated.

C. *Deprecated Link/Node Rules for Avoiding Deadlocks*

DPRA applies the deprecated link/node rules to avoid deadlocks. Although it is deadlock-free when no fault exists, the occurrence of a fault may change the original transmission path, generate a new turn (e.g., the ES turn, n9=>n10=>n6, in Fig. 3(e)) that violates the turn model, and finally lead to a deadlock. According to the turn model, if some turns are forbidden, and the remaining turns on NoC never form a ring, the deadlock will not appear. In this work, we forbid the east-south (ES) and north-west (NW) turns. Furthermore, to reduce the number of sacrificial nodes, we develop three pre-processing operations for faulty links/nodes:

P1. *When a faulty link appears on the west (or south) boundary of the NoC, we also set its opposite link as a faulty one, and call them as deprecated links.*

P2. *When a fault emerges on the east-neighbor (or north-neighbor) link of the vertical (or horizontal) deprecated links, we set the faulty link and its opposite link as deprecated links.*

P3. *When the west and south links of a node are not all deprecated links, but the west output link and the south output link are both faulty links, we set the node as a deprecated node.*

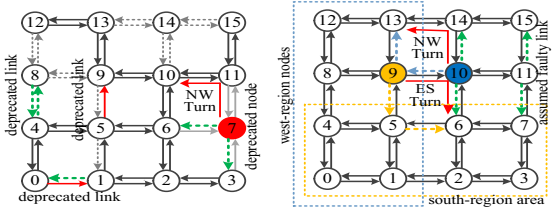
The algorithm can easily execute these pre-processing operations offline. For example, two opposite links between n0 and n1 in Fig. 6(a) are treated as deprecated links according to P1; two opposite links between n9 and n5 are also treated as deprecated links according to P2; the node n7 is treated as a deprecated node according to P3, and thereby a possible NW turn is forbidden. After these pre-processing operations, the algorithm develops two rules to prohibit the ES and NW turns, prevents the turns from forming a ring, and thereby avoids deadlocks.

```
Input: current id(id), Information of fault
Output: routing table of current node(table)

table,queue
table[id]=DIRECTION_LOCAL
queue.enqueue(id)
While queue not empty
  node = queue.dequeue
  neighbors = countNeighbors(node)
  For i = 0 to neighbors.size
    neighbor = neighbors[i]
    If neighbor not traversed
      queue.enqueue(neighbor)
      If node == id
        table[neighbor] = countDirection(node,neighbor)
      Else
        table[neighbor] = table[node]
      End If
    End If
```

Fig. 4 The algorithm for routing table generation.

```
Input: Number of nodes(n), Information of fault
Output: Maximum strongly connected component(MSCC)

MSCC , DFN , Low , stack , index=0
For i = 0 to n
    If !DFN[i] tarjan(i)
    End If
tarjan(u):
    DFN[u] = Low[u] = ++index
    stack.push(u)
    For each link (u , v) in NoC
        If v is not visted
            tarjan(v)
            Low[u] = min( Low[u] , Low[v] )
        Else If v in stack
            Low[u] = min( Low[u] , DFN[v] )
        End If
    If DFN[u] == Low[u]
        tempSCC
        Repeat v = stack.pop
               tempSCC.add(v)
        until (u == v)
        If tempSCC.size > MSCC.size
            MSCC = tempSCC
        End If
    End If
```

Fig. 5 The algorithm for the maximum strongly connected component.



(a) Examples of pre-processing rules. (b) Rules of avoiding deadlock.

Fig. 6 The deprecated link/node rules.

R1. *If both the west links of the current node and the north links of the west-neighbor node of the current node are not deprecated links, but either the west output link of the current node or the north output link of the west-neighbor node is faulty, we assume the north output links at the east side of the faulty link are faulty when calculating the hop directions for the nodes whose X coordinates are smaller than that of the current node (called west-region nodes). Then, we updates the routing table of the current node for these west-region nodes, and the routing tables of the nodes on the paths to these west-region nodes.*

R2. *If both the north links of the current node and the east links of the south-neighbor node of the current node are not deprecated links, but either the south output link of the current node or the east output link of the south-neighbor node is faulty, we assume the south output links at the east side of the faulty link are faulty when calculating the hop direction for these south-region nodes. Then, we updates the routing table of the current node for these south-region nodes whose Y coordinates are smaller than that of the current node, and the routing tables of the nodes on the paths to these south-region nodes.*

Fig. 6(b) presents two examples for the above rules. Assume the current node is n10, if a fault emerges on the link n10=>n9, rule R1 is excited. We assume the links n10=>n14 and n11=>n15 are faulty when calculating the hop direction for the nodes inside the blue box consisting of nodes n0, n1, n4, n5, n8, n9, n12, and n13. Then we update the routing table of the current node for these blue-box nodes as well as the routing tables of the nodes n6 and n2. Due to R1, the NW turn will not appear. Assume the current node is n9, if a fault emerges on the link n9=>n5, rule R2 is excited. We assume the links n10=>n6 and n11=>n7 are faulty when calculating the hop direction for the node inside the yellow box consisting of nodes n0, n1, n2, n3, n4, n5, n6, and n7. Then we update the routing table of the current node for these yellow-box nodes as well as the routing tables of node n8. Due to R2, the ES turn will not appear.

We use the counter-evidence method to prove that the algorithm is deadlock-free. We assume a counter-clockwise ring emerges even though the NoC meets the deprecated link/node rules. That means at least one NW turn happens. Hence, normal transmission paths in a fault-free NoC are blocked, and packets are forced to travel along the NW turn. In this case, at least one of these two conditions will occur.

C1. *At least one pair of horizontal deprecated links or vertical deprecated links emerge in the zone of that ring.*

C2. *Either a faulty west output link or a faulty north output link exist in the zone of that ring.*

However, once a pair of horizontal deprecated links (or vertical deprecated links) exists, all links at the south side of the faulty link are deprecated links due to P1 and P2. That violates the assumption that a counter-clockwise ring exists. Meanwhile, if a faulty west output link (or a faulty north output link) exits in the zone of that ring, R1 is excited, all north output links at the east side of the faulty link are assumed to be faulty ones when calculating the hop directions. Hence the NW turn will not appear. Hence the counter-clockwise ring does not exist. The same argument can be used for the clockwise ring. In conclusion, no transmission ring exists in the NoC, and the method avoids deadlocks without virtual channels.

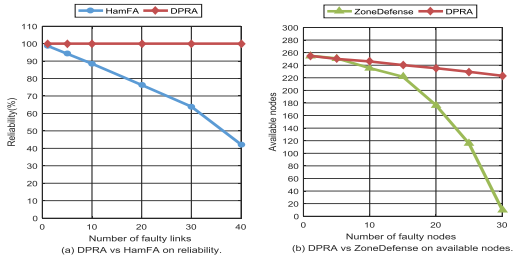


Fig. 7 The reliability of different algorithms.

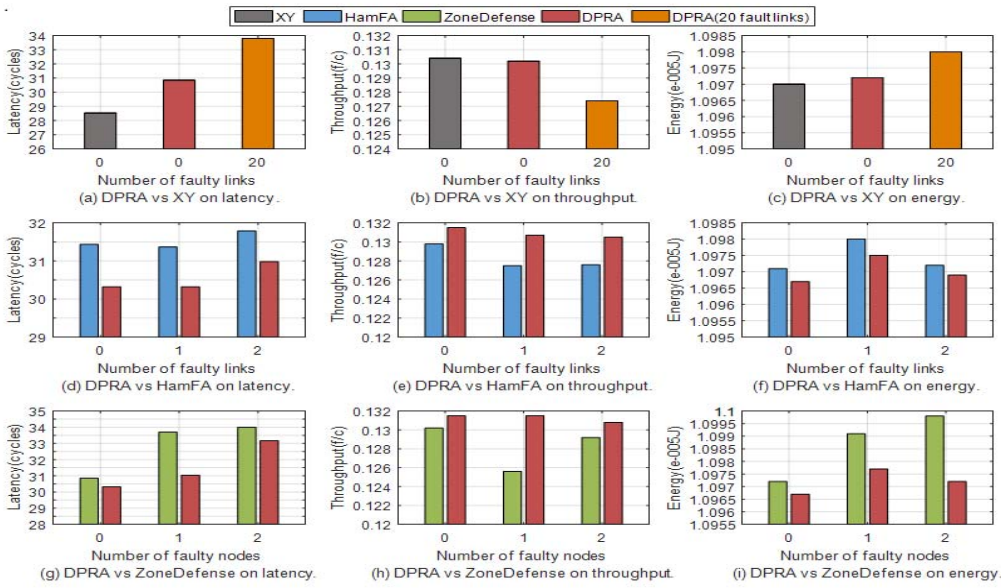


Fig. 8 Performance comparison for different methods on a 16*16 NoC.

V. Experiments

We have evaluated the performance of the NoC based on DPRA with respect to reliability, the number of available nodes, latency, throughput, energy, maximum delay of the switch, and hardware area overhead. For comparison, we use XY routing algorithm, HamFA, and ZoneDefense as references. We implement all these algorithms in a 16*16 NoC (very likely to be implemented as a wafer-level NoC) using the Noxim simulator [12], where we use the random traffic mode, and calculate the performance metrics by the average value from multiple operations. To accurately evaluate the reliability and the available node count, we set different number of faulty links for HamFA, and different number of faulty nodes for ZoneDe-fense. Each fault count also contains several fault configurations, and we calculate their average values. When evaluating latency, throughput, and energy, we inject only one or two faults into the NoCs, and compare these factors of different algorithms because HamFA and ZoneDefense are not suitable for many-fault scenarios.

We have also applied these algorithms on a switch of the Herms NoC [13] that uses the XY algorithm, and evaluate the cost of the hardware implementation of these algorithms on 6*6, 10*10, and 16*16 NoCs, respectively. In addition, we only implement the function that calculates the hop direction using the global configuration of Lchain and Fchain for ZoneDefense because the function that generates the global configuration of Lchain and Fchain is too complex to be applied using hardware. Then we synthesize these switches as well as the original switch with a 90nm lib, and then get their maximum delay and area overhead.

A. Reliability of Different Algorithms

The experimental results demonstrate the powerful capacity of DPRA for tolerating many faults. When we inject a number of faulty links to the 16*16 NoC, the reliability of DPRA always maintains at a high level, as shown in Fig. 7(a), regardless of the number of faulty links. This is because the breadth-first traversal can search any reachable path for packet transition if that path exists. The deprecated link/node rules have also successfully avoided deadlocks in the NoC. They ensure that any sending packet is received by its destination node. As a comparison, the reliability of HamFA quickly declines when the number of faulty links increases, as shown in Fig. 7(a). This is because HamFA cannot tolerate the effects of many faults. Even if a single fault occurs on a link where the ascending path (or the descending path) overlaps the fast path, the packets transmitted through that link cannot reach their destinations.

When we inject a number of faulty nodes to the NoC, the available nodes of DPRA slowly decreases as the number of faulty node increases, as shown in Fig. 7(b). This is because DPRA uses the Tarjan algorithm to search the strongly connected components, and reconfigure the faulty NoC as its maximum strongly connected component. ZoneDefense can also tolerate the effects of multiple faults, but it sacrifices all fault-free nodes in the unsafe zone. In Fig. 7(b), the available nodes of ZoneDefense is reduced exponentially with the increase of the number of faulty nodes. Excessively sacrificing fault-free nodes makes ZoneDefense not applicable for NoC with many faults, since this algorithm will often lead to the situation that the design requirement in respect to the number of available working nodes can't be met. In conclusion, DPRA does not only tolerate the effects of many faults, but also maximizes the available nodes in the reconfigured NoC.

B. Performance of Different Algorithms

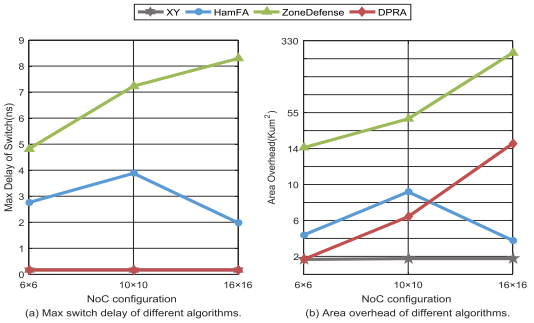
DPRA also outperforms the existing solutions in term of average latency, throughput, and energy consumption. As shown in Fig. 8(a)-8(c), the average latency, throughput, and energy of DPRA on the 16*16 fault-free NoC are very close to the simplest XY routing algorithm. When the NoC is fault-free, the breadth-first searching strategy ensures to search the shortest path for every packet transmission. Even if we inject 20 faulty links in the NoC, the performance of DPRA is not significantly reduced compared with that of the XY algorithm without fault-tolerating capability. On the other hand, existing fault-tolerant algorithms fail to search short paths for packet transmission when faults emerge. As Fig.

8(d)-8(i) show, the performances of HamFA and ZoneDefense lag behind that of DPRA in the case of no fault; and the gap becomes more obvious when faults occur. In the HamFA algorithm, some packets often have to be transmitted through the longer ascending path (or descending path), or waiting in the buffer forever when some faulty links emerge. In the ZoneDefense algorithm, many packets are forbidden to travel through the adjacent and fault-free node in the unsafe zone when some faulty nodes appear nearby. Instead, these packets will be transmitted along the Lchain or Fchain, and that will increase the length of the transmission path. All these conditions have led to the reduction of their performance.

C. Cost of Hardware Implementation

The hardware implantation of DPRA does not cause large delay in the online operation phase, because the sole online operation is just to read the routing table. As shown in Fig. 9(a), the delay of its switch is very small, and almost coincides with that of the XY algorithm. The delay of HamFA is higher than that of DPRA, because it requires to calculate the HamFA IDs, perform conversions between HamFA IDs and node coordinates, and determines the hop direction. These operations cause a lot of delay. In addition, some unusual situations occur on 6*6 and 10*10 NoCs. The synthesis process brings in a special divider for Ham FA to calculate the HamFA IDs, when the value of the NoC X-axis dimension is not equal to the power of 2. The delay of ZoneDefense is much higher than that of DPRA, and quickly grows when the size of NoC increases. Even if ZoneDefense only calculates the hop direction using the existing configuration of Lchain and Fchain, the algorithm also requires a lot of calculation. Hence, its hardware implementation is inevitably complex, and requires a lot of delay. The large delay of the above two methods exceeds a single cycle period, and consequently the NoC has to assign multiple cycles to their switches. In conclusion, the simple online operation of DPRA ensures that it never jeopardizes the timing constraint of having the switch operation performed in a single cycle period.

The area overhead of DPRA is reasonable for tolerating many faults. Most operations of DPRA are done offline, and its switch only requires a simple register array to provide hop direction online for every destination node. As a comparison, HamFA requires lower area overhead than DPRA because it just uses simple local information of faulty links. However, due to the lack of the global configuration of faulty links, HamFA is hardly able to tolerate the effects of many faults. For ZoneDefense, calculating the hop direction to bypass all faulty links is a computation-intensive operation. In Fig. 9(b), the hardware implementation of ZoneDefense to only calculate the hop direction using the solved configuration of Lchains and Fchains is already far more than that of DPRA. Therefore, the online routing method using a simple routing table is an ingenious method to reduce area overhead.



(a) Max switch delay of different algorithms. (b) Area overhead of different algorithms.

Fig. 9 Maximum delay of switch and area overhead of different algorithms.

VI. Conclusions

This paper develops a deterministic-path routing algorithm for tolerating many faults on wafer-level NoCs. The algorithm uses a breadth-first traversal strategy to search the shortest and reachable path between nodes, and then generates a routing table for each node offline. Once a link is not connected due to a fault, the algorithm uses the Tarjan algorithm to search the strongly connected components, and replace the faulty NoC with its maximum strongly connected component. We have also developed the deprecated link/node rules for avoiding deadlocks when faults occur and the routing paths need to be updated. Experimental results have demonstrated that the proposed algorithm does not only tolerate the effects of many faults, but also maximizes the available nodes in the reconfigured NoC.

References

[1] Y. Zhang, K. Chakrabarty, H. Li and, J. Jiang, "Software-based online self-testing of network-on-chip using bounded model checking", in *Proc. Int. Test Conf.*, Fort Worth, TX, 2017, pp.1-10.

[2] J. Dongarra, "Report on Sunway TaihuLight System", *Oak Ridge National Laboratory*, Jun. 2016.

[3] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," in *Proc. Annual International Symposium on Computer Architecture*, Gold Coast, Australia, 1992, pp. 278-287.

[4] R. Alizadeh, M. Saneei and M. Ebrahimi, "Fault-tolerant circular routing algorithm for 3D-NoC," in *Proc. International Congress on Technology, Communication and Knowledge*, Mashhad, 2014, pp. 1-7.

[5] M. Ebrahimi, M. Daneshtalab, J. Plosila and H. Tenhunen, "MAFA: Adaptive Fault-Tolerant Routing Algorithm for Networks-on-Chip," in *Proc. Euromicro Conference on Digital System Design*, Izmir, 2012, pp. 201-207.

[6] M. Ebrahimi, M. Daneshtalab and J. Plosila, "Fault-tolerant routing algorithm for 3D NoC using hamiltonian path strategy," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, Grenoble, France, 2013, pp. 1601-1604.

[7] S. Maabi, F. Safaei, A. Rezaei, M. Daneshtalab and D. Zhao, "ERFAN: Efficient reconfigurable fault-tolerant deflection routing algorithm for 3-D Network-on-Chip," in *Proc. IEEE International System-on-Chip Conference*, Seattle, WA, 2016, pp. 306-311.

[8] B. Fu, Y. Han, H. Li and X. Li, "ZoneDefense: A Fault-Tolerant Routing for 2-D Meshes Without Virtual Channels," *IEEE Transactions on VLSI Systems*, vol. 22, no. 1, pp. 113-126, Jan. 2014.

[9] H. Zhao, N. Bagherzadeh and J. Wu, "A General Fault-Tolerant Minimal Routing for Mesh Architectures," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1240-1246, 1 July 2017.

[10] R. E. Tarjan, "Depth-first search and linear graph algorithms", *SIAM Journal on Computing*, 1972, vol. 1(2), pp.146–160,

[11] C. R. Jesshope, P. R. Miller and J. T. Yantchev, "High Performance Communications In Processor Networks," in *Proc. Annual International Symposium on Computer Architecture*, Jerusalem, 1989, pp. 150-157.

[12] V. Catania, A. Mineo, S. Monteleone, M. Palesi and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proc. International Conference on Application-specific Systems, Architectures and Processors*, Toronto, ON, 2015, pp. 162-163.

[13] Hermes NoC. [Online]. Available: http://toledo.inf.pucrs.br/~grph/Projects/Hermes/Herme s.html