Microprocessors and Microsystems 45 (2016) 198–207

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

A runtime fault-tolerant routing algorithm based on region flooding in NoCs

Lu Wang[a,b,*], Sheng Ma[a,b], Zhiying Wang[a,b]

[a] State Key Laboratory of High Performance Computing, National University of Defense Technology, China
[b] College of Computer, National University of Defense Technology, Changsha, China

ARTICLE INFO

Article history:
Received 7 April 2015
Revised 9 May 2016
Accepted 12 May 2016
Available online 14 May 2016

Keywords:
Reliability
Network-on-chip
Fault tolerance
Communication protocol
Region flooding algorithm

ABSTRACT

Aggressive scaling of the CMOS process technology allows the fabrication of highly integrated chips, and enables the design of the network-on-chip (NoC). However, it also leads to widespread reliability problems. A reliable NoC system must operate normally even in the face of a lot of transistor failures. Aiming towards permanent faults on communication links, we introduce a fault-tolerant MPI-like communication protocol. It detects the link failure if there exist unresponsive requests and automatically starts the new path exploration. The region flooding algorithm is proposed to search for a fault-free path and reroute packets to avoid system stalls. The experimental result shows our approach significantly reduces the latency compared with the basic flooding algorithm. The maximum latency reduction is 25% under the bit complement traffic pattern. Also, it brings only 2% fault tolerance loss.

1. Introduction

The Moore's Law scaling is continuing to yield even higher transistor density with each succeeding process generation, leading the design of the network-on-chip (NoC). Unfortunately, the deep sub-micro CMOS process technology is marred by increasing susceptibility to wear out [1]. Widespread reliability challenges are expected in nearest fabrication technologies. Building a fault-tolerant NoC system should be concerned as a necessity. Actually, traditional fault-tolerant algorithms such as using repetitive structures [2–4] are infeasible for the NoC due to area restrictions [5]. Several solutions have been proposed to design a reliable NoC system [6–8], especially for transient or permanent faults on links or routers. These approaches generally provide reliability from four different hierarchies: link control, router control, network interface control and end to end control.

In this article, we mainly address permanent and hard errors on links which result in flit dropping from the hierarchy of end to end control. The error control generally involves three basic steps: detection, containment, and recovery [9]. However, previous works mostly focus only on one step of them. We try to establish an integrated hardware-software framework involving the runtime detection of faulty links, containment of link failure and reconfiguration of healthy routes. Hence, this paper proposes a fault-tolerant MPI-like communication protocol. It detects link failure if there are unresponsive requests and automatically starts the new path exploration.

A key issue to be solved is providing a fault-tolerant routing algorithm, which is discussed in several works [10–13]. A good fault-tolerant routing algorithm means low route latencies and minimal extra consumption. However, most proposed algorithms [10–12] have special restrictions on the number of faulty links as well as their locations.

Watchter et al. [13] have coped with this problem recently. To provide high scalability, they adopted a typical MPI-like protocol for core-to-core communication. The source node detects link failure through unresponsive requests and broadcasts seek packets through the entire mesh NoC to obtain an alternative healthy route. This approach takes advantage of the path redundancy between a pair of nodes and successfully combines high performance of hardware with high flexibility of software.

Although broadcasting seek packets to all other nodes provides complete reachability, it also brings unnecessary packet transmissions. Actually, in most cases, we can find an alternative path within the minimum rectangle defined by source and destination nodes. Based on this observation, we introduce a region flooding algorithm to efficiently search for a fault-free path. It makes use of the NoC's regular structure to direct a search following the minimal path to the destination and dramatically improves network efficiency by limiting the search area.

Generally, a better fault tolerance means a worse performance. Although Wachter et al.'s methodology [13] can perform the best

* Corresponding author.
E-mail addresses: wwanglu1991@gmail.com, 734809187@qq.com (L. Wang), masheng@nudt.edu.cn (S. Ma), zywang@nudt.edu.cn (Z. Wang).

http://dx.doi.org/10.1016/j.micpro.2016.05.004
0141-9331/© 2016 Elsevier B.V. All rights reserved.

fault tolerance, our region flooding approach significantly optimizes the latency with little fault tolerance loss. By analyzing the synthesized influence of fault tolerance and latency, we conclude that our approach is suitable for NoCs, especially for large systems with low fault rates.

Our contributions concentrate on two aspects. First of all, our design arrives at an optimal trade-off between fault tolerance and performance which has not been discussed by previous work [10–17]. Next, our work simultaneously addresses fault detection, containment and recovery.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents a fault-tolerant MPI-like communication protocol. Section 4 proposes the design of the NoC architecture and the network interface. Section 5 introduces our approach of searching for a fault-tolerant route and also discusses the deadlock avoidance mechanism and router pipeline. The simulation as well as evaluation are presented in Section 6. After that, we draw a conclusion in Section 7.

2. Related work

Fault-tolerant routing algorithms have been discussed for many years. Previous works mostly focus on adaptive fault-tolerant routing algorithms for mesh networks. Chien and Kim [10] proposed the fault-tolerant planar adaptive routing (PAR) algorithm for n-dimensional meshes. Their algorithms can tolerate rectangle faults with no overlapping of f-rings. Su and Shin [11] proposed an adaptive fault-tolerant routing algorithm for n-dimensional meshes. Their algorithms can tolerate a disconnected rectangular block in an n-dimensional mesh. However, these algorithms can only be used in special topologies or special region shapes such as L, T or +.

Recently, some topology-agnostic fault-tolerant routing algorithms are discussed [14–17]. Dumitras et al. [15] proposed a probabilistic flooding scheme. Costas et al. [16] introduced a deadlock free hybrid routing algorithm, utilizing load-balancing routing on fault-free paths to support high performance and providing pre-reconfigured escape path in the vicinity of faults. Aisopos et al. [17] obtained an alternative path by broadcasting reconfiguration flags upon any number of concurrent network faults in any location. Watcher et al. [13] presented a novel fault-tolerant communication protocol that takes advantage of intrinsic redundancy of the NoC to provide alternative paths between any source-target pair, even in the presence of multiple faults. Different from these algorithms, our approach restrains the search in a rectangular area rather than the entire NoC, which dramatically decreases the latency and improves network efficiency.

There is also some work on designing fault-tolerant message passing libraries as a fault recovery method [18–21]. For instance, Batchu et al. [20] tested unresponsive processes by implementing self-checking threads which use heartbeat messages to monitor the MPI/FT progress. Aulwes and Daniel [21] proposed fault-tolerant mechanisms for the MPI such as the checksum, message retransmission, and automatic message re-routing. The timeout seeking mechanism in our proposed fault-tolerant MPI-like communication has similar idea with these work.

3. Fault-tolerant MPI-like communication protocol

3.1. Basic communication protocol

The basic communication protocol between nodes in this work is message passing. Two MPI-like primitives are adopted: *MPI_Send()* and *MPI_Receive()*. The communication protocol in our approach derives from the non-blocking synchronous communication mode. Fig. 1 illustrates the communication procedure between two tasks. Particularly, the task A and B are mapped to the node A and B respectively.

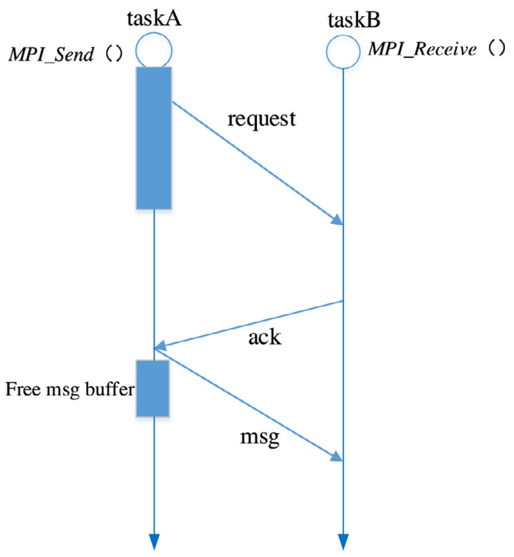


Fig. 1. Basic MPI-like communication protocol between two nodes.

During one point to point communication, task A executes the function *MPI_Send ()* on the source node. First, the node A sends a *request* message through the NoC. Being a control message, the request message has a single flit. After sending a *request* message, some information such as the destination, the tag, the starting address and the size of the data message should be written to '*msg buffer*' which is a dedicated memory space or register stacks in the node A. At the same time, the computation could be carried on in the task A. Only after an acknowledgment from the destination has been received, the node A will free the corresponding '*msg buffer*' and send the data message to the node B.

On the destination node, the task B executes the function *MPI_Receive()* and waits for a request message. After receiving the expected *request* message, it will send an acknowledgment (*ack*) to the source node. Once there exist faulty links in the communication path which cause interruptions for the request or acknowledgement messages, the data message will not be transferred normally and the system will stall. So we improve this communication protocol by adding fault tolerant designs to ensure service qualities.

3.2. Fault tolerant communication protocol

Fig. 2 describes the proposed fault-tolerant communication protocol. It illustrates two fault scenarios, the link failure from A to B when sending a *request* message and link failure from B to A when sending an acknowledgment message. In order to tolerate these faults and provide a reliable service, a timeout seeking mechanism has been added. We suppose that there are faulty links in the original path if the task A cannot receive an acknowledgment from the task B during a scheduled time. Particularly, the scheduled time *d_time* is set according to the average latency in the fault-free circumstances. After that, a seek message will be triggered to find a fault-free path between source and destination nodes. When the node B receives the *seek* message, it will return a *track* message which contains a recording of the new fault-free route. The node A then updates the route table and delivers the data message with the new healthy route. However, new faults may appear in this path. In order to deal with the occasion where new faults appear during the current data transmission, the node B should send a seek packet to the node A if it does not receive expected packets

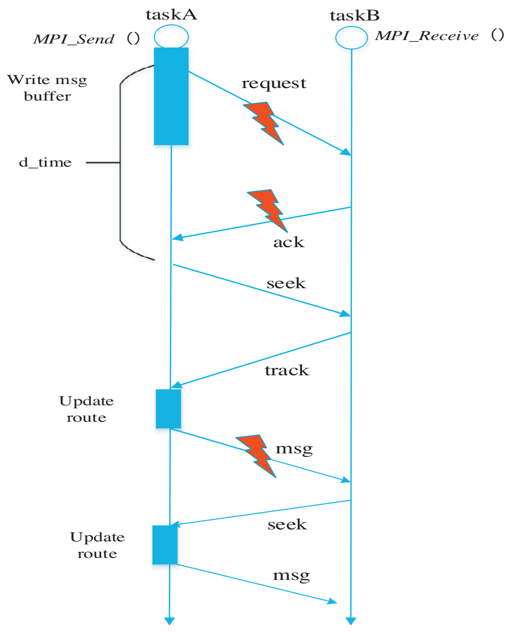


Fig. 2. Fault-tolerant MPI-like communication protocol.

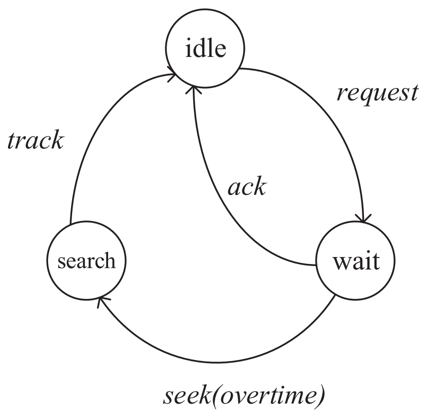


Fig. 3. State transition in source node.

in a period of time. Expected packets id should be included in the seek packet to notify the source node. When the node A receives the seek packet, it will update the route table again and retransmit un-received packets of the node B. If the node B receives expected packets from the first transmission later, it will drop retransmission packets. However, the possibility of this occasion is very low because the period of one point to point communication is quite short. More generally, new faults will appear after the current data transmission. The node A will detect if there exist faulty links on this new path again through sending request message in the next communication with task B and it will also send seek packets to update route if new faults appear.

3.3. Triggering of seek packets

A critical problem to be settled in the implementation of our *fault-tolerant MPI-like communication protocol* is when and how to trigger a seek packet. In order to handle it, an N-entry *seek state table (SST)* is added to the network interface. The i-th entry records corresponding *state* and *d_time* of the *seek* packet whose destination is the node i. Fig. 3 describes the transitions of this 2-bit FSM. The state is initialized to be the ‘idle’ state. When a request message is sent to the target node, it will turn into the ‘wait’ state.

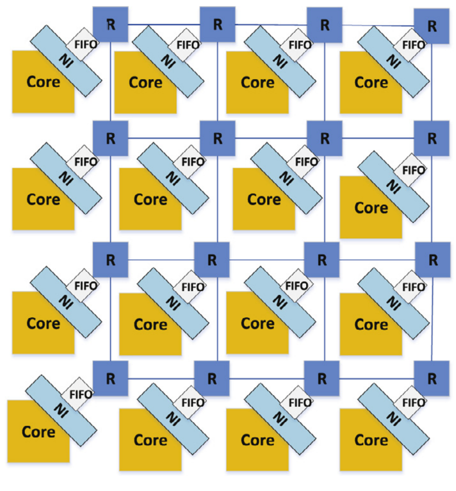


Fig. 4. The NoC architecture.

After receiving the acknowledgment from the destination, it will be updated to be the ‘idle’ state again. However, if the source node can not receive an acknowledgment during the scheduled time, the *seek* packet will be triggered and the state will turn into the ‘search’ state. When the source node receives a *track* packet, the state will turn back to be the ‘idle’ state. Hence, each time the source node scans its *seek state table (SST)*, if ‘state’ field is the ‘wait’ state and the current time overtakes the scheduled time, a *seek* packet will be triggered. We will discuss the appropriate value for the scheduled time later. A fixed amount of time to start the seek process is not suitable for all applications. A communication intensive application may stall for a long period while a computation intensive application may not. We define this scheduled time as $K \times AVG$. Here, *AVG* means the average latency for a fault-free NoC. And *K* is a constant related to the characteristics of a particular application.

4. Communication architecture

4.1. NoC architecture

Fig. 4 shows the block diagram of the NoC architecture with a 4×4 2D mesh topology in our design. Each node contains a local processor (*Core*) and a network interface (*NI*) module. A FIFO buffer is leveraged to connect the NI and the router. We use the distributed XY routing algorithm as the basic routing algorithm. However, after a new healthy path is updated in the route table, we will choose the source routing algorithm. The network interface is used to support the implementation of *fault-tolerant MPI-like communication protocol* such as triggering *seek* packets. Also, route tables are stored in the NI.

4.2. Design of network interface

The network interface is an important module for implementing our *fault-tolerant MPI-like communication protocol*. There are five types of packets: *the request, acknowledgment, seek, track* and *data* message. When to generate the corresponding packet and how to select a correct packet to deliver are illustrated next. As shown in Fig. 5, four packet generators are added in the network interface. The *Request* packet generator is triggered by the *MPI_send* operation. The *Data* message generator is used to build the data message. It is triggered by receiving an acknowledgment packet or a track packet. And related control information, data payloads and route information are fetched from the message buffer, cache and route table respectively. Once receiving a *request* packet or a *seek*

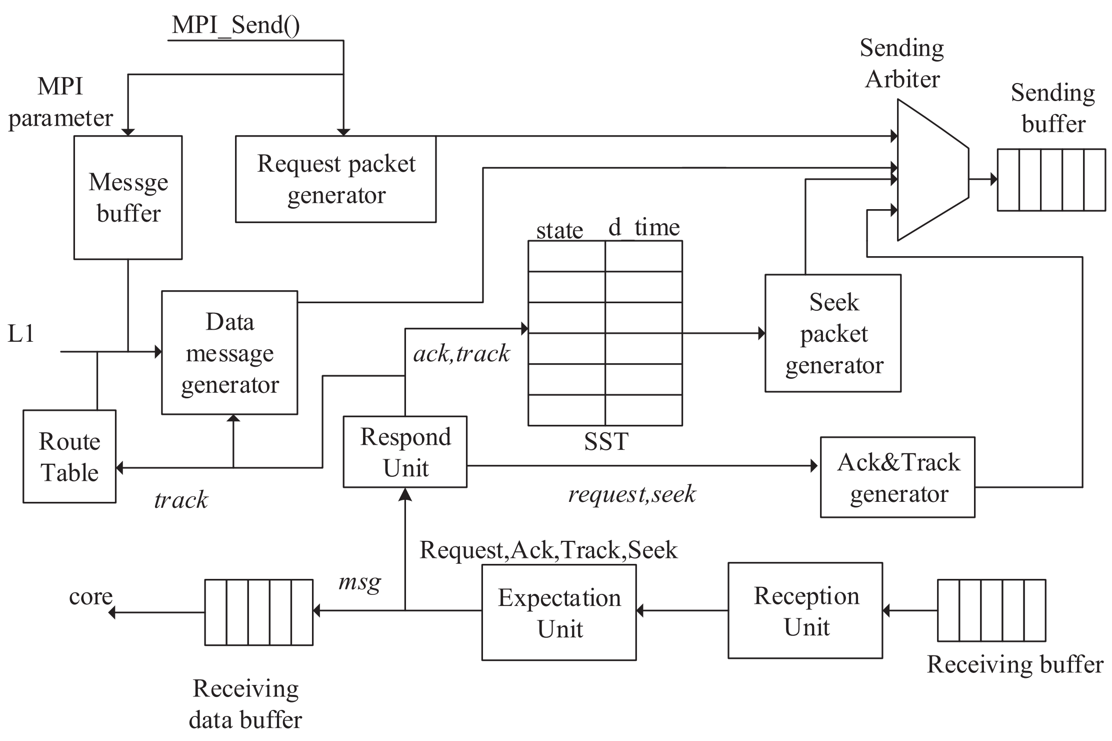


Fig. 5. Network Interface diagram.

Table 1
Seek packet format.

src	dest	type	pid	f_id	**m_id**	hops	**outport1**	**outvc1**	**outport2**	**outvc2**	...	**outportN**	**outvcN**

packet, an *acknowledgment* packet or a *track* packet will be generated through the corresponding unit. A *seek* packet is triggered based on the seek state table *(SST)* described in Section 3.3. And a sending arbiter is utilized to decide which type of the packet is selected to send. Here, we define the priority of the packet based on their types. The acknowledgment and track packets have the highest priority. The seek packet takes the second position. Next comes the data message packet. The request packet bottoms the list of all kinds.

5. Fault-tolerant routing algorithm

In this article, we use the region flooding algorithm to search for a fault-free route in most cases. However, for some special cases with one or two faults, we can directly reconfigure routes. Particularly, our fault-tolerant routing algorithm provides 100% fault tolerance for the single fault network.

5.1. Region flooding algorithm

Our region flooding algorithm consists of three steps. (1) send seek packets to find a new healthy path. (2) backtrack the path. (3) obtain a new path and update the route table.

5.1.1. Seek packet format

In our approach, *seek* packets are delivered through the region flooding algorithm to find a fault-free route. Different from broadcasting *seek* packets to all neighbors [13], it restrains each step of searching towards the destination. Hence, all paths to be searched have minimal hops. To trace these paths, corresponding information is recorded in seek packets. The format of seek packets is described in Table 1. Bold fields present extra payloads. The *'m_id'* field is reserved to distinguish different copies of one seek packet. This field is initially set to be zero and is updated when a *seek* packet replicates during transmission. The 2-bit *'outport i'* field records the output port of each hop and the 1-bit *'outvc i'* records the chosen virtual channel which will be discussed in Section 5.1.3 in detail. In an $n \times n$ network, the maximum hop count equals to 2n. So, the length of extra information of the seek packet is O(n).

To eliminate redundant seek packets, an $n \times n$-bit table initially to be 0 is added to each router. When a *seek* packet from the i-th source node to the destination j firstly arrives at this router, it updates the $(i \times n + j)$-th bit to be '1'. And latter *seek* packets drop if the corresponding bit is set to be '1'. When the source node obtains a new path, it will request all routers in the searching area to clear this bit.

5.1.2. A delivery example of seek packet

A scenario of two faulty links within a 4 × 4 mesh network is illustrated in Fig. 6. In this example, R0 is the source node while R10 is the destination node. M2 is transferred to the current node R5. Because R5 locates in the south-west of the destination, we choose the north and east directions to search. M2 is forwarded through the east port and a new generated packet M5 is forwarded through the north port. Particularly, each router only receives the first arrival seek packet. The latter arrival ones are dropped since we need to find the minimal-latency route.

Table 2 shows the content of each packet. First, a *seek* packet M0 is injected to R0. Then, it routes through the east output port, updates the 'hop' field and sets 'outport 1' field to be 'E'. At the same time, it is replicated into M1 whose output port is the north port.

Table 2
Key contents of seek packets in Fig. 6.

packet	last router	src	dest	hops	outport1	outport2	outport3	outport4
M0	R0	0	10	1	E			
M0	R1	0	10	2	E	E		
M1	R0	0	10	1	N			
M2	R1	0	10	2	E	N		
M2	R5	0	10	3	E	N	E	
M3	R4	0	10	2	N	N		
M3	R8	0	10	3	N	N	E	
M5	R5	0	10	3	E	N	N	
M5	R9	0	10	4	E	N	N	E
M6	R2	0	10	3	E	E	N	

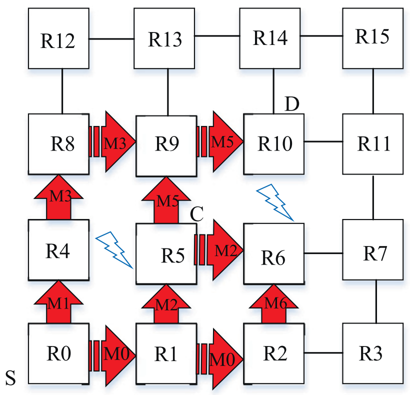


Fig. 6. Scenario of two faulty links within a 4 × 4 mesh network.

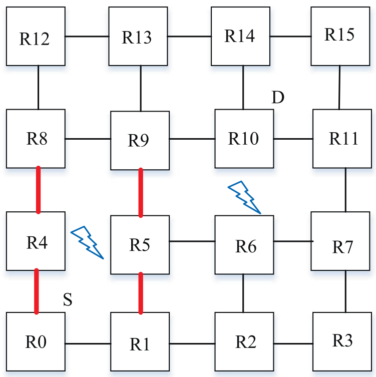


Fig. 7. Virtual channel selection example. R0 is the source node while R10 is the destination node. Only adaptive virtual channels can be selected in bold links.

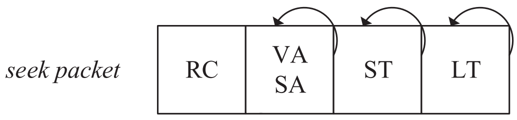


Fig. 8. Router pipeline of seek packets.

When it comes to router 1, M0 is transferred through the east port and 'outport 2' field is set to be E. A replicated packet M2 is generated and routed through the north port. Note in router 9, only the first arrival packet will be forwarded. Assuming M5 arrives R9 earlier, then M3 is discarded. The seeking procedure continues until the first seek packet reaches the target node. In this scenario, we get a fault-free path based on M5 which is (E,N,N,E).

5.1.3. Deadlock avoidance and router pipeline

On one hand, in order to minimize the influence on the point to point transmission, we add extra virtual channels to transfer seek packets. And we use the round-robin mechanism to select virtual channels. Through this kind of segregation, network loads caused by seek packets can hardly influence the point to point communication. On the other hand, in order to minimize extra resources and avoid deadlock, two virtual channels, VC1 and VC2 are designed to transmit seek packets. VC1 is used as the escape virtual channel and VC2 is the adaptive virtual channel. Only the XY routing is allowed in VC1. However, a packet can always use VC2. Any *seek* packet residing in VC2 has an opportunity to use VC1. By using this rule, only VC2 can be selected in bold links for the example shown in Fig. 7, while either VC1 or VC2 could be chosen on other links. It's obvious there is no deadlock in VC1. Hence, this mechanism is deadlock free [22]. The chosen virtual channel of each hop is also recorded in the *seek* packet.

Our baseline router is a speculative VC router [9]. Fig. 8 shows the router pipeline of seek packets. It replicates inside the router if multiple output ports are needed to find a healthy path. Particularly, two output ports are needed in our approach. We use asynchronous replication to eliminate the lock-step traversal. Specifically, the multicast *seek* packet is handled as multiple independent unicast packets in the virtual channel allocation (VA) and switch allocation (SA) stages, except that a *seek* flit is not removed from the input VC until all requested output ports are satisfied [23–25].

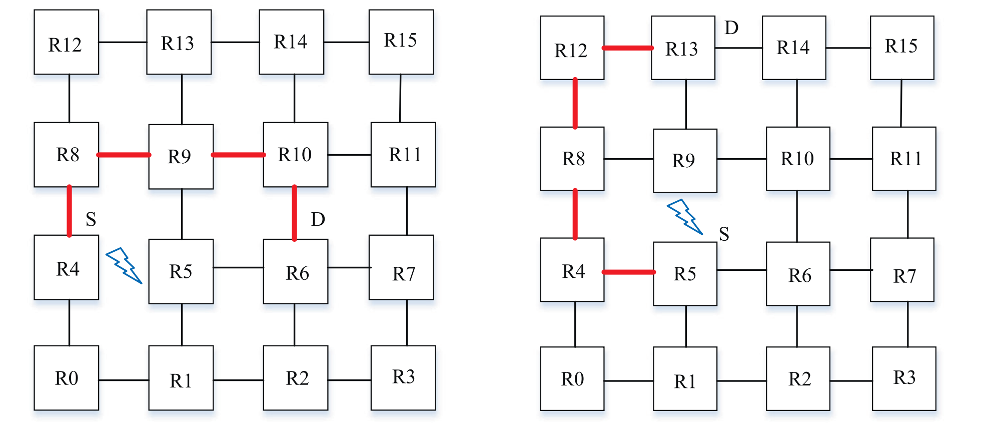
5.1.4. Backtrack and update route procedure

When the target node receives the first arrival seek packet, a backtrack procedure begins. The *track* packet is injected into the network with corresponding ports and virtual channels recorded in the seek packet. It uses the source routing algorithm, following the reverse path of the seek packet. The source node obtains a new fault-free path when receiving the track packet. Then it updates the route table, including selected ports and virtual channels, and transmits the data message using this route.

5.2. Route reconfiguration on special cases

Aiming at the network with one or two simultaneous faults, we directly reconfigure the route table in some special cases.

(1) Single-fault case
Fig. 9 shows the single-fault cases where the source node and target node locate in the same row or same column. A fault-free path marked in bold lines is directly recorded in the route table. Rules of selecting the path and the method of avoiding deadlock are as follows. If the source node and the target node locate in the same row, we first select an accessible neighbor node in Y-dimension as an intermediate node and then use the XY routing algorithm to select



(a) Source node and target node locate in the same row. (b) Source node and target node locate in the same column.

Fig. 9. Special cases on a single fault network where we directly reconfigure route. Bold lines mark the new route.

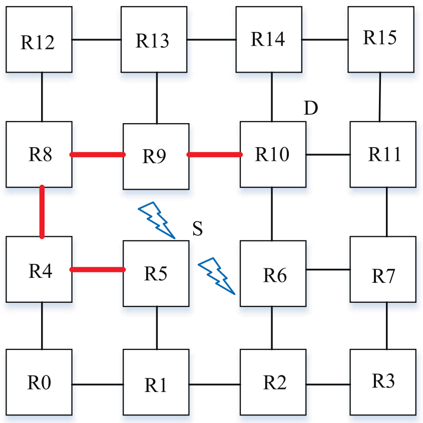


Fig. 10. Special cases on network with double faults where we could directly reconfigure route. Bold lines show an alternative route.

Table 3
Simulation configuration.

Characteristic	Value
Topology(mesh)	4 × 4
VC configuration	8VCs/port,8flits/VC
VC allocation	VC0,VC1: request and message VC2,VC3:acknowledgment VC4,VC5:seek packet VC6,VC7:track packet
Request message	uniform,transpose
Traffic pattern	bitcomp,neighbor,tornado
AVG	50cycle
k	2
Fault number	1
Injection rate	0.04

path from the intermediate node to the target node. However, if the source node and the target node locate in the same column, we select the X-dimension accessible neighbor node as the intermediate node and adopt the YX routing algorithm from this node to the target node. The selection of virtual channel is the same as the method described in Section 5.1.3 to avoid deadlock.

(2) Double-fault case
A special case is illustrated in Fig. 10 on the network with double faults. Two output links toward destination interrupt. We first select an accessible neighbor node in X-dimension (R4) and then use the YX routing algorithm to select a path from this node to the target node. The method of avoiding deadlock is the same as that in the single-fault case. Bold lines in Fig. 10 show an alternative fault-free path.

6. Evaluation

6.1. Performance evaluation

To evaluate our approach, we modify the cycle-accurate booksim simulator [9] to support our router pipeline described in Section 5.1.3 and the network interface design described in Section 4.2.

Table 3 summarizes basic configurations in the latency comparison. Eight VCs are designed to avoid the protocol-level and network-level deadlock. Two of them are used for requests and data messages, which are injected based on synthetic traffic patterns [9], including uniform random, transpose, bit complement, neighbor and tornado. The VC2 and VC3 are specialized for acknowledgment packets which are injected to the network when the target node receives the request packet. *Seek* packets use the VC4 and VC5, one is the escape VC and the other one is the adaptive VC, to avoid the network-level deadlock described in Section 5.1.3). According to our design, track packets and data messages use the source routing algorithm. The virtual channel is selected statically according to payloads of the seek packet and the route table respectively. We test the sum of the request and acknowledgement latencies under synthetic traffic patterns to obtain the average network latency *AVG*. And we set a constant K related to real applications to be 2.

In our experiment, a new fault-free path is fetched from the route table when the originally distributed XY routing algorithms cannot offer a healthy path. In order to compare our approach with the basic flooding algorithm which broadcasts seek packets throughout the entire network, we also implement Wachter et al.'s [13] work. Fig. 11 shows the latency of seek packets. Compared with the basic flooding algorithm, our approach reduces the latency under different traffic patterns. Specifically, the region flooding algorithm presents the largest latency decrease as 25% under the bit complement traffic pattern. However, under the transpose traffic pattern, different seek methods have the minimum latency gap which is only 9%.

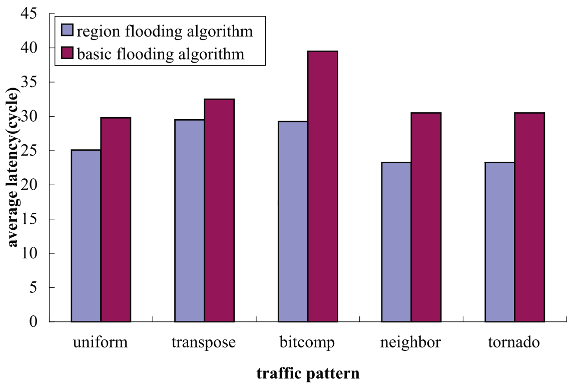


Fig. 11. The average latency of seek packets between region flooding algorithm and basic flooding algorithm under different traffic patterns.

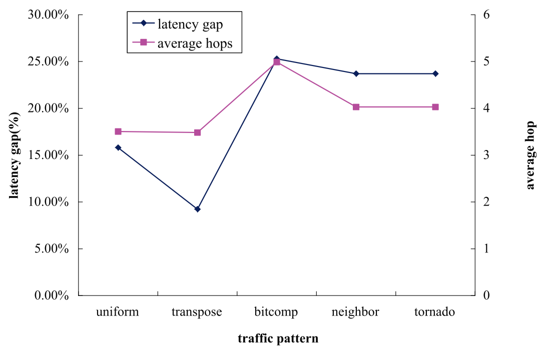


Fig. 12. The latency gap and average hop under different traffic patterns.

In order to explain this phenomenon, we analyze the relationship between the latency gap and the average hop under different traffic patterns. As shown in Fig. 12, the latency gap generally changes with the average hop. The latency gap accumulates during each hop of transmission. Specifically, the region flooding algorithm mainly applies two output ports to deliver packets. It costs at least two cycles in the switch allocation *(SA)* phase. However, the basic flooding algorithm needs at least three cycles in the *SA* phase. Hence, large average hops lead to large latency gaps. The bit complement traffic pattern has large average hops, so it leads to largest latency gaps. In comparison, the transpose traffic pattern obtains minimum latency gap because it has least average hops.

6.2. Fault tolerance evaluation

Compared with Wachter et al.'s [13] design, our approach reduces the latency, but at the expense of fault tolerance. We define fault tolerance as the percentage of scenarios where a healthy path can be found by using a certain seek method.

First, we evaluate the effect of the permanent fault rates on fault tolerance. We conduct 10,000 simulations at each fault rates on different size networks. One simulation randomly sets the source node and the target node as well as faulty links based on fault rates. The experiment is performed to compare fault tolerance of our approach with the basic flooding algorithm in a 4 × 4 2D mesh network.

Fig. 13 shows the results; the X-axis is the fault rate of links and the Y-axis is the fault tolerance. We observe that under the low fault rates such as 2% or 4%, our approach obtains almost the same fault tolerance as the basic flooding algorithm, approximately 98%. In other words, our approach dramatically decreases latencies and almost has no influence on fault tolerance. However, as fault rates increase, the fault tolerance of our method drops quickly compared

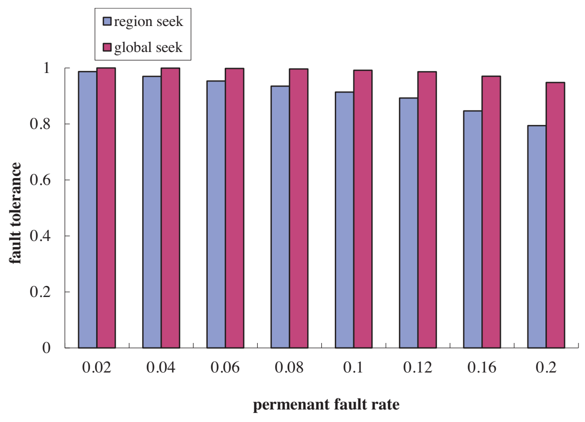


Fig. 13. Fault tolerance comparison of region flooding algorithm and basic flooding algorithm in a 4 × 4 mesh network.

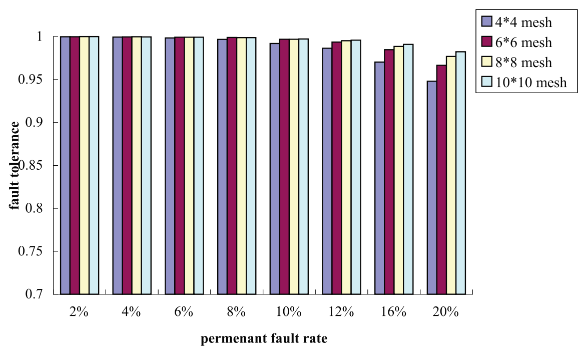
with that of the basic flooding algorithm which generally levels off. Hence, our approach is especially adequate to low fault rate cases.

Then we discuss the effect of network sizes on fault tolerance. Fig. 14 illustrates how fault tolerance changes with fault rates under different network sizes by using the region flooding algorithm as well as the basic flooding algorithm. They present an opposite trend of fault tolerance under high fault rate cases (over 12%). With the increase of network size, fault tolerance of our approach drops while the basic flooding algorithm increases. In other words, fault tolerance of the region flooding algorithm reveals a poor scalability in high fault rate cases. However, it generally remains stable when the fault rate of network is less than 10%.

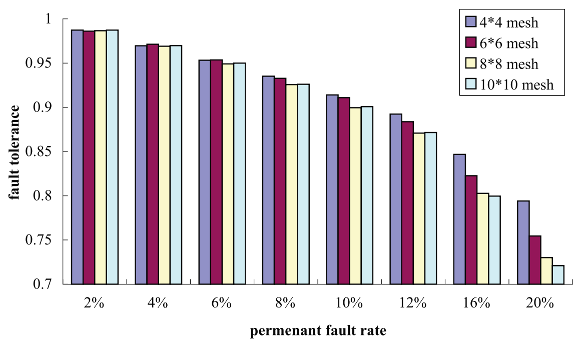
So, our approach can be applied to all sizes of mesh networks under low fault rate occasions while only be applied to the small network when the fault rate exceeds 12%. From the discussion about latencies, we know that latency gap of seek packets using our approach and the basic flooding algorithm is proportional to average hops. Hence, the region flooding algorithm presents a better superiority in large size networks with low fault rates.

6.3. Area and power analysis

To evaluate the hardware overhead, we implement our work as well as Watcher et al.'s [13] by using Verilog HDL. Both designs are synthesized in NangateOpenCell 45 nm library under 1.0 V and 0.5 GHz. The area and power consumption are calculated by Synopsys Design Compiler. We also compare these two kinds of fault-tolerant designs with the basic router which does not support MPI-like communication protocol and fault tolerance. For the sake of fairness, all the routers in the comparison are equipped with the same amount of buffers per port. Table 4 shows the configuration information. *MPF* unit refers to the components in the NI which are related to MPI-like communication and fault tolerance. The main difference between our work and Watcher et al.'s [13] work is the width of the routing table. In a 4 × 4 mesh network, the maximum hop of a fault-free path in our design is 8. However, the maximum hop in basic flooding algorithms is 16. Two bits are needed to describe the output port of each hop. Fig. 15 compares the area cost. Comparing with the basic router, the extra area cost of our design is 10%. Most extra consumption comes from the *MPF* unit because we need a lot of buffers to support our fault-tolerant MPI-like communication protocol. However, the area cost of our design is 5.2% lower than Watcher et al's [13] design in the *MPF* part and 1% lower in the router part. The decrease of the width of routing table is one of the reasons. Another one is that in Watcher et al.'s [13] design, the information of each hop in the delivery of *seek* packets is recorded in the router while our design decreases



(a) Fault tolerance of basic flooding method under different network size.



(b) fault tolerance of region flooding algorithm method under different network size.

Fig. 14. Fault tolerance comparison under different network sizes.

Table 4
Hardware evaluation configuration.

		Basic router	Our design	Watcher et al.'s design
Router configuration	VCs per port	8VCs	8VCs	8VCs
	Buffers per VC	8 buffers	8 buffers	8 buffers
	flit	128 bits	128 bits	128 bits
MPF configuration	Maximum message size		512 bits	512 bits
	Message buffer size		8 buffers	8 buffers
	SST width		16 bits	16 bits
	Routing table width		16 bits	32 bits

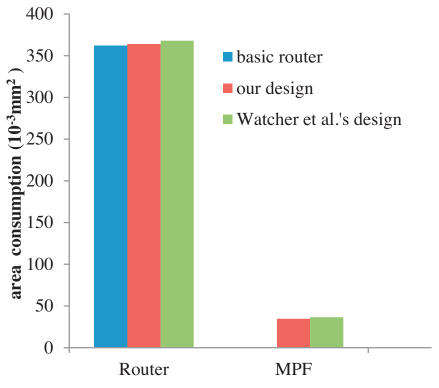


Fig. 15. Area consumption comparison.

the area cost of this part by recording to the flit. As for the power consumption, our design is generally the same as Watcher et al.'s [13], which is 12% higher than the basic router.

6.4. Discussion

Our experiment on the fault tolerance and performance evaluation are conducted through simulations. In order to obtain results on the real efficiency of our proposed algorithm, we aim to implement our design in FPGA and inject physical faults in the real system in the future. Then we can observe the error correction abilities and latency influence on real applications. These deep works can make our design more practical.

7. Conclusion

In order to implement a reliable NoC system, this paper proposes a fault-tolerant MPI-like communication protocol. It detects the link failure if there exist unresponsive requests and automatically starts the new path exploration at runtime. The region flooding algorithm is utilized to find a fault-free path. It improves performance by restricting routes to be minimal. Simulation results show that our approach dramatically reduces the latency. The latency decrease is generally proportional to the average hop. Also, it causes only 2% fault tolerance loss for low fault rate cases. What's more, the real hardware consumption is low. Our fault-tolerant communication protocol and region flooding algorithm can be easily extended to other NoC architecture.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61572508, No.61272144, No. 61303065), Doctoral Fund of Ministry of Education (No. 20134307120028), Hunan Provincial Natural Science Foundation (No. 14JJ3002) and National High Technology Research and Development Program of China (No. 2012AA010905).

References

[1] H. Kim, A. Vitkovskiy, P.V. Gratz, V. Soteriou, Use it or lose it: wear-out and lifetime in future chip multiprocessors, in: Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 2013, pp. 136–147.
[2] I. Koren, Z. Koren, Defect tolerance in vlsi circuits: techniques and yield analysis, Proc. IEEE 86 (9) (1998) 1819–1838, doi:10.1109/5.705525.
[3] S. Makar, T. Altinis, N. Patkar, J. Wu, Testing of vega2, a chip multi-processor with spare processors, in: Test Conference, 2007. ITC 2007. IEEE International, IEEE, 2007, pp. 1–10, doi:10.1109/TEST.2007.4437584.
[4] S. Shamshiri, K.-T. Cheng, Yield and cost analysis of a reliable noc, in: VLSI Test Symposium, 2009. VTS '09. 27th IEEE, 2009, pp. 173–178, doi:10.1109/VTS.2009.34.
[5] M. Pirretti, G.M. Link, R.R. Brooks, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, Fault tolerant algorithms for network-on-chip interconnect, in: IEEE Computer Society Annual Symposium on VLSI, 2004. Proceedings, IEEE, 2004, pp. 46–51, doi:10.1109/ISVLSI.2004.1339507.
[6] A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, Nocalert: an on-line and real-time fault detection mechanism for network-on-chip architectures, in: 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2012, IEEE, 2012, pp. 60–71, doi:10.1109/MICRO.2012.15.

[7] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, D. Blaauw, A highly resilient routing algorithm for fault-tolerant nocs, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2009, pp. 21–26, doi:10.1109/DATE.2009.5090627.

[8] T. Lehtonen, P. Liljeberg, J. Plosila, Self-timed noc links using combinations of fault tolerance methods, in: Proceedings IEEE Design Automation and Test in Europe (DATE), 2007.

[9] W.J. Dally, B.P. Towles, Principles and Practices of Interconnection Networks, Elsevier, 2004.

[10] A.A. Chien, J.H. Kim, Planar-adaptive routing: low-cost adaptive networks for multiprocessors, J. ACM (JACM) 42 (1) (1995) 91–123, doi:10.1145/200836.200856.

[11] C.-C. Su, K.G. Shin, Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes, IEEE Trans. Comput. 45 (6) (1996) 666–683, doi:10.1109/12.506423.

[12] S.-P. Kim, T. Han, Fault-tolerant wormhole routing in mesh with overlapped solid fault regions, Parallel Comput. 23 (13) (1997) 1937–1962, doi:10.1016/S0167-8191(97)00093-8.

[13] E. Wachter, A. Erichsen, L. Juracy, A. Amory, F. Moraes, Runtime fault recovery protocol for noc-based mpsocs, in: International Symposium on Quality Electronic Design (ISQED), 2014, pp. 132–139, doi:10.1109/ISQED.2014.6783316.

[14] E. Wachter, A. Erichsen, A. Amory, F. Moraes, Topology-agnostic fault-tolerant noc routing method, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, 2013, pp. 1595–1600, doi:10.7873/DATE.2013.324.

[15] T. Dumitraş, S. Kerner, R. Mărculescu, Towards on-chip fault-tolerant communication, in: Proceedings of the 2003 Asia and South Pacific Design Automation Conference, ACM, 2003, pp. 225–232.

[16] C. Iordanou, V. Soteriou, K. Aisopos, Hermes: architecting a top-performing fault-tolerant routing algorithm for networks-on-chips, in: 32nd IEEE International Conference on Computer Design (ICCD), 2014, IEEE, 2014, pp. 424–431, doi:10.1109/ICCD.2014.6974715.

[17] K. Aisopos, A. DeOrio, L.-S. Peh, V. Bertacco, Ariadne: agnostic reconfiguration in a disconnected network environment, in: International Conference on Parallel Architectures and Compilation Techniques (PACT), 2011, IEEE, 2011, pp. 298–309, doi:10.1109/PACT.2011.61.

[18] P. Mahr, C. Lorchner, H. Ishebabi, C. Bobda, Soc-mpi: a flexible message passing library for multiprocessor systems-on-chips, in: International Conference on Reconfigurable Computing and FPGAs, 2008. ReConFig'08, IEEE, 2008, pp. 187–192, doi:10.1109/ReConFig.2008.27.

[19] F. Fu, S. Sun, X. Hu, J. Song, J. Wang, M. Yu, Mmpi: a flexible and efficient multiprocessor message passing interface for noc-based mpsoc, in: IEEE International SOC Conference (SOCC), 2010, IEEE, 2010, pp. 359–362, doi:10.1109/SOCC.2010.5784695.

[20] R. Batchu, Y.S. Dandass, A. Skjellum, M. Beddhu, Mpi/ft: a model-based approach to low-overhead fault tolerant message-passing middleware, Cluster Comput. 7 (4) (2004) 303–315.

[21] R.T. Aulwes, D.J. Daniel, N.N. Desai, R.L. Graham, L.D. Risinger, M.A. Taylor, T.S. Woodall, M.W. Sukalski, Architecture of la-mpi, a network-fault-tolerant mpi, in: Proceedings. 18th International Parallel and Distributed Processing Symposium, 2004, IEEE, 2004, p. 15, doi:10.1109/IPDPS.2004.1302920.

[22] J. Duato, A new theory of deadlock-free adaptive routing in wormhole networks, IEEE Trans. Parallel Distrib. Syst. 4 (12) (1993) 1320–1331, doi:10.1109/71.250114.

[23] Y.H. Kang, J. Sondeen, J. Draper, Multicast routing with dynamic packet fragmentation, in: Proceedings of the 19th ACM Great Lakes symposium on VLSI, ACM, 2009, pp. 113–116.

[24] L. Wang, Y. Jin, H. Kim, E.J. Kim, Recursive partitioning multicast: a bandwidth-efficient routing for networks-on-chip, in: Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip, IEEE Computer Society, 2009, pp. 64–73, doi:10.1109/NOCS.2009.5071446.

[25] S. Ma, N.E. Jerger, Z. Wang, Supporting efficient collective communication in nocs, in: High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on, IEEE, 2012, pp. 1–12, doi:10.1109/HPCA.2012.6168953.

Lu Wang received the B.S. degree in computer science and technology from the National University of Defense Technology (NUDT) in 2015. She is currently a first year Ph.D.student in the School of Computer, NUDT. Her research interests include Network on-chip and reliable system designs

Sheng Ma received the B.S. and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT) in 2007 and 2012, respectively. He visited the University of Toronto from Sept. 2010 to Sept. 2012. He is currently an Assistant Professor of the School of Computer, NUDT. His research interests include on-chip networks, SIMD architectures and arithmetic unit designs.

Zhiying Wang received the Ph.D. degree in electrical engineering from the National University of Defense Technology in 1988. He is currently a Professor with School of Computer, NUDT. He has contributed over 10 invited chapters to book volumes, published 240 papers in archival journals and refereed conference proceedings, and delivered over 30 keynotes. His main research fields include computer architecture, computer security, VLSI design, reliable architecture, multicore memory system and asynchronous circuit. He is a member of the IEEE and ACM.