ELSEVIER

Parallel Computing 26 (2000) 455–465

PARALLEL COMPUTING

www.elsevier.com/locate/parco

A fault-tolerant routing algorithm for wormhole routed meshes

Pao-Hwa Sui, Sheng-De Wang*

Department of Electrical Engineering, National Taiwan University, Room 441, EE Building, 1 Roosevelt Rd., Sec. 4, Taipei 106, Taiwan, ROC

Received 1 November 1996; received in revised form 22 October 1999

Abstract

We investigate fault-tolerant routing schemes which aim at using low number of virtual channels in wormhole-routed mesh networks. The faults under consideration are rectangular block faults, which are suitable for modeling faults on board level in networks with grid structures. There is no restriction on the number of faults. The concepts of *f*-ring and *f*-chain are used in our scheme. Messages are routed minimally when not blocked by faults and are routed along the boundaries of the faults encountered. Only three virtual channels and local knowledge of faults are required for our routing scheme to be correct, deadlock- and livelock-free. By allocating virtual channels to messages carefully, all virtual channels have the potential to be used by messages; hence, none of the virtual channels and its associated hardware is wasted. © 2000 Published by Elsevier Science B.V. All rights reserved.

Keywords: Wormhole routing; Virtual channel; Fault-tolerant; Deadlock-free; Mesh networks

1. Introduction

Direct networks have become a popular means for interconnecting components of massively parallel computer systems. In direct networks, nodes (computers) are connected to only a few nodes, its neighbors, according to the topology of the network and communicate with each other by passing messages. The *n*-dimensional mesh network is currently the most popular topology for massively parallel com-

* Corresponding author. Tel.: +886-2-23635251; fax: +886-2-23671909.
E-mail address: sdwang@cc.ee.ntu.edu.tw (S.-D. Wang).

puter systems. Low dimensional mesh networks, due to its low node degree, are more popular than the high dimensional mesh networks. The two-dimensional mesh topology has been adopted by Symult 2010 [1], Intel Touchstone DELTA [2] and Intel paragon; the MIT J-machine adopts three-dimensional mesh topology.

The wormhole switching technique by Dally and Seitz [5] has been widely used in the latest generation of direct networks for switching messages. In the wormhole technique, a message is divided into packets and a packet is composed of flow control digits or *flits*. The header flit governs the route. As the header advances along a specific route, the remaining flits follow in a pipeline fashion. If the header encounters a busy channel, it is blocked until the channel becomes available and all the flits in same packet remain in the flit buffer along the specified route. A survey of wormhole routing for direct networks can be found in [6]. The characteristics of pipelining and buffering flits in flit buffers when header blocked by busy channel make the wormhole switching more deadlock-prone than *virtual-cut-through* [3] and *store-and-forward* [4] switching techniques.

A massively parallel computer system cannot avoid having failure components in real world. A realistic data communication scheme should have the capability of fault tolerance. Several fault-tolerant communication schemes for direct networks have been proposed in recent years [7–17] Simulating multiple virtual channels on each physical channel and enforcing an order on allocation of virtual channels to messages have been used by many schemes for avoiding deadlock. Large number of virtual channels cause more hardware complexity, cost and delay in routing logic. In this paper, we address the issue of designing fault-tolerant routing schemes for wormhole routed mesh networks with small number of virtual channels. The faults under consideration are rectangular block faults, which are suitable for modeling faults on board level in networks with grid structure, since nowadays it is often to place multiple nodes on a printed circuit board. Only local knowledge of faults are required in our schemes.

Works by Chien and Kim [12] and Boppona and Chalasani [16] are two most similar studies to our works. Chien and Kim present a partially adaptive algorithm for mesh networks. Only three virtual channels are required in their methods. However, their methods need to deactivate extra nodes when faults are located on the boundaries of meshes. For instance, even only one node or link fault on a boundary row of a 2D mesh, all nodes on that row must be deactivated. In fact, all faults need to be augmented to form rectangular faults, convex in [12], to assure the correctness of their methods. Boppana and Chalasani consider arbitrary-located rectangular faults. The concepts of *f*-rings and *f*-chains are introduced and are used for routing messages around rectangular faults. Two virtual channels are required to provide non-adaptive deadlock-free routing in networks with non-overlapping *f*-rings. For more complex faults, such as overlapping *f*-rings and *f*-chains, four virtual channels are used. Although Boppana and Chalasani claim that their methods provide deadlock-free message routing in 2D meshes with complex faults, deadlocks among column messages may occur in some cases [18]. A deadlock example is given in Appendix A. In contrast, our methods can handle any combination of arbitrary-located rectangular block faults and assure deadlock-free property with only three virtual channels. Faults on boundary cause no deactivation in our scheme. When there is no restriction on the number of arbitrary-located faults, three virtual channels are the minimum number, to the best of our knowledge, of virtual channels used in fault-tolerant routing methods for meshes. The rest of this paper is organized as follows. Section 2 describes the fault patterns under consideration. In Section 3, a fault-tolerant routing algorithm is presented for 2D meshes with multiple rectangular block faults. Deadlock-free proof of the proposed algorithm is also presented in this section. In Section 4, we extend our routing techniques for 2D meshes to *n*D meshes. A conclusion is given in Section 5.

2. Preliminary

An *n*-dimensional mesh has $k_{n-1}k_{n-2}\cdots k_0$ nodes, k_i nodes along dimension i, $0 \leqslant i \leqslant n-1$, and $k_i \geqslant 2$. Each node x is uniquely indexed by an *n*-tuple $(x_{n-1}, x_{n-2}, \ldots, x_0)$, where $0 \leqslant x_i \leqslant k_i - 1$. Two nodes $x = (x_{n-1}, x_{n-2}, \ldots, x_0)$ and $y = (y_{n-1}, y_{n-2}, \ldots, y_0)$ are neighbors if and only if $x_i = y_i$ for all i except one, j, where $x_j = y_j \pm 1$. Each node has from n to $2n$ neighbors up to their location on the mesh. Neighboring nodes are connected by direct link implemented by two unidirectional physical channels with opposite directions. The link between nodes x and y is denoted $\langle x,y \rangle$.

The four sides of a 2D mesh, hereinafter, labeled as North, East, South and West. A faulty block, which is a set of faulty nodes and/or links and does not contain an entire row or column failure, is a rectangular block fault if: (a) the boundary of the faulty block forms a rectangle and contains only fault-free nodes and links and (b) all components within this rectangle are faulty. An entire row or column failure makes the meshes disconnected and, therefore, is not allowed. The definition of rectangular block faults and some other definitions and terminology used in this paper are the same to those in [16]. The boundary of a rectangular block fault is of rectangle shape and is called *f-ring* of the fault. Block faults that contain boundary faults are rectangular block faults if the above definition is satisfied when the mesh is extended with fault-free nodes on all four sides and the boundary of the faulty components is called *f-chain* of the faulty block. By exchanging link status to non-faulty neighbors, each node can know easily its position on an *f*-ring. The nodes at which *f*-chain touches the boundaries of the mesh are the end nodes of the *f*-chains. Links incident to faulty nodes are considered faulty. Fig. 1 shows three rectangular faults: $F1 = \{(5,2),(6,2)\}$, $F2 = \{\langle(2,0)(3,0)\rangle, \langle(2,1),(3,1)\rangle, \langle(2,2),(3,2)\rangle, \langle(2,3),(3,3)\rangle, \langle(2,4),(3,4)\rangle\}$ and $F3 = \{\langle(1,5),(2,5)\rangle, \langle(1,6),(2,6)\rangle, \langle(1,7),(2,7)\rangle\}$ on a 2D mesh. Faulty components are not shown in Fig. 1. Nodes (2,0), (3,0) and (1,7), (2,7) are end nodes of F2 and F3, respectively. Just like most of the fault-tolerant routing literature, messages are assumed to be destined only to fault-free nodes and meshes are connected under faults in this paper.

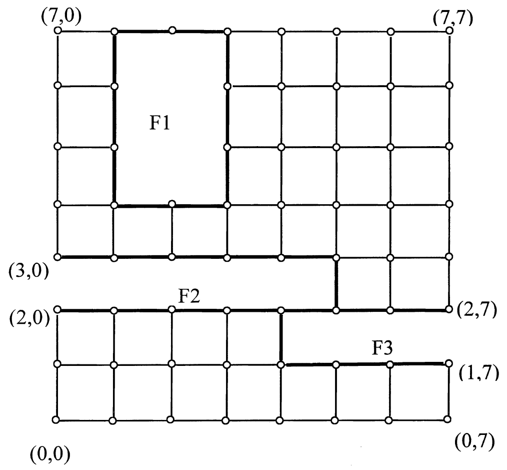

Fig. 1. Example of three rectangular block faults and the corresponding *f*-ring and *f*-chains (indicated by heavy lines) in a 2D mesh.

3. Fault-tolerant routing in 2D meshes

In this section, a fault-tolerant routing algorithm, *MESH2D*, for 2D meshes with multiple arbitrary-located rectangular block faults, is proposed. The *f*-rings and *f*-chains of these rectangular faults could be overlapped or not. The fault-tolerant methods used in this section will be generalized for *n*D meshes in Section 5. In the following, row hops and column hops are corresponded to hops in dimension 0 and 1, respectively.

Definition 1. The first hop on the path, which is specified by the *e*-cube algorithm for a message M from node x to node y, is called the *e*-cube hop of message M at node x.

Definition 2. A message that has one or more row hops remaining is called a row message. A message that needs to travel only in a column to reach its destination is called a column message.

Definition 2 is cited from [16]. Row messages that travel from west to east (east to west) are WE (EW) messages. Column messages are classified into NS and SN messages by similar method. The well-known *e*-cube algorithm is used as a basis in our scheme. Messages are routed on 2D meshes in the order of dimension 0 and 1. Therefore, in *MESH2D*, a row message may change into a column message, but a column message never becomes a row message.

3.1. The MESH2D algorithm

Each message is injected into the network as a row message and its direction is set to null. Messages are routed along their *e*-cube hop if not being blocked by faults. When faults are encountered, depending upon the message type and the relative position of the destination nodes to the source nodes, the direction of messages are set to clockwise or counter-clockwise (steps 6–9 in procedure set-direction). Messages are routed on *f*-rings or *f*-chains according to the specified directions. The node at which a column message is blocked by a fault is recorded at (x_1, x_0) and this information will be used for deciding when the direction of a column message can be reset to null (steps 8, 9 and 3 in procedure set-direction). The purpose of step 0 (in procedure set-direction) is to make sure that the value of (x_1, x_0) is always defined when it is used. When an end point of *f*-chain is reached, messages take a *u*-turn and their directions are reversed (step 1 in procedure set-direction).

Algorithm *MESH2D*

```
/* message M is currently resided at node $(c_1, c_0)$, the source and the destination node
is $(s_1, s_0)$ and $(d_1, d_0)$, respectively*/
0. If $(c_1, c_0) = (s_1, s_0)$, then begin
   (1) set message type to WE, if $c_0 \leqq d_0$, or EW, if $c_0 > d_0$, and
   (2) set direction of M to null, end.
1. If $(c_1, c_0) = (d_1, d_0)$, consume M and return.
2. If M is a row message and $c_0 = d_0$ then change its type to
      NS if $c_1 > d_1$, or
      SN if $c_1 < d_1$.
3. Set-direction(M).
4. If the direction of M is null, then route M along its e-cube hop, else route M on
   the f-ring or f-chain according to the specified direction.
```

Procedure set-direction(M)

```
0. If M is a column message and its direction is null, then set $(x_1, x_0) = (c_1, c_0)$.
1. If the direction of M is not null and the current node is an end node then reverse
   the direction of M and return.
2. If M is a column message and $c_0 \neq x_0$, then return.
3. If M is a column message and $c_1 \neq x_1$, $c_0 = x_0$, then set its direction to null.
4. If the next e-cube hop of M is not faulty, set its direction to null and return.
5. If direction of M is not null, then return.
6. If M is a WE message, set its direction to
   6.1. clockwise if $c_1 < d_1$, or
   6.2. counter-clockwise if $c_1 > d_1$, or
   6.3. either direction if $c_1 = d_1$.
7. If M is an EW message, set its direction to
   7.1. clockwise if $c_1 > d_1$, or
   7.2. counter-clockwise if $c_1 < d_1$, or
   7.3. either direction if $c_1 = d_1$.
```

8. If M is an NS message, set its direction to counter-clockwise, if the current node is not located on the WEST boundary of 2D meshes, or clockwise, otherwise, and set $(x_1, x_0) = (c_1, c_0)$.
9. If M is an SN message, set its direction to clockwise, if the current node is not located on the WEST boundary of 2D meshes, or counter-clockwise, otherwise, and set $(x_1, x_0) = (c_1, c_0)$.

3.2. Usage of virtual channels

Virtual channels of class i are denoted as hc_i and vc_i, $0 \leqslant i \leqslant 2$, if they are on dimension 0 and 1, respectively. Virtual channels hc_0 are further divided into two disjoint subclasses hc_0^+, which are of direction from west to east, and hc_0^-, which are of direction from east to west. That is, virtual channels hc_0^+ (hc_0^-) are from nodes with smaller (larger) address to nodes with larger (smaller) address in dimension 0. Virtual channels hc_1 (hc_2) that are on the southern (northern) side of f-chains, which have end nodes located on the WEST boundary of 2D meshes, and have direction from west to east are denoted as hc_1^b (hc_2^b). All other hc_1 (hc_2) are denoted as hc_1^a (hc_2^a). Virtual channels vc_i, $0 \leqslant i \leqslant 2$, are also further classified into two disjoint subclasses vc_i^+ and vc_i^-. Virtual channels vc_i^+ (vc_i^-) are from nodes with smaller (larger) addresses to nodes with larger (smaller) addresses in dimension 1.

In our method, WE messages use hc_0^+, vc_1^- and vc_2^+ as they travel in the corresponding directions, EW messages use hc_0^-, vc_1^+ and vc_2^- as they travel in the cor-

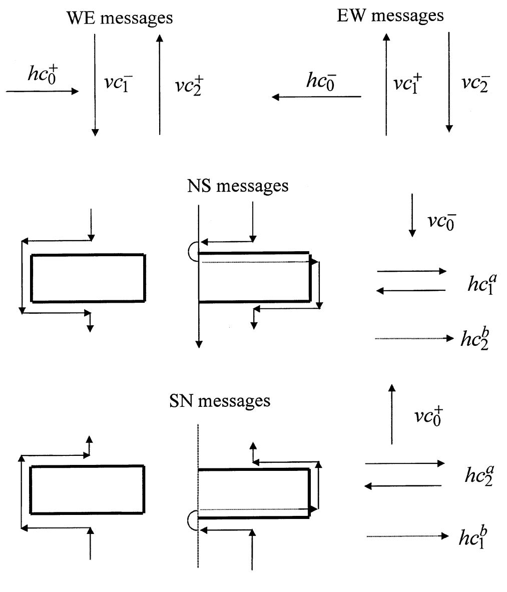

Fig. 2. Usage of virtual channels.

responding directions, NS messages use vc_0^-, hc_1^a and hc_2^b and SN messages use vc_0^+, hc_2^a and hc_1^b. The usage of virtual channels are depicted in Fig. 2. No SN (NS) messages can use virtual channels hc_2^b (hc_1^b), for it is impossible for any SN (NS) message to get to the northern (southern) side of an *f*-chain, that has end nodes located on the WEST boundary of 2D meshes, in direction of clockwise (counter-clockwise). In fact, these four types of messages use disjoint sets of virtual channels. From Fig. 2, it is clear that all virtual channels in each direction of each class are allocated to one or two message type. None of the virtual channels and its associated hardware are wasted.

3.3. Example

Let us consider the example of routing message M from (5, 0) to (1, 2) in Fig. 1. The path taken by M and the step number of the path are shown in Fig. 3. M is routed as a WE message from (5, 0) to (5, 1). At (5, 1), its next *e*-cube hop is faulty and its direction is set to counter-clockwise, since the destination node is in a row below (5, 1). At (4, 1), its direction is reset to null and M is routed along its *e*-cube hop to (4, 2). At (4, 2), M becomes an NS message and travels from (4, 2) to (3, 2). At (3, 2), due to its next *e*-cube hop is faulty, M travels in the counter-clockwise direction to (3, 0). At (3, 0), M takes a *u*-turn and its direction is reversed to clockwise, since an end node is encountered. M travels along the *f*-chain of F2 in the clockwise direction from (3, 0) to (2, 2). Direction of M is reset to null again at (2, 2) and M is routed along its *e*-cube hop to destination node (1, 2). Steps 1 and 3 of the path use virtual channels hc_0^+, step 2 uses vc_1^-, steps 4, 12 and 16 use vc_0^-, steps 5, 6, and 13–15 use hc_1^a and steps 7–11 use hc_2^b.

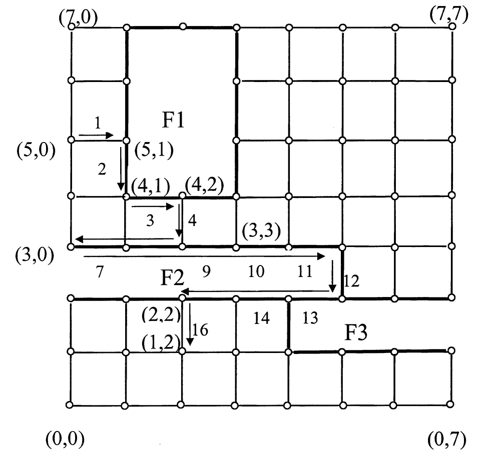

Fig. 3. Example of routing message from (5,0) to (1,2).

3.4. Deadlock-freeness of algorithm MESH2D

Since (a) EW, WE, SN and NS messages use disjoint sets of virtual channels and (b) row messages (EW and WE) can become column messages (NS and SN), but column messages cannot change into row messages and (c) EW and WE messages cannot change into each other and NS and SN cannot change into each other, algorithm *MESH2D* is a deadlock-free routing method if no deadlock can occur in each of the four types of messages.

Theorem 1. *The MESH2D algorithm is a deadlock-free routing method for 2D meshes with multiple rectangular block faults.*

Proof. WE messages can travel from west to east but not from east to west, there cannot be a deadlock among WE messages waiting in different columns. A WE message can travel from north to south or south to north, if its next *e*-cube hop is faulty. A north-to-south (south-to-north) WE message can take south-to-north (north-to-south) hops only if it encounters an end node and takes a *u*-turn at the end node. For a deadlock to occur among WE messages waiting in same column, all west to east channels that might be the next *e*-cube hop for these waiting WE messages must be faulty, that is, the entire column to the east of these waiting WE messages is faulty. An entire column fault disconnects meshes, which is contrary to our assumption. No deadlock occur among EW messages can be assured by similar statements. NS messages can travel from north to south but not from south to north, there cannot be a deadlock among NS messages waiting in different rows. NS messages are designed to get around the faulty components in counter-clockwise direction. An NS message can take a *u*-turn at an end node located on the WEST boundary of 2D meshes and change its direction to clockwise, but cannot take a *u*-turn at the EAST boundary of 2D meshes, since no entire row of faulty components is allowed. Thus, no deadlock can occur among NS messages waiting on the same row. We assure that no deadlock can occur among SN messages by similar statements. □

Row messages are routed on dimension 0 towards their destinations as long as their next *e*-cube hop are not faulty. Since there is no whole-column failure, each row message can always get to the column on which its destination resides. A column message can always route around faults encountered, for there is no whole-row failure. Therefore, by *MESH2D*, every message can get to its destination. Since the number of faults is finite and message never visits a fault more than once, our routing scheme is also livelock-free.

4. Fault-tolerant routing in *n*D meshes

An *n*D mesh can be decomposed into many 2D submeshes by removing links on all dimensions but the two dimensions on which these 2D submeshes are formed. In this paper, an nD mesh with multiple rectangular block faults is in the sense that to any 2D submesh of the nD mesh, every fault is confined to a rectangular block fault. By combining the *planar-adaptive routing* (PAR) [12] techniques with our routing method for 2D meshes, algorithms, which can route messages in nD meshes with multiple rectangular block faults using three virtual channels, can be designed easily. In the following, we focus on the issue, which is also the main problem of using PAR techniques, of how to divide virtual channels into disjoint subclasses for messages usage in successive 2D submeshes.

Virtual channels of class j, $0 \leqslant j \leqslant 2$, on dimension i, $0 \leqslant i \leqslant n-1$, are denoted as c_{ij}. Virtual channels c_{ij}, except the virtual channels of class 1 and 2 on dimension 0, that are from nodes with smaller (larger) addresses to nodes with larger (smaller) addresses in dimension i are denoted as c_{ij}^{+} (c_{ij}^{-}). Virtual channels of class 1 (2) on dimension 0 are partitioned, in the same way of partitioning virtual channels hc_1 (hc_2) in algorithm *MESH2D*, into two disjoint subclasses, denoted as C_{01}^{a} and C_{01}^{b} $(c_{02}^{a}$ and $c_{02}^{b})$. We define n routing planes, A_0 to A_{n-1}, as the combination of the virtual channels:

$$A_i = c_{i0} + c_{(i+1)1} + c_{(i+1)2} \quad (\text{for } 0 \leqslant i \leqslant n-2),$$

$$A_{n-1} = c_{(n-1)0} + c_{01} + c_{02}.$$

Each routing plane A_i is divided into two disjoint subplanes A_i^{+} and A_i^{-}.

$$A_i^{+} = c_{i0}^{+} + c_{(i+1)1}^{-} + c_{(i+1)2}^{+}, \quad A_i^{-} = c_{i0}^{-} + c_{(i+1)1}^{+} c_{(i+1)2}^{-} \qquad (\text{for } 0 \leqslant i \leqslant n-2),$$

$$A_{n-1}^{+} = c_{(n-1)0}^{+} + c_{02}^{a} + c_{01}^{b}, \quad A_{n-1}^{-} = c_{(n-1)0}^{-} + c_{01}^{a} + c_{02}^{b}.$$

Routing techniques for row and column messages in *MESH2D* are adopted for routing messages in A_0 to A_{n-2} and A_{n-1}, respectively. Messages that have destination address greater (smaller) than that of the source node in dimension i are routed in $A_i^{+}(A_i^{-})$. Routing in A_i reduces the distance between source and destination in dimension i to 0. After routing through A_0 to A_{n-1} seriously, messages have reached their destinations.

Since (a) messages are routed seriously through A_0 to A_{n-1} and (b) routing planes A_i and A_j, $i \neq j$, contain distinct set of virtual channels, and (c) messages routed in A_i^{+} and A_i^{-} do not change into each other and (d) no deadlock occur among messages routing in A_i^{+} or A_i^{-} (due to result of *MESH2D*), the proposed routing method for nD meshes are deadlock-free.

For virtual channel has its own control logic and flit buffer, the virtual channel utilization, μ, which is defined as the ratio of the total number of virtual channels allocated potentially to messages to the total number of virtual channels in networks, is used to compare the routing hardware utilization between [16] and this paper. In [16], virtual channels of each class with direction heading to three of the four boundaries in each 2D submesh are allocated to messages. Therefore, one-fourth of the virtual channels are not allocated to any messages, that is $\mu = 0.75$.

In our scheme, virtual channels of each class with direction heading to all four boundaries in each 2D submesh are allocated to massages, that is $\mu = 1.0$ in this paper. None of the virtual channels and its associated hardware is wasted in our scheme.

5. Conclusion

A fault-tolerant wormhole routing algorithm for 2D meshes with multiple faults is presented. The faults under consideration are rectangular block faults, which are suitable for modeling faults in networks with grid structures. Messages are routed minimally if not blocked by faults. By combining the PAR techniques with our routing methods for 2D meshes, fault-tolerant routing methods for *n*D meshes can be designed easily. Only three virtual channels and local knowledge of faults are required for our routing scheme to be correct, deadlock- and livelock-free. By allocating virtual channels to messages carefully, all virtual channels are fully utilized and, therefore, none of the virtual channels and its associated hardware are wasted. The number of virtual channels used in this paper is the minimal number known in fault-tolerant routing methods for mesh networks, when there is no restriction on the number of arbitrary-located faults.

Any routing algorithm for meshes can be enhanced for tolerating rectangular block faults by incorporating our routing schemes. For fully adaptive routing algorithms, routing schemes used by column messages only are enough to achieve the enhancement, for every message could only be blocked at node having the same address value in all dimensions but one to its destination in meshes with multiple rectangular faults.

Appendix A

In [18], NS messages use virtual channels of class 2 to get to their destinations. When a fault is encountered, the direction of the NS message can be set to either clockwise or counter-clockwise. Fig. 4 shows a deadlock example between two NS messages.

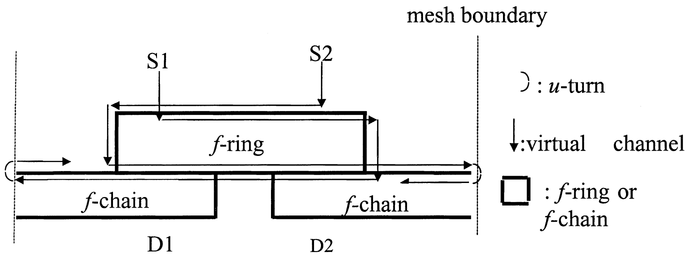

Fig. 4. A deadlock example of NS messages routed around *f*-regions with arbitrary orientation.

References

[1] C.L. Seitz, W.C. Athas, C.M. Flaig, A.J. Martin, J. Seizovic, C.S. Steele, W.K. Su, The architecture and programming of the Ametek Series 2010 multicomputer, in: Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications I, 1988, pp. 33–36.

[2] Intel Corp., A Touchstone DELTA System Description, 1991.

[3] P. Kermani, L. Kleinrock, Virtual cut-through: a new computer communication switching technique, Computer Networks 3 (1979) 267–286.

[4] K.D. Gunther, Prevention of deadlock in packet-switched data transport systems, IEEE Transactions on Communications 29 (1981) 512–524.

[5] W.J. Dally, C.L. Seitz, The torus routing chip, Journal of Distributed Computing 1 (3) (1986) 187–196.

[6] L.M. Ni, P.K. Mckinley, A survey of wormhole routing techniques in direct networks, IEEE Computer 26 (1993) 62–76.

[7] C.J. Glass, L.M. Ni, The turn model for adaptive routing, in: Proceedings of the 19th Annual International Symposium on Computer Architecture, 1992, pp. 278–287.

[8] C.J. Glass, L.M. Ni, Fault-tolerant wormhole routing in meshes, in: Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing, 1993, pp. 240–249.

[9] D.H. Linder, J.C. Harden, An adaptive and fault tolerant wormhole routing strategy for *k*-ary *n*-cubes, IEEE Transactions on Computers 40 (1) (1991) 2–12.

[10] M.S. Chen, K.G. Shin, Adaptive fault-tolerant routing in hypercube multicomputers, IEEE Transactions on Computers 39 (12) (1990) 1406–1416.

[11] M.S. Chen, K.G. Shin, Depth-first search approach for fault-tolerant routing in hypercube multicomputers, IEEE Transactions on Parallel and Distributed Systems 1 (2) (1990) 152–159.

[12] A.A. Chien, J.H. Kim, Planar-adaptive routing: low-cost adaptive networks for multi-processors, in: Proceedings of the 19th International Symposium on Computer Architecture, 1992, pp. 268–277.

[13] T.C. Lee, J.P. Hayes, A fault-tolerant communication scheme for hypercube computers, IEEE Transactions on Computers 41 (10) (1992) 1242–1256.

[14] R.V. Boppana, S. Chalasani, Fault-tolerant routing with non-adaptive wormhole algorithms in mesh networks, Supercomputing (1994) 693–702.

[15] S. Chalasani, R.V. Boppana, Adaptive fault-tolerant wormhole routing algorithms with low virtual channel requirements, in: Proceedings of the International Symposium on Parallel Architecture, Algorithms, and Networks, 1994, pp. 214–221.

[16] R.V. Boppana, S. Chalasani, Fault-tolerant wormhole routing algorithms for mesh networks, IEEE Transactions on Computers 44 (7) (1995) 848–864.

[17] G.M. Chiu, S.P. Wu, A fault-tolerant routing strategy in hypercube multicomputers, IEEE Transactions on Computers 45 (2) (1996) 143–155.

[18] P.H. Sui, S.D. Wang, Comments on "fault-tolerant wormhole routing algorithms for mesh networks", Technique Report NTUEE-TR-96-001, Department of Electrical Engineering, National Taiwan University, July 1996.