Fault-tolerant wormhole routing algorithm for mesh networks

P.-H.Sui and S.-D.Wang

Abstract: A multicomputer system can hardly avoid having faulty components in the real world. A good fault-tolerant routing scheme should tolerate as many fault patterns as possible and, hence, reduce the number of disabled functional nodes. The authors consider disconnected unsurrounded faults, i.e. all disconnected faults discussed in the literature. In disconnected unsurrounded fault models, there is no restriction on the shapes of faults. In the proposed routing scheme, a message always leaves each *f*-ring encountered at an appropriate node such that no message will encounter the same *f*-ring again and therefore never get trapped in faults.

1 Introduction

Direct networks have become a popular means for interconnecting components of multicomputers. In direct network, nodes (computers) are connected to only a few nodes, their neighbours, according to the topology of the network, and communicate with each other by passing messages. The *n*-dimensional (*n*D) mesh network is currently the most popular topology for multicomputer systems. Low dimensional mesh networks, due to their low node degree, are more popular than high dimensional mesh networks. The 2D mesh topology has been adopted by Symult 2010 [1], Intel Touchstone DELTA [2] and Intel paragon; the MIT J-machine adopts 3D mesh topology.

Network latency is one of the major factors to affect the performance of a multicomputer system. In order to minimise network latency, the wormhole switching technique has been widely used in the current generation multicomputers. 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 the same packet remain in the flit buffers along the specified route. A survey of wormhole routing for direct networks can be found in [4].

A multicomputer system can hardly avoid having faulty components in the real world. Fault-tolerant routing in direct networks has been the subject of extensive research in recent years [5–19]. Chien and Kim [10] present a partial adaptive routing algorithm for *k*-ary *n*-cubes and multi-dimensional meshes. Every fault needs to be augmented, by disabling functional nodes, to form *rectangular* faults, or *convex* faults in [10], to assure the correctness of their method. Boppana and Chalasani [12–14] proposed a method to enhance current routing algorithms for fault-tolerant routing. The concepts of a fault-ring (*f*-ring) and a fault-chain (*f*-chain) are introduced and are used for routing messages around *rectangular* faults. Sui and Wang [15] improved the routing algorithm proposed by Boppana and Chalasani. Su and Shin [17] proposed an adaptive routing algorithm for meshes and hypercubes, where a given interconnection network is decomposed into two virtual interconnection networks, VIN_1 and VIN_2. VIN_1 supports deterministic deadlock-free routing and VIN_2 supports fully adaptive routing. In *n*D meshes, their algorithm tolerates *disconnected rectangular* faults.

In a real multicomputer system, nonrectangular faults may occur. In [18], Chalasani and Boppana introduced the *solid fault model*, which can reduce the number of disabled nodes. Solid faults include all convex faults and many nonconvex faults, such as faults in the shape of L and T. Messages blocked by faults are routed along the fault ring either clockwise or counter-clockwise. Overlapped fault rings are not allowed in [18]. Kim and Han [19] also proposed a fault-tolerant routing algorithm, which is an extension of the algorithm proposed in [18]. The algorithm in [19] can tolerate any number of overlapped *f*-rings and *f*-chains based on the *solid fault model*.

Since nonrectangular and nonsolid faults may occur in a multicomputer system, a good fault-tolerant scheme should tolerate a large class of fault patterns so as to reduce the number of disabled functional nodes. We consider *disconnected unsurrounded* faults, i.e. all disconnected faults discussed in the literature. In our routing scheme, messages never encounter the same *f*-ring more than once. Each routing algorithm uses a different amount of fault information from the network to route messages. The more the fault information used, the higher the probability that the messages can avoid faults and, hence, can achieve better network performance. It is expensive to collect and distribute detailed fault information, however. In this paper, we develop routing algorithms that include a little pre-work to set node status. Only local information is needed while routing messages, i.e. each node need only to know the status of its neighbours. Since only local information is used in our algorithm, messages may encounter faults and may block each other.

IEE Proceedings online no. 20000187

DOI: 10.1049/ip-cdt:20000187

Paper first received 5th March 1999 and in revised form 9th November 1999

The authors are with the Department of Electrical Engineering, EE building, Rm. 441, National Taiwan University, Taipei 106, Taiwan

2 Preliminary

An n-dimensional mesh has $k_{n-1}k_{n-2} \dots k_0$ nodes, k_i nodes along dimension i, $0 \le i \le n-1$, and $k_i \ge 2$. Each node x is uniquely indexed by an n-tuple $(x_{n-1}, x_{n-2}, \dots, x_0)$, where $0 \le x_i \le k_i - 1$. Two nodes $x = (x_{n-1}, x_{n-2}, \dots, x_0)$ and $y = (y_{n-1}, y_{n-2}, \dots, y_0)$ are neighbours if and only if $x_i = y_i$ for all i except one, j, where $x_i = y_i \pm 1$. Each node has from n to $2n$ neighbours up to its location on the mesh. Neighbouring nodes are connected by a direct link implemented by two unidirectional physical channels with opposite directions. The four sides of a 2D mesh are hereafter labelled North, East, South and West.

A fault block is a set of connected faulty nodes and/or links. Channels incident to faulty nodes are considered faulty. In this paper, we assume that no fault touches the mesh boundaries. Therefore, an *f*-ring consists of fault-free nodes, and links can always be formed around each fault block. By exchanging link status to nonfaulty neighbours, each node can easily know its position on an *f*-ring. A node on an *f*-ring is a NE (NW, SE, SW) node, if all of its links are good and links to the South and West (South and East, North and West, North and East, respectively), are on the *f*-ring. In Fig. 1, nodes (13, 8) and (13, 0) are NE and NW nodes, respectively, on the *f*-ring of F1.

Two fault blocks are connected if their *f*-rings share some common channels; otherwise, they are disconnected. A fault block F1 is surrounded by fault block F2 if a node (x_1, y_1) exists on the *f*-ring of F1, and two nodes (x_2, y_2) and (x_3, y_3) exist on the *f*-ring of F2 such that $x_2 < x_1 < x_3$ or $y_2 < y_1 < y_3$. Similarly, a node (x, y) is surrounded by an *f*-ring if two nodes (x_2, y_2) and (x_3, y_3) exist on the *f*-ring such that $x_2 < x < x_3$ or $y_2 < y < y_3$. Faults that are not surrounded by any other faults are unsurrounded faults. Fig. 1 shows a 2D mesh with three disconnected faults: F1, F2 and F3. In Fig. 1, filled circles indicate faulty nodes, and heavy lines indicate *f*-rings. The combination of faults F1 and F2 is not allowed in the unsurrounded fault model, for F2 is surrounded by F1. In a 2D mesh, the set of nodes on an *f*-ring with the largest (smallest) index value in dimension 0 is the Emax (Emin) of the *f*-ring, and the set of nodes that has the largest (smallest) index value in dimension 1 is the Nmax (Nmin) of the *f*-ring. Nodes in each of the four sets (Emax, Emin, Nmax and Nmin) are not necessarily contiguous or in some kind of order, for the faults could be of any shape. For instance, the Nmax of F1 contains nodes (13,0), (13,1), (13,2), (13,6), (13,7), and (13,8).

A two-pass process is proposed for node identification. In pass one, node informations of *f*-rings are collected. Each NE node, say (x_1, y_1), sends a message along the *f*-ring in C.C.W. The message contains the co-ordinate of the NE node and four fields, D0max, D0min, D1max and D1min. The four fields are used to record the maximal and minimal index values in dimensions 0 and 1 of all nodes on the *f*-ring. Each node, say (x_2, y_2), of types NE, NW, SE and SW intercepts and processes the message. In order to reduce network traffic and avoid deadlock some of the messages are aborted.

Only the message generated by the East-most North-most node survived after a certain amount of time, for all other messages will eventually be aborted due to step 1 in Fig. 2. The maximal and minimal index values in dimensions 0 and 1 of all nodes on the *f*-ring are recorded in D0max, D0min, D1max and D1min after the message returns to the East-most North-most node. In pass two, the collected node information is used for node identification. In pass two, the East-most North-most nodes send D0max, D0min, D1max and D1min in a message along the *f*-ring, again in C.C.W. Each node on the *f*-ring can easily decide whether it is a node in Emax, Emin, Nmax or Nmin by comparing its index value to the values in D0max, D0min, D1max and D1min. We assume that each node uses six one-bit flags, N, E, S, W, B1 and B2, to indicate its position on an *f*-ring. The node that sends a message in pass two sets B1 directly, for it is the East-most North-most node on the *f*-ring. A node with a B2 set indicates that it is the East-most South-most node on the *f*-ring.

Nodes in Nmax, Emin, Nmin and Emax are identified sequentially in steps 1, 2, 3 and 4 in Fig. 3. Step 5 is used by the East-most South-most node of the *f*-ring to set its B2 flag, for it is the node first identified to be in Emax. The value of D1max is equal to 0 after the East-most South-most node has been identified, no other node can set B2.

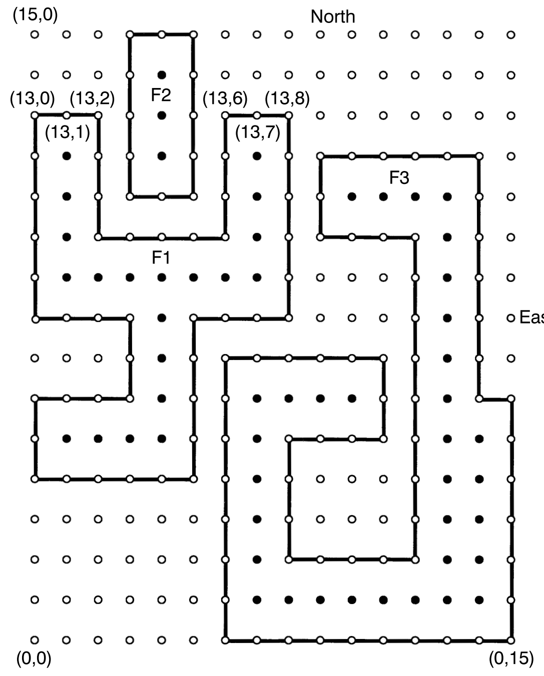

Fig. 1 *2D mesh with three disconnected faulty regions*

```
Pass one of node identification:
/* assume the message is generated at node $(x_1, y_1)$ and is intercepted by node $(x_2, y_2)$ */
/* D0max and D0min are initialised to $y_1$, and D1max and D1min are initialised to $x_1$ */
1. If $y_2 > y_1$ or $y_2 = y_1$ AND $x_2 > x_1$, abort the message.
2. If $x_2 >$ D1max, D1max $= x_2$, if $x_2 <$ D1min, D1min $= x_2$,
if $y_2 >$ D0max, D0max $= y_2$, if $y_2 <$ D0min, D0max $= y_2$.
3. Forwarding the message to its next neighbour.
```

Fig. 2 *Pass one of node identification*

```
Pass two of node identification:
/* assume the current node is $(x_1, y_1)$, and the received message contains D0max, D0min, D1max and D1min */
1. If $x_1$ = D1max, set N.
2. If $y_1$ = D0min, set W.
3. If $x_1$ = D1min, set S.
4. If $y_1$ = D0max, set E.
5. If $y_1$ = D0max AND D1max $\neq 0$, set B2 and D1max = 0.
```

Fig. 3 *Pass two of node identification*

3 Fault-tolerant routing

3.1 The FT-Routing algorithm

The well known *e*-cube routing algorithm is enhanced to be fault-tolerant in this section. In a 2D mesh, messages are routed along dimension 0 first and then routed along dimension 1. At any node, the first hop of the path, which is specified by the *e*-cube algorithm for a message, is called the *e*-hop for the message at the node. Messages are classified into one of the following four types: WE (West-to-East), EW (East-to-West), NS (North-to-South), and SN (South-to-North). Two bits in the message header can represent four message types. WE and EW messages are called row messages, NS and SN messages are column messages. Once a row message completes its routing on dimension 0, it becomes a column message. Thus, row messages can become column messages, but not vice versa. Each message is injected into the network as either a WE message or an EW message, depending on the relative position of the source and the destination node. In our routing scheme, the direction of a message is set to be null, clockwise (C.W.), or counter-clockwise (C.C.W.) at any instance of time. When a message is newly generated its direction is set to be null. Procedure Set-Type (Fig. 4) defines the rules of changing a row message to a column message.

In the fault-free region, messages have null direction and are routed along their *e*-hops. If a message header encounters an *f*-ring, depending on the message type and its location on the *f*-ring, its direction may set to be clockwise or counter-clockwise. The message is then routed on the *f*-ring along its specified direction. The direction is reset to be null again after bypassing each fault encountered.

If a WE (EW) row message encounters an *f*-ring at a node not located at the Emax (Emin) of the *f*-ring, its direction is set to be clockwise (counter-clockwise). The WE (EW) message is then routed on the *f*-ring along a clockwise (counter-clockwise) direction until either its routing on dimension 0 is completed or the Emax (Emin) of the current *f*-ring is reached. If the Emax (Emin) has been reached before the row message completes its routing on dimension 0, the row message leaves the *f*-ring at Emax (Emin); otherwise, it changes into a column message. In both cases, the row message is reset to have null direction. If a WE (EW) message is generated in the Emax (Emin), its direction remains null. Since WE (EW) row messages leave every *f*-ring encountered at the respective East-most (West-most) column, if they have not yet changed into column messages, they never encounter the same *f*-ring more than once.

When an SN (NS) column message encounters an *f*-ring at a node which is not located at the Nmax (Nmin) of the *f*-ring, its direction is set to be clockwise (counter-clockwise). The SN (NS) column message is then routed on the *f*-ring along a clockwise (counter-clockwise) direction until reaching its *off-node* (to be defined below) on the current *f*-ring. If an SN (NS) message is generated in the Nmax (Nmin), it will leave the *f*-ring directly if its *e*-hop is not occupied by another message. The off-node of an SN (NS) message is the node at which the message leaves the *f*-ring, and is used to prevent the message from looping on the *f*-ring. This assures that a column message never encounters the same *f*-ring more than once. The concept of the off-node will also help in detecting whether or not the destination node is surrounded by the *f*-ring. We will use a 1-bit flag, called FLAG, to record this status when detected. In order to find the off-node, a column message has to trace the *f*-ring along its specified direction. We assume that there is a field in the message header that holds the index of the candidate off-node and this field can be updated while the message is tracing on an *f*-ring. The candidate off-node is initialised to be the node at which the column message encounters the *f*-ring the first time. The update policy is as follows: each time the SN (NS) message gets to the destination column, it examines the index value of the node reached. If the index value in dimension 1 of this node is larger (smaller) than that of the candidate off-node, but is smaller (greater) than that of the destination node, then this node is set to be the candidate off-node for the SN (NS) message. If the index value of the visited node in dimension 1 is greater (smaller) than that of the destination node, the candidate off-node is unchanged and the SN (NS) message set the 1-bit flag, FLAG. The candidate off-node becomes the off-node for the SN (NS) message in the following two cases: *a*) if the candidate off-node is in the Nmax (Nmin); *b*) the candidate off-node has been visited the second time after travelling the entire *f*-ring. It is noticed that the off-node for an SN (NS) message is on the destination column. The index value of the off-node in dimension 1 is smaller

```
Procedure Set-Type ($M$)
/* Assume that message $M$ is currently at node $(c_1, c_0)$ and destination node is $(d_1, d_0)$ */
1. If $(c_1, c_0) = (d_1, d_0)$, consume $M$.
2. If $M$ is a row message and $c_0 = d_0$ then
      set its direction to be null and change its type to
      NS, if $c_1 > d_1$, or SN, if $c_1 < d_1$.
```

Fig. 4 *Procedure Set-Type*

(larger) than, or equal to, that of the destination node, but is greater (smaller) than that of nodes, which are below (above) the destination node, on the destination column. Therefore, no node of the *f*-ring locates between the off-node and the destination node of the SN (NS) message on the destination column.

A column message with its FLAG set means that its destination node is located in the area surrounded by the *f*-ring encountered. When the message gets to its off-node, the flag is used to guide the selection of virtual channels such that the column message will not be blocked by other column messages routing in the surrounded fault-free area. Fig. 5 shows two NS messages NS1 and NS2, whose destinations are d1 and d2, respectively. The FLAG value will be 1 for NS1 when it has traced on the *f*-ring and detected the surrounding situation, and will be 0 for NS2.

Steps 1, 2, 3 and 4 in Procedure Set-Direction (Fig. 6) are used for deciding if a message with non-null direction should be reset to have null direction again. In step 5, depending on the message type WE, EW, NS or SN, and whether the current node is a node in the Emax, Emin, Nmax or Nmin of the *f*-ring, respectively, the direction of messages are set to be C.W. or C.C.W. Routing algorithm FT-Route is given in Fig. 7.

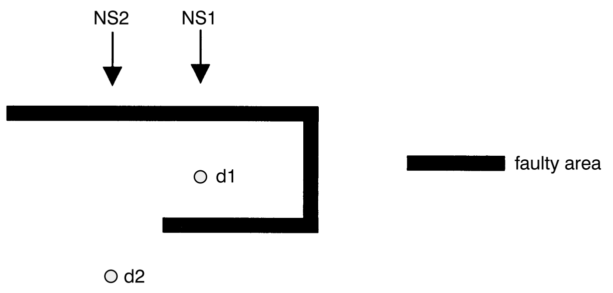

Fig. 5 *Two NS messages with different FLAG values*

```
FT-Route ($M$)
1. Set-Type ($M$).
2. Set-Direction ($M$).
3. If the direction of $M$ is null then
      route $M$ along its $e$-hop
   else
      route $M$ along the specified direction.
```

Fig. 7 *Fault-tolerant routing algorithm*

3.2 Usage of virtual channels

In our routing method, four virtual channels per physical channel are needed. Virtual channels of class i are denoted as VC_i, $0 \leq i \leq 3$. WE row messages use VC_0 in a C.W. direction, and EW row messages use VC_1 in a C.C.W. direction on their journey. Since each column message routed on an *f*-ring has to find out its off-node on the *f*-ring, it circumvents the *f*-ring encountered. After visiting all nodes on the *f*-ring, a column message has to use a different set of virtual channels to get to its off-node so as not to be blocked by the message itself. Since different column messages may encounter the same *f*-ring at a different node, we define two *breaking nodes*, where column messages switch to a different set of virtual channels, on each *f*-ring. The East-most North-most node and the East-most South-most node on each *f*-ring are the breaking nodes for NS and SN messages, respectively. An NS (SN) message uses VC_0 (VC_1) in the fault-free area, if its FLAG is of value 0. When an *f*-ring is encountered, it uses VC_0 (VC_1) before passing through the breaking node. Depending on the position where the column message encounters the *f*-ring, this column message may at most pass through the breaking node two times before getting to its off-node. NS (SN) messages use VC_2 and VC_3 in a C.W. (C.C.W.) direction after passing through the breaking node the first and the second time, respectively. If the FLAG is

```
Procedure Set-Direction ($M$)/* Assume that message $M$ is currently at $(c_1, c_0)$ */
1. If $M$ is a WE and the direction of $M$ is not null then
      if $(c_1, c_0)$ is in the Emax then
         set its direction to null
      else return
2. If $M$ is an EW and the direction of $M$ is not null then
      if $(c_1, c_0)$ is in the Emin then
         set its direction to null
      else return
3. If $M$ is an NS and the direction of $M$ is not null then
      if $(c_1, c_0)$ is the off-node then
         set its direction to null
      else return
4. If $M$ is an SN and the direction of $M$ is not null then
      if $(c_1, c_0)$ is the off-node then
         set its direction to null
      else return
5. If the direction of the WE (EW, SN, NS) message is null and $(c_1, c_0)$ is on an $f$-ring
but is not a node in the Emax (Emin, Nmax, Nmin) then set its direction to be
      C.C.W., if $M$ is an EW or NS message, or
      C.W., if $M$ is a WE or SN message.
```

Fig. 6 *Procedure Set-Direction*

set for an NS (SN) message, the NS (SN) message uses VC_2 (VC_3) between its off-node and its destination node. The usage of virtual channels is depicted in Fig. 8 and Table 1. It is clear that the four types of messages use disjointed virtual channels.

3.3 Example

Let us consider the example of routing message M from (7,0) to (3,10) in Fig. 9. M begins as a WE message and is routed along its e-hop to (7,3), where its e-hop is faulty. It is then misrouted along the f-ring of F1 in a clockwise orientation to (13,8). Since (13,8) is a node in the Emax of the f-ring, the direction of M is reset to be null at the node.

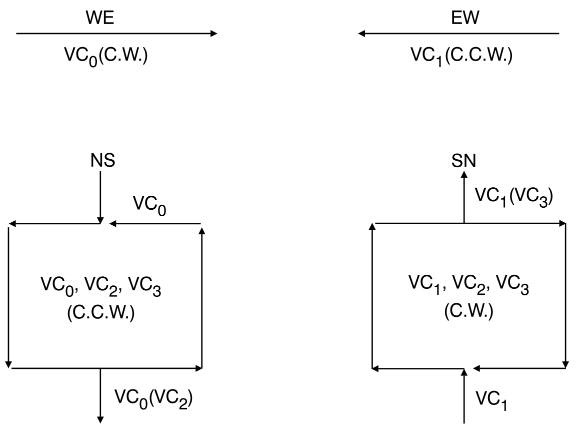

Fig. 8 *Usage of virtual channels by WE, EW, NS and SN messages*

Table 1: Virtual channels used by messages on *f*-rings

	C.W.	C.C.W.
VC_0	WE	NS
VC_1	SN	EW
VC_2	SN	NS
VC_3	SN	NS

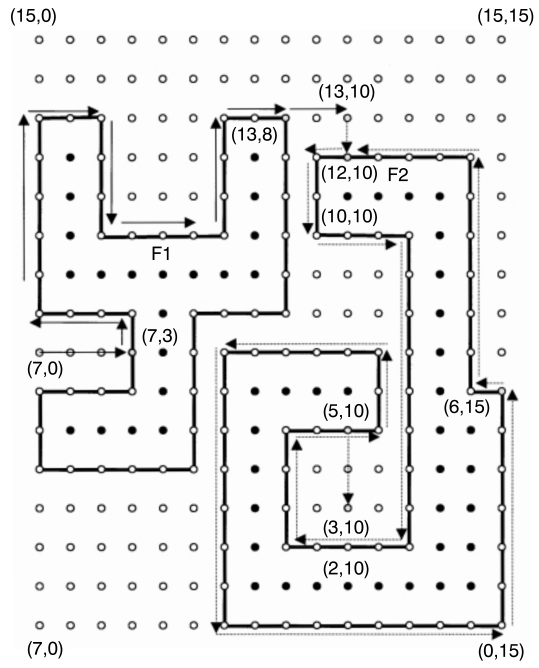

Fig. 9 *A routing example from (7,0) to (3,10)*

M is then routed along its e-hop to (13,10), where M becomes an NS message. From node (7,0) to (13,10), M uses VC_0. At (13,10), M is routed along its e-hop to (12,10), where its e-hop is faulty. It is then routed along the f-ring of F2 in the counter-clockwise orientation, and its candidate off-node is (12,10). Node (10,10) becomes the candidate off-node for M when the message gets to it, for its index value in dimension 1 is smaller than that of the candidate off-node (12,10) and is greater than that of the destination node (3,10). When M gets to (2,10), the candidate off-node is still (10,10) and FLAG is set, for the index value of (2,10) in dimension 1 is smaller than that of the destination node. Node (5,10) becomes the candidate off-node when M gets to it. M is then routed along the f-ring of F2 with its off-node unchanged afterward. From (13,10) to (6,15), M uses VC_0. Since (6,15) is the breaking node for NS messages, M begins to use VC_2 at (6,15). When M gets to (5,10) again, (5,10) becomes the off-node for M. The direction of M is then reset to null and M is routed along its e-hop to its destination node (3,10) using VC_2, for the FLAG is set.

3.4. Deadlock-freedom of algorithm FT-Route

We have the following facts: a) each of the four types (EW, WE, NS and SN) of messages use a disjoint set of virtual channels; b) row messages (EW and WE) can become column messages (NS and SN), but not vice-versa; c) EW and WE messages cannot change into each other, and SN and NS messages cannot change into each other. Thus, to prove that the algorithm FT-Route is deadlock-free it is necessary only to prove the deadlock-freedom in each of the four types of messages. Since the routing schemes for WE and EW messages are the same, as are the routing methods for NS and SN messages, we have given proof of deadlock-freedom for WE and NS messages only.

Lemma 1: No deadlock occurs among WE messages.

Proof: We assign an integer channel number to each virtual channel in the West-to-East VC_0. In an X*Y mesh, virtual channels in VC_0 from (i, j) to $(i, j+1)$ have channel number j, where $0 \le i \le X-1$, $0 \le j \le Y-2$. When faults occur, a non-integer channel number is assigned to each virtual channel on f-rings. If the top-most node in the Emax of an f-ring is (x, y), then each virtual channel on the f-ring is assigned a non-integer channel number between $y-1$ and y, $0 \le y \le Y-2$. The channel number is assigned increasingly from the virtual channel leaving the top-most node in the Emax to the virtual channel entering the top-most node in the Emax in a clockwise orientation. A channel number assignment example is given in Fig. 10.

We have the following facts: a) WE messages routed in a fault-free region travel virtual channels in an increasing order of channel number; b) the channel number of any West-to-East virtual channel entering an f-ring is less than that of every virtual channel on the f-ring entered; c) WE messages routed on f-rings travel virtual channels in an increasing order of channel number; d) the channel number of the virtual channels leaving an f-ring at the Emax is larger than that of every virtual channel on the f-ring. Therefore each WE message travels virtual channels in an increasing order of channel number on its journey. Hence, no deadlock occurs among WE messages. □

Lemma 2: No deadlock occurs among EW messages.

Proof: Similar to the proof of lemma 1.

Lemma 3: No deadlock occurs among NS messages.

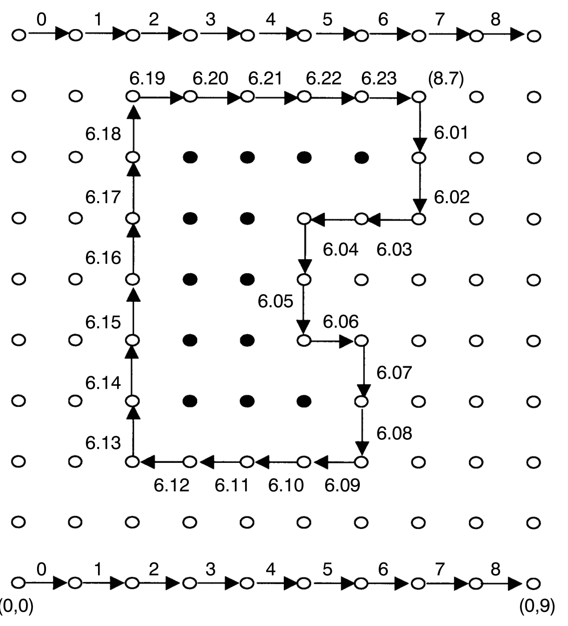

Fig. 10 *Channel number assignment*

Proof: Lemma 3 is proved by showing that no deadlock can occur among NS messages in the following three cases: a) on one *f*-ring, b) on one *f*-ring and in the fault-free area surrounded by the *f*-ring and c) on multiple *f*-rings.

NS messages routed on *f*-rings use counter-clockwise orientation to travel on virtual channels. Each time they pass through the breaking node, they use a different set of virtual channels. Depending upon the number of times NS messages pass through the breaking node, VC_0, VC_2 and VC_3 are used sequentially by NS messages. Since each NS message can pass through the breaking node two times at most, three sets of virtual channels are enough to guarantee that no deadlock can occur among NS messages routed on one *f*-ring.

If the destination node of an NS message is located in the fault-free area surrounded by an *f*-ring, the value of the FLAG is 1 when the NS message gets to its off-node on the *f*-ring. Then VC_2, instead of VC_0, is used by the NS message in the fault-free area. Hence, the NS message will not be blocked by any NS message, which may encounter the *f*-ring on its journey, using VC_0 in the surrounded fault-free area.

Since a) the off-node, where an NS message leaves an *f*-ring, is surely below the node where the NS message enters the *f*-ring; b) each NS message is routed from North to South in the fault-free area, *f*-rings are used acyclically by NS messages. Hence, no deadlock can occur among NS messages routed on multiple *f*-rings. □

Lemma 4: No deadlock occurs among SN messages.

Proof: Similar to the proof of lemma 3.

Theorem 1: Algorithm FT-Route is deadlock-free.

Proof: From lemmas 1, 2, 3, and 4, it is clear that no deadlock would occur among each of the four types of messages. Thus, the FT-Route is a deadlock-free routing algorithm. □

4 Conclusion

Most of the fault-tolerant routing schemes in the literature assume rectangular or solid faults, although in the real world, it is likely that nonrectangular and nonsolid fault patterns may occur. A large number of functional nodes may have to be disabled to transform nonrectangular and nonsolid faults to rectangular and solid faults. We have proposed a fault-tolerant routing algorithm for 2D meshes to handle disconnected unsurrounded faults, this covering all disconnected faults discussed in the literature. Since there is no restriction on the shapes of the faults in the disconnected unsurrounded fault model, the number of disabled functional nodes can be reduced greatly. The concept of the *f*-ring is used in our routing scheme. Misrouted messages are routed along *f*-rings until they reach an appropriate node such that messages never encounter the same *f*-ring more than once. Messages therefore never get trapped in faults. By fully utilising virtual channels in C.W. and C.C.W. directions on *f*-rings, only four virtual channels per physical channel are needed.

5 References

1 SEITZ, C.L., ATHAS, W.C., FLAIG, C.M., MARTIN, A.J., SEIZOVIC, J., STEELE, C.S., and SU, W.K.: 'The architecture and programming of the Ametek Series 200 multicomputer'. Proc. 3rd Conference Hypercube Concurrent Computers and Applications, Jan. 1988, vol. **I**, pp. 33–36, Association for Computing Machinery

2 Intel Corp.: 'A Touchstone DELTA System Description', 1991

3 DALLY, W.J., and SEITZ, C.L.: 'Deadlock-free message routing in multiprocessor interconnection networks', *IEEE Trans. Comput.*, 1987, **C-36**, (5), pp. 547–553

4 NI, L.M., and MCKINLEY, P.K.: 'A survey of wormhole routing techniques in direct networks', *IEEE Comput.*, 1993, 62–76

5 DALLY, W.J., and AOKI, H.: 'Deadlock-free adaptive routing in multicomputer networks using virtual channels', *IEEE Trans. Parallel Distrib. Syst.*, 1993, **4**, (4), pp. 466–475

6 GLASS, C.J., and NI, L.M.: 'Fault-tolerant wormhole routing in meshes'. Proc. 23th Ann. Int'l Symp. Fault-Tolerant Computing, 1993, pp. 240–249

7 LINDER, D.H., and HARDEN, J.C.: 'An adaptive and fault tolerant wormhole routing strategy for *k*-ary *n*-cubes', *IEEE Trans. Comput.*, 1991, **40**, (1), pp. 2–12

8 CHEN, M.S., and SHIN, K.G.: 'Depth-first search approach for fault-tolerant routing in hypercube multiprocessors', *IEEE Trans. Parallel Distrib. Syst.*, 1990, **1**, (2), pp. 152–159

9 CHEN, M.S., and SHIN, K.G.: 'Adaptive Fault-tolerant routing in multicomputers', *IEEE Trans. Comput.*, 1990, **39**, (12), pp. 1406–1416

10 CHIEN, A.A., and KIM, J.H.: 'Planar-adaptive routing: low-cost adaptive networks for multi-processors', *Proc. 19th Int'l Symp. Computer Architecture*, 1992, 268–277

11 LEE, T.C., and HAYES, J.P.: 'A fault-tolerant communication scheme for hypercube computers', *IEEE Trans. Comput.*, 1992, **41**, (10), pp. 1242–1256

12 BOPPANA, R.V., and CHALASSANI, S.: 'Fault-tolerant routing with non-adaptive wormhole algorithms in mesh networks', *Supercomput.*, 1994, 693–702

13 CHALASSANI, S., and BOPPANA, R.V.: 'Adaptive fault-tolerant wormhole routing algorithms with low virtual channels requirements'. Int'l Symp. Parallel Architecture, and Networks, 1994, pp. 214–221

14 BOPPANA, R.V., and CHALASSANI, S.: 'Fault-tolerant wormhole routing algorithms for mesh networks', *IEEE Trans. Comput.*, 1995, **44**, (7), pp. 848–864

15 SUI, P.H., and WANG, S.D.: 'An improved algorithm for fault-tolerant wormhole routing in meshes', *IEEE Trans. Comput.*, 1997, **46**, (9), pp. 1040–1042

16 CHIU, G.M., and WU, S.P.: 'A fault-tolerant routing strategy in hypercube multicomputers', *IEEE Trans. Comput.*, 1996, **45**, (**2**), pp. 143–155

17 SU, C.C., and SHIN, K.G.: 'Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes', *IEEE Trans. Comput.*, 1996, **45**, (**6**), pp. 666–682

18 CHALASSANI, S., and BOPPANA, R.V.: 'Communication in multicomputers with nonconvex faults', *IEEE Trans. Comput.*, 1997, **46**, (**5**), pp. 612–622

19 KIM, S.P., and HAN, T.: 'Fault-tolerant wormhole routing in mesh with overlapped solid fault regions', *Parallel Comput.*, 1997, **23**, pp. 1937–1962