International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 4, July- August 2017 ISSN 2278-6856

Minimal Adaptive Fault Tolerant Routing in Multi-Mesh

Amit Datta[1], Mallika De[2]

[1] Department of Engineering & Technological Studies, University of Kalyani Kalyani, West Bengal 741235, India

[2]Department of Computer Science and Engineering, Dr. Sudhir Chandra Sur Degree Engineering College, West Bengal 700074, Kolkata, India

Abstract

In this paper an algorithm has been proposed to route a message from a source in one mesh to a destination in another mesh in a Multi-Mesh (MM) architecture constructed by n^2 different blocks or meshes where each mesh is constructed by n × n processors. Depending on the position of the mesh in MM architecture there are two self loops within the mesh – one is horizontal and another is vertical. The algorithm proposed here traces different paths from source to destination and select the shortest path in the absence of node or link failure In the presence of node or link failure the message will move to another intermediate block from where a new path would be re-calculated to reach the destination.

Keywords: Deterministic routing, Adaptive routing, Minimal routing, Multi-Mesh, Shortest path, Deadlock in routing, Fault tolerant, Virtual channel, e-cube algorithm

1. INTRODUCTION

Different architectures are being used to support parallel processing. Among those direct network is one of the popular architecture in which each node is directly connected to specified number of neighbours by means of a set of channels. For example, *k*-ary *n*-cube is an *n*-dimensional grid structure having *k*-nodes in each dimension and each node is directly connected to two other nodes in each dimension. This *k*-ary *n*-cube direct network is called *n*-dimensional mesh topology.

Wormhole (WH) routing [2], [3], [5] is one of the popular techniques for message communication in direct network.This routing algorithm divides the message into packets which further divides into sequence of fixed-size unit of data called flits. In WH routing while a flit is transferred through a communication channel, the channel is reserved until all the flits corresponding to the packet are transferred.

There are several works on fault tolerant wormhole routing algorithm in meshes and other networks [4-14]. Boppana and Chalasani [2] proposed a fault model for mesh network. According to the model link or node failure inside a mesh network can create rectangular f-ring or f-chain formed by non-faulty nodes and links surrounding the rectangular fault region. They also provided a fault tolerant fully adaptive routing algorithm for Mesh network and four virtual channels had been used in the presence of overlapping f-rings and f-chains.

Other than the e-cube algorithm [2] which is more suited for mesh, there are Region-based routing algorithm [13] and positive-first and negative-first turn models [14].

A fault tolerant non adaptive routing algorithm has been proposed in [1] for routing messages in Multi Mesh network consisting of n^4 processors where a single mesh (block) consist of $n \times n$ processors and the whole Multi-Mesh network is constructed by such $n \times n$ meshes. The degree of each processor in this network is 4 and the diameter of the network is $2n$.

In this paper a fault tolerant fully adaptive routing algorithm for MM network has been proposed. Due to adaptive nature this algorithm permits all the possible paths among the source destination pair and selects the path having shortest path length. While message is traversing inside the mesh it utilizes the f-cube4 [2] algorithm to reach the destination.

During traversal through the shortest path, message may encounter link fault or node fault and then it goes through the f-ring or f-chain to reach the destination inside the mesh [2]. To avoid inter block link fault, message moves to neighbouring mesh/block and from there a new path is determined to reach the destination. To avoid deadlock inside mesh three virtual channels c_0, c_1, c_2 are being used [3]. The proposed algorithm is deadlock and livelock free. To overcome deadlock involving inter block link same virtual channels c_0, c_1 and c_2 have been used.

2. CLASSIFICATIONS OF ROUTING ALGORITHMS

Routing algorithm can be classified depending on different characteristics [17]. It can be classified as source routing and distributed routing based on the place where routing decision has been made. In source routing entire path for message routing is selected at the source node before the message is sent for the destination. In case of distributed routing the routing decision is made at the intermediate node through which message traverses. The routing algorithm decides next neighbour to which the packet should be sent.

Based on the path selection process routing can be classified as deterministic and adaptive. While the message traversal path is selected based on source and destination address only, it is called deterministic routing. In deterministic routing only one path is available between the source and destination.

Adaptive routing on the other hand provides multiple paths between source and destination based on the network conditions and the routing algorithm. Routing algorithm can be as minimal and non-minimal. In minimal routing a message traverses along one of the shortest paths between source and destination. In non-minimal routing a message can select any path between source and destination and this might be a longer path.

3. ADAPTIVE WORMHOLE ROUTING ALGORITHM

Deterministic algorithms provide one and only one path between source and destination. To avoid network congestion and enhance fault tolerance [7], [10]-[12], it is preferred that the routing algorithm provide multiple paths between source destination pair. Algorithms that adapt to the network and traffic condition and provide multiple paths between source destination pair are called adaptive routing algorithms and classified as fully adaptive or partially adaptive routing algorithms based on whether these algorithms allow all possible paths or only a subset of them [8], [9], [17]. In this paper a minimal fully adaptive routing algorithm has been proposed for Multi Mesh (MM) network.

4. THE MULTI MESH (MM) NETWORK

The Multi-Mesh network that was proposed by the authors in [15] is made up of n^2 meshes. In an $n \times n$ mesh, the processors are arranged in n rows and n columns. Such a mesh is used as the basic building block of the Multi-Mesh (MM) network. Here total n^2 such meshes are arranged in the form of an $n \times n$ matrix where each constituent matrix is termed as a block in MM network. In each block there are $4(n\text{-}2)$ processors on the four outer boundaries each of which has three neighbours within that block. These are called boundary processors. Also, in each block there are four corner processors each of which has two neighbours within that block. These are called corner processors. The rest of the $(n\text{-}2)^2$ processors in every block will be termed as internal processors. Each block in this network is connected to another block by suitable links so that each processor has four links in this network topology.

A processor inside a given block can be uniquely identified by two coordinates. Again blocks are organized as matrix form so each block can be identified by two coordinates, say α and β as B (α, β). Thus, each of the n^4 processors in MM can uniquely be identified using a 4-tuple of the coordinate values. The first two coordinates are used to describe the block in which the processor lies and the other two coordinates are used to identify the position of the processor inside that particular block. For example, P (α, β, x, y) is a processor lying at the x-th row and y-th column of the block B (α, β). Each of these four coordinates has value between 1 to n. A special symbol * will be used for any one of these four coordinates to denote the set of all processors with all possible values of the respective coordinates. For example, P (*, *, 1, 1) signifies the set of the top left corner processors of all the n^2 blocks. If the processors P(α, β_1, x_1, y_1) are connected to P(α, β_2, x_2, y_2) for all values of α, $1 \leq \alpha \leq n$, we denote these sets of links by an interconnection between the sets P(*, β_1, x_1, y_1) and P(*, β_2, x_2, y_2). Inter-block connections among the boundary processors are given by the following rules:
Vertical Connection is identified by following rule

u β, $1 \leq \beta \leq n$, P (α, β, 1, y) are connected to P (y, β, n, α), where $1 \leq y, \alpha \leq n$, and

Horizontal Connection is identified by following rule

u α, $1 \leq \alpha \leq n$, P (α, β, x, 1) are connected to P (α, x, β, n), where $1 \leq x, \beta \leq n$

All these links are two-way connections. Hence, in the multi-mesh network, all processors have a uniform degree of 4. These inter-block connections among the boundary processors are called inter-block links.

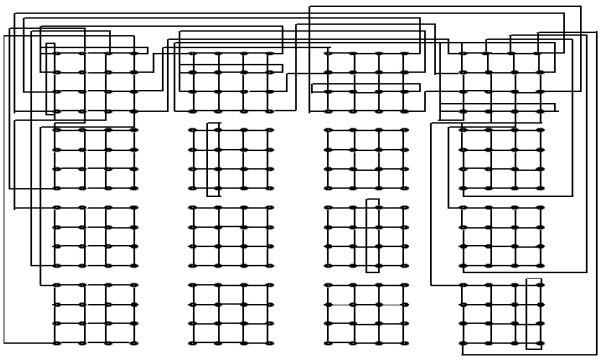

Figure 1 A simple n × n Multi-Mesh network with n = 4

5. ADAPTIVE ROUTING ALGORITHM IN MULTI-MESH

In this section it is shown how to trace the minimal adaptive path from source to destination in MM and traverse the message in that path to reach the destination. An example of routing a message through minimal adaptive routing path in multi mesh is shown below.

5.1 An example for the message routing in Multi-Mesh

Example 1: In a 4×4 Multi Mesh a message originated at P (1, 1, 2, 2) and destined for P (3, 4, 2, 4). The message traversal in minimal path is shown below.

In the source block B (1, 1) there are two different exit points, i.e. (4, 1) and (4, 4) to move the message into horizontal intermediate block. Similarly, there are two different exit points i.e. (1, 3) and (4, 3) to move the message into vertical intermediate block. To traverse a

message from source to destination in minimal path in each block minimal length path has been considered and the minimal length of path has been calculated for the entire path. In Fig. 2, two different paths are shown; one through horizontal intermediate block and another through vertical intermediate block and among those two paths the length of the path through vertical intermediate block is less than the length of the path through horizontal intermediate block. In each of the intermediate block the minimum length path has been considered to traverse from the source to exit processor of that block. The minimal adaptive path has been shown in bold in Fig. 2.

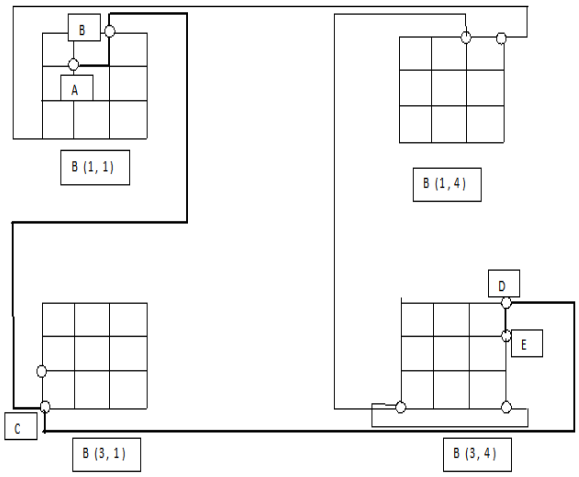

Figure 2 Message routing through minimal adaptive path (Minimal path has been shown in bold) and message is identified as a small circle

Considering the horizontal intermediate block, source to exit processor path lengths inside the source block are calculated as given below.

P (1, 1, 2, 2) → P (1, 1, 4, 1), path length = 3 and P (1, 1, 2, 2) → P (1, 1, 4, 4), path length = 4. Here the minimum length is 3.

Considering P (1, 1, 4, 1) as the source block exit point, the source/entry point at the horizontal intermediate block is P (1, 4, 1, 4) and the corresponding path length is 1. There will be two exit points in this intermediate block to move the message into destination block. These two paths with length in the intermediate block are given below. P (1, 4, 1, 4) → P (1, 4, 1, 3), path length = 1 P (1, 4, 1, 4) → P (1, 4, 4, 3), path length = 4. Here the minimum length is 1.

Considering P (1, 4, 1, 3) as the exit point in the intermediate block, the entry point/source processor in the destination block is P (3, 4, 4, 1) and the corresponding path length for this inter block movement is 1. Now to move the message from P (3, 4, 4, 1) to P (3, 4, 2, 4), the required minimum path length is 3 using the horizontal self loop.

So, the minimum path length to move the message from the source to destination through horizontal intermediate block is 3 + 1 + 1 + 1 + 3 = 9. Now, considering the vertical intermediate block, source to exit processor path lengths inside the source block are calculated as given below.

P (1, 1, 2, 2) → P (1, 1, 1, 3), path length = 2 and P (1, 1, 2, 2) → P (1, 1, 4, 3), path length = 3. Here the minimum length is 2.

Considering P (1, 1, 1, 3) as the source block exit point the source/entry point at the vertical intermediate block is P (3, 1, 4, 1) and the corresponding path length is 1. There will be two exit points in this intermediate block to move the message into destination block. These two paths with length in the intermediate block are given below. P (3, 1, 4, 1) is both the entry and exit processor in the in this intermediate block, so the path length between source and destination in the intermediate block is 0. Another exit point in this intermediate block is P (3, 1, 4, 4) for which distance between entry and exit point in this intermediate block is 3.

Here the minimum length i.e. 0 is considered for the vertical intermediate block. Considering P (3, 1, 4, 1) as the exit point in the vertical intermediate block, the entry point/source processor in the destination block is P (3, 4, 1, 4) and the corresponding path length for this inter block movement is 1. Now to move the message from P (3, 4, 1, 4) to P (3, 4, 2, 4) the required minimum path length is 1 using normal path. The minimum path length to move the message from the source to destination through vertical intermediate block is calculated as 2 + 1 + 0 + 1 + 1 = 5. So, the minimum path length is 5 and the path is ABCDE, which is through the vertical intermediate block as shown in Fig. 2. Now if in the above communication, say the link from (3, 1, 4, 1) to (3, 4, 1, 4) got disconnected during message traversal, the message will move to the node P (3, 1, 3, 1) (denoted by R) in clockwise direction. From this node the message will reach P (3, 3, 1, 4) (denoted by S) at block B (3, 3), which is the new source block due to fault in the communication path. So, co-ordinate of the source block exit processor = (3, 3, 4, 4) and co-ordinate of the intermediate block exit processor = 0**.** The shortest path from the new source block B (3, 3) to the destination is given below.

P (3, 3, 1, 4) → P (3, 3, 4, 4) → P (3, 4, 3, 1) → P (3, 4, 4, 1) → P (3, 4, 4, 4) → P (3, 4, 2, 4)
Path is denoted by STUVWX in figure 3 and the length is 3+1+1+1+2 = 8.

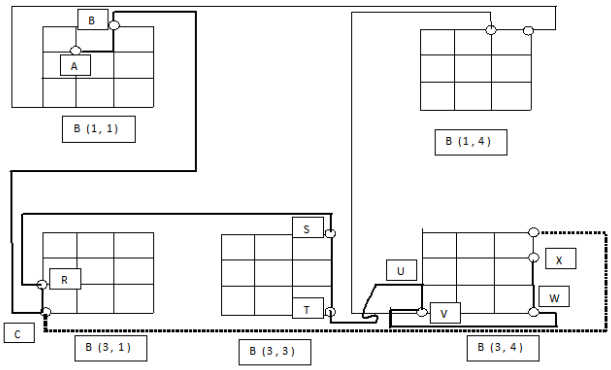

Figure 3 Message routing in alternative path in case of fault in minimal path (Dotted line is showing the broken link due to fault)

The entire source to destination path is given below.

P (1, 1, 2, 2) → P (1, 1, 1, 3) → P (3, 1, 4, 1) → P (3, 1, 3, 1) → P (3, 3, 1, 4) → P (3, 3, 4, 4) → P (3, 4, 3, 1) → P (3, 4, 4, 1) → P (3, 4, 4, 4) → P (3, 4, 2, 4).

Denoted by ABCRSTUVWX in Fig. 3 and the path length is 13. Here due to the inter block link fault the traversed path length becomes 13 instead of 5 as shown in Fig. 3.
In the above example if the link (2, 2) to (2, 3) in B (1, 1) becomes faulty then as per f-cube4 algorithm [2] the message will be routed clockwise direction to reach at (1, 2) and then the message will reach to the destination node inside the mesh i.e. (1, 3). The block B (1, 1) with link fault is shown in Fig. 4.

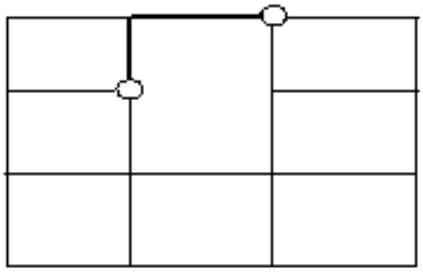

Figure 4 Link fault inside mesh and message traversal (shown in bold) for misrouted message as per f-cube4 algorithm

5.2 Minimal Adaptive Routing Algorithm and functions

- Minimal_Adaptive_Routing_In_MM – The main procedure to route the message from source to destination in a fault tolerant adaptive path.
- CalculateShortestPathMM – Procedure to calculate the shortest path in each block while message is moving from source block to destination block in MM.
- CalculateSDLengthInBlk – Procedure to calculate the distance between source and destination inside a mesh while the message is traversing from the source to destination processor.

Procedure Minimal_Adaptive_Routing_In_MM ( )
/* f1 = Co-ordinate of the source block exit processor, f2 = Co-ordinate of the intermediate block exit processor, f3 = Co-ordinate of the destination processor ***/**

1. Input message (msg), message source (msgSrc) and message destination (msgDest).
2. f1 = 0; f2 = 0; f3 = 0;
3. shortestPathinMM = CalculateShortestPathMM (msg, msgSrc, msgDest, f1, f2, f3)

/* shortestPathinMM will contain path within each block while message is moving from source block to destination block in MM*/

4. The message is send through the above shortest path in MM from step 3 using the following rules.
   i. Move the message as per above shortest path inside mesh but in case of any faulty node inside mesh the message will reach to the destination node inside mesh using the f-cube4 algorithm [2] using f-ring/f-chain.
   ii. Move the message from the given source to f1, the node by which the blocks are interlinked.
   iii. if the inter block link at f1 is not disconnected go to step iv

if the inter-block link at the f1 is disconnected
if (the message is moving horizontally and the coordinate of the message is any of (2, n) to (n, n) or any of (2, 1) to (n, 1))
Move the message towards clockwise which basically shift the message one step up and reaches to a neighboring processor, following that move to a new block through corresponding inter block link
end if
else // corner point (1, 1) or (1, n)
Move the message towards counter-clockwise which basically shift the message one step down along column and reaches to a neighbouring processor, following that move to a new block through corresponding inter block link
end //end of else
Set msgSrc as the entry processor of the new block
Go to Step 3
if (the message is moving vertically and the coordinate of that message is any of (n, 2) to (n, n) or any of (1, 1) to (1, n-1) in that mesh)
Move the message towards counter-clockwise which basically shift the

message to the left of the actual exit processor and reaches to a neighbouring processor following that move to a new block through corresponding inter block link
end if
else // corner point (1, n) or (n, 1)
Move the message towards clockwise which basically shift the message to the right of the actual exit processor and reaches to a neighbouring processor, following that move to a new block through corresponding inter block link
end // end of else
Set msgSrc as the entry processor of the new block
Go to Step 3
end if // end of if

iv. if ($f2 \neq 0$)

a. Move the message from the f1 to source/ entry processor of the intermediate block
b. Move the message from the source processor at the intermediate block to f2 as per path in step 3 (shortestPathinMM). For any fault inside mesh use the f-cube4 algorithm [2].
c. if the inter block link at the f2 is not disconnected go to step v
if the inter-block link at the f2 is disconnected
if (the message is moving horizontally and the coordinate of the message is any of (2, n) to (n, n) or any of (2, 1) to (n, 1))
Move the message towards clockwise which basically shift the message one step up and reaches to a neighbouring processor, following that move to a new block through corresponding inter block link
end if // end of if
else // (1, 1) or (1, n) i.e., corner point
Move the message towards counter-clockwise which basically shift the message one step down along column and reaches to a neighbouring processor, following that move to a new block through corresponding inter block link
end //end of else
Calculate the shortest path between new block/mesh and destination block
Update values for f2 considering the new intermediate block
Go to step v
if (the message is moving vertically and the coordinate of that message is any of (n, 2) to (n, n) or any of (1, 1) to (1, n-1) in that mesh)
Move the message towards counter-clockwise which basically shift the message to the left of the actual exit processor and reaches to a neighbouring processor, following that move to a new block through corresponding inter block link
end if
else // (1, n) or (n, 1), i.e., corner //point
Move the message towards clockwise which basically shift the message to the right of the actual exit processor and reaches to a neighbouring processor, following that move to a new block through corresponding inter block link
end //end of else
Calculate the shortest path between new block/mesh and destination block
Update values for f2 considering the new intermediate block
end if // end of if the inter-block link at the f2 is disconnected
end if // end of if ($f2 \neq 0$)

v. Move the message from f1/f2 into source/entry processor of the destination block
vi. Move the message from the source processor at the destination block to f3 (destination) using the shortestPathinMM of step 3. For any fault inside mesh use the f-cube4 algorithm [2] as mentioned above.

end Minimal_Adaptive_Routing_In_MM ()

In example 1, the message movement is described as given below.

f1:= (1, 1, 1, 3), f2:= (3, 1, 4, 1), f3:= (3, 4, 2, 4)

Move the message from P (1, 1, 2, 2) to P (1, 1, 1, 3) as per step ii. As the inter block link at f1 is not disconnected here, the message will move to P (3, 1, 4, 1) as per step iii and iv in the above algorithm. Move the message to f2 i.e. P (3, 1, 4, 1) from P (3, 1, 4, 1) as per step iv (b) of the above algorithm. Move the message to P (3, 4, 1, 4) from P (3, 1, 4, 1) as per step v through sub-step c of step iv of the above algorithm. Move the message from P (3, 4, 1, 4) to P (3, 4, 2, 4) as per step vi of the above algorithm.

In the example 1 if the f1 is disconnected i.e. the link < (3, 1, 4, 1), (3, 4, 1, 4)> is disconnected. In that case the initial f2 will be changed to (3, 3, 4, 4) due to the disconnected link at P (3, 1, 4, 1) as per step iv in the above algorithm. Due to disconnected link the message will move from P (3, 3, 4, 4) to P (3, 4, 3, 1) then to P (3, 4, 2, 2) i.e. destination of the message.

Procedure CalculateShortestPathMM (msg, msgSrc, msgDest, f1, f2, f3)

if ((α-value of msgSrc != α-value of msgDest) and (β-value of msgSrc != β-value of msgDest))

1. If the source block is B (α_s, β_s) and destination block is B (α_d, β_d), the horizontal intermediate block will be at β_d column of the multi mesh and vertical intermediate block will at α_d row of the multi mesh.
2. There are two exit points in the source block to move the message along horizontal intermediate block and two exit points in this source block to move the message along vertical intermediate block. (β_d, 1) and (β_d, n) are exit points for the horizontal intermediate block and (1, α_d) and (n, α_d) for the vertical intermediate block.

/* In example 1, B (1, 1) is source block and B (3, 4) is destination block for a message. Here P (1, 1, 4, 1) and P (1, 1, 4, 4) are exit processors along horizontal intermediate block. The exit processors along vertical intermediate blocks are P (1, 1, 1, 3) and P (1, 1, 4, 3)*/

3. SBHmin = CalculateSDLengthInBlk ((αs, βs, x1, y1), (αs, βs, βd, 1), n) /* SBHmin is the distance between source processor to the one exit processor in source block (Read as "Horizontal wise minimum in source block")
4. SBHmax = CalculateSDLengthInBlk ((αs, βs, x1, y1), (αs, βs, βd, n), n) /*SBHmax is the distance between source processor to the another exit processor in source block (Read as "Horizontal wise maximum in source block") */

// In example 1, SBHmin = 3 and SBHmax = 4
//SBHminExit is the exit processor for minimum path from source processor to exit processor in source block
SBHminExit = (αs, βs, βd, 1)
if (SBHmax < SBHmin)
SBHmin = SBHmax
SBHminExit := (αs, βs, βd, n)
end if
//In example 1, SBHminExit := (1, 1, 4, 1)
HIsrc = (y coordinate of SBHminExit = = 1) ? (αs, βd, βs, n) : (αs, βd, βs, 1)
// Source processor at horizontal intermediate block
//In example 1, HIsrc = (1, 4, 1, 4)
HIBmin = CalculateSDLengthInBlk (HIsrc, (αs, βd, 1, αd), n) /*HIBmin is the distance between source/entry processor and one exit processor inside horizontal intermediate block */
HIBmax = CalculateSDLengthInBlk (HIsrc, (αs, βd, n, αd), n) /*HIBmax is the distance between source/entry processor and another exit processor inside horizontal intermediate block*/
//In example 1, HIBmin = 1 and HIBmax = 4
HIBminExit := (αs, βd, 1, αd) /* HIBminExit is the exit processor for minimum path from source processor to exit processor inside horizontal intermediate block*/
if (HIBmax < HIBmin)
HIBmin = HIBmax
HIBminExit := (αs, βd, n, αd)
end if
//In example 1, HIBminExit := (1, 4, 1, 3)
DESHsrc = (x coordinate of HIBminExit = = 1) ? (αd, βd, n, αs) : (αd, βd, 1, αs)
/* DESHsrc is source/entry processor at destination mesh reached via horizontal intermediate block*/
//In example 1, DESHsrc := (3, 4, 4, 1)
DESHL = CalculateSDLengthInBlk(DESHsrc, (αd, βd, x2, y2), n) /*DESHL is distance between source/entry processor to destination processor inside destination block reached via horizontal intermediate block*/
/*In example 1, DESHL = 3. */
pathLength_HI = SBHmin + 1 + HIBmin + 1 + DESHL /* Path length from source to destination via horizontal intermediate mesh/block and the path length for inter block movement is considered as 1. */
/*In example 1, pathLength_HI = 3 + 1 + 1 + 1 + 3 = 9*/

5. SBVmin = CalculateSDLengthInBlk((αs, βs, x1, y1), (αs, βs, 1, αd), n) /* SBVmin is the distance between source processor to the one exit processor in source block (Read as "Vertical wise minimum in source block")*/

SBVmax = CalculateSDLengthInBlk((αs, βs, x1, y1), (αs, βs, n, αd), n) /*SBVmax is the distance between source processor to the another exit processor in source block (Read as "Vertical wise maximum in source block")*/
/*In example 1, SBVmin = 2 and SBVmax = 3*/
SBVminExit := (αs, βs, 1, αd) /* SBVminExit is the minimum path length from source processor to exit processor along vertical intermediate block in source block */
if(SBVmax < SBVmin)
SBVmin = SBVmax
SBVminExit := (αs, βs, n, αd)
end if // end of if
/*In example 1, SBVmin = 2, SBVminExit := (1, 1, 1, 3)*/
VIsrc = (x coordinate of SBVminExit = =1) ? (αd, βs, n, αs) : (αd, βs, 1, αs) /*Source/entry processor at vertical intermediate block*/
//In example 1, VIsrc := (3, 1, 4, 1)
VIBmin = CalculateSDLengthInBlk(VIsrc, (αd, βs, βd, 1), n) /*VIBmin is the distance between source/entry processor and one exit processor inside vertical intermediate block*/
VIBmax = CalculateSDLengthInBlk(VIsrc, (αd, βs, βd, n), n) /*VIBmax is the distance between source/entry processor and another exit processor inside vertical intermediate block */
//In example 1, VIBmin = 0 and VIBmax = 3

VIBminExit := (αd, βs, βd, 1) /*VIBminExit is the minimum path length from source processor to exit processor inside vertical intermediate block*/
if (VIBmax < VIBmin)
VIBmin = VIBmax
VIBminExit = (αd, βs, βd, n)
end if
//In example 1, VIBmin = 0, VIBminExit := (3, 1, 4, 1)
DESVsrc = (y coordinate of VIBminExit = =1) ? (α_d, β_d, β_s, n) : (α_d, β_d, β_s, 1) /* DESVsrc is source/entry processor at destination mesh reached via vertical intermediate block*/
//In example 1, DESVsrc = (3, 4, 1, 4)
DESVL = **CalculateSDLengthInBlk (**DESVsrc**,** (α_d, β_d, x_2, y_2), n**) /***DESVL is distance between source/entry processor to destination processor inside destination block reached via vertical intermediate block*/
//In example 1, DESVL = 1
pathLength_VI = SBVmin + 1 + VIBmin + 1 + DESVL /*Path length from source to destination via vertical intermediate mesh/block*/
//In example 1, pathLength_VI = 2 + 1 + 0 + 1 + 1 = 5
/*In example 1, pathLength_HI is greater than pathLength_VI. So else condition will be executed in below code.*/
if (pathLength_HI < pathLength_VI)
f1 = Assign the identified co-ordinate of the source block exit processor for pathLength_HI
f2 = Assign the identified co-ordinate of the intermediate block exit processor for pathLength_HI
f3 = Assign the identified co-ordinate of the destination for pathLength_HI
Calculate the path through horizontal intermediate block
return (path through horizontal intermediate block)
end if
else
f1 = Assign the identified co-ordinate of the source block exit processor for pathLength_VI
f2 = Assign the identified co-ordinate of the intermediate block exit processor for pathLength_VI
f3 = Assign the identified co-ordinate of the destination processor for pathLength_VI
Calculate the path through vertical intermediate block
return (path through vertical intermediate block)
end else
end if // end of if in step 1
//In example 1, f1 := (1, 1, 1, 3), f2 := (3, 1, 4, 1), f3 := (3, 4, 2, 4)
else
f1 = Assign the identified co-ordinate of the source block exit processor
f2 = 0
f3 = Assign the identified co-ordinate of the destination processor
Calculate the path without any intermediate block
return (source to destination path without intermediate block);
// No intermediate block required
end CalculateShortestPathMM (msg, msgSrc, msgDest)

Procedure CalculateSDLengthInBlk (msgSrc, msgDest, lengthOfBlock)
/* Calculate distance between source and destination inside a mesh*/
/* Let, msgSrc is (α_i, β_i, x_1, y_1), msgDest is (α_i, β_i, x_2, y_2) and lengthOfBlock = n */
/* Length between (x_1, y_1) and (x_2, y_2) through vertical self loop = Traversed distance to reach from the source column to vertical self loop (VSL) column+ Traversed distance from source to reach the nearest boundary along vertical + 1 (for VSL) + Traversed distance in horizontal from VSL to destination column + Traversed distance in vertical from current position to reach the destination. Length between (x_1, y_1) and (x_2, y_2) through horizontal self loop = Traversed distance to reach from source row to the horizontal self loop (HSL) row + Traversed distance from source to reach the nearest boundary along horizontal + 1 (for HSL) + Traversed distance in vertical from HSL to destination row + Traversed distance in horizontal from current position to reach the destination
The following notations will be used in the algorithm
pathLengthNormal = normal path length between (x_1, y_1) and (x_2, y_2)
pathLengthVSL = length between (x_1, y_1) and (x_2, y_2) through vertical self loop
pathLengthHSL = length between (x_1, y_1) and (x_2, y_2) through horizontal self loop*/

minPathLength = 0;
if (mod($x_1 - x_2$) $\leq$ n/2 and mod($y_1 - y_2$) $\leq$ n/2)
minPathLength = pathLengthNormal
return minPathLength;
end if
/*In example 1, for source block mod (2-1) $\leq$ 2 and mod (2-3) $\leq$ 2, Normal path length will be considered for source block.
For intermediate block mod (4-4) $\leq$ 2 and mod (1-1) $\leq$ 2, Normal path length will be considered for intermediate block
For destination block mod (1-2) $\leq$ 2 and mod (4-4) $\leq$ 2, Normal path length will be considered for destination block*/

```
 if (mod (x1 – x2) > n/2 and mod (y1 – y2) ≤ n/2)
 if (pathLengthVSL < pathLengthNormal)
    minPathLength = pathLengthVSL
 endif
 else
    minPathLength = pathLengthNormal
 end else
 return minPathLength;
end if
if (mod (x1 – x2) ≤ n/2 and mod (y1 – y2) > n/2)
 if (pathLengthHSL < pathLengthNormal)
    minPathLength = pathLengthVSL
 endif
 else
    minPathLength = pathLengthNormal
 end else
 return minPathLength;
end if
if (mod(x1 – x2) > n/2 and mod(y1 – y2) > n/2)
 if (pathLengthNormal < pathLengthVSL and
 pathLengthNormal < pathLengthHSL)
    minPathLength = pathLengthNormal
 else if (pathLengthVSL < pathLengthNormal
 and pathLengthVSL < pathLengthHSL)
    minPathLength = pathLengthVSL
 else if (pathLengthHSL < pathLengthNormal
 and pathLengthHSL < pathLengthVSL)
    minPathLength = pathLengthHSL
 end if
 return minPathLength;
end if
end CalculateSDLengthInBlk (msg, msgSrc,
msgDest)
```

6. PROOF OF DEADLOCK AND LIVELOCK FREEDOM

The proof of deadlock freedom for mesh architecture has been given in [3]. To avoid deadlock they have used three virtual channels c_0, c_1, c_2 as indicated in Table I. In case of Multi Mesh the same proof will also hold as the inter block links act just like the intra block links for the purpose of proof of deadlock avoidance. In case of inter block link failure message is normally routed through another exit processor to reach the destination. So, the algorithm is also livelock free.

Table 1: Usage of virtual Channels by misrouted message

Message Type	Channel	Used for
EW	c_0	row hops
EW	c_2	column hops in the south-to-north direction
EW	c_1	column hops in the north-to-south direction
NS	c_1	all hops
SN	c_2	all hops
WE	c_0	row hops
WE	c_1	column hops in the south-to-north direction
WE	c_2	column hops in the north-to-south direction

7. CONCLUSION AND FUTURE WORK

A fault tolerant minimal adaptive routing algorithm in Multi Mesh architecture has been proposed in this paper.
The proposed method in Multi-Mesh can be compared with the mesh structure, as Multi-Mesh has meshes (or blocks) as its constituent components. In a mesh having n^4 processors the diameter is $2(n^2 – 1)$, whereas for a Multi-Mesh of n^4 processors the diameter is 2n, And that's why the adaptive routing for any source to destination, even in presence of fault, will take $O(n)$ routing steps, not $O(n^2)$ as in case of mesh having the same number of processors.

To overcome deadlock condition three virtual channels c_0, c_1, c_2 are being used. This algorithm is also free from livelock as any message even in case of node or link fault is able to approach towards destination without routing through the same path repeatedly. The existing approach always uses a fixed path even if there might be shortest path available from a particular source to destination. Our own approach is able to measure multiple paths and among them the algorithm will choose the shortest path from the source to destination. The algorithm proposed here can tolerate a single fault in inter block link.
As a future work, the algorithm can be modified for a generalized MM network where the constituent meshes are of size m × n, and total number of processors is $m^2 n^2$.

References

[1] A. Datta and M. De, "Fault-tolerant and non adaptive wormhole routing algorithm for Multi-Mesh network," International Journal of Computer Technology and Electronics Engineering (IJCTEE), vol. 2, Issue 3, pp 1-9, 2012.
[2] R. V. Boppana and S. Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," IEEE Transactions on Computers, vol. 44, no. 7, pp 848-854, 1995.
[3] R. V. Boppana and S. Chalasani, "Adaptive fault-tolerant wormhole routing algorithms with low virtual channel requirements," International Symposium on Parallel Architectures, Algorithm and Networks, pp 214-221, 1994.
[4] Y. M. Boura and C. R. Das, "Fault-tolerant routing in mesh networks," Proc. International. Conf. Parallel Processing, vol. 1, pp. 106-109, 1995.
[5] R. V. Boppana and S. Chalasani, "A framework for designing deadlock free wormhole routing algorithm," IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 2, pp 169-183, 1996.

[6] S. Chalasani and R. V. Boppana ,"Communication in multicomputers with nonconvex faults," IEEE Transactions on Computers, vol. 46, pp 616-622, 1997.
[7] C.-L. Chen and G.-M. Chiu, "A fault-tolerant routing scheme for Meshes with nonconvex faults," IEEE Transactions on Parallel and Distributed Systems, vol. 12, pp 467-475, 2001.
[8] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," IEEE Transactions on Parallel and Distributed Systems, vol. 4, No. 4, pp 466-475, 1993.
[9] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," IEEE Transactions on Computers, vol. 4, pp 1320-1331, 1993.
[10] C. J. Glass and L. M. Ni, "Fault-tolerant wormhole routing in meshes," Proc. 23rd International Symposium. Fault-Tolerant Computing, pp. 240-249, 1993.
[11] J. Zhou and Francis C. M. Lau, "Multiphase Minimal Fault-Tolerant Worming Routing in 2D Meshes," Proc. ICPADS, pp. 323-330, 2001.
[12] C.-Tien Ho and L. Stockmeyer, "A new approach to fault-tolerant wormhole routing for mesh connected parallel computers," IEEE Transactions on Computers, vol. 53, no. 4, pp 427-438, April 2004.
[13] T. Schönwald, O.Bringman and W. Rosenstiel, "Region-based routing algorithm for network-on-chip architectures," Norchip, 19-20 Nov., pp. 1-4, 2007.
[14] Juan Xhen, Du Xu and Ling fu Xie, "A fault-tolerant routing protocol in 2D torus based on positive-first and negative-first turn models," Proc. Int'l Conference on Information Engineering and Computer Science (ICIECS), Wuhan, 19-20 Dec., pp. 1- 5, 2009.
[15] D. Das, M. De and B. P. Sinha, "A new network topology with multiple meshes," IEEE Transactions on Computers, vol. 48, no. 5, pp 536-544, 1999.
[16] B. P. Sinha, "Multi-Mesh an efficient topology for parallel processing", Proc. 9th International Parallel Processing Symposium, (IPPS), 1995, 17-21.
[17] Prasanta Mohapatra, "Wormhole Routing Techniques for Directly Connected Multicomputer Systems," ACM Computing Surveys (CSUR), Vol. 30 Issue 3, pp. 374-410, 1998.

Author

Amit Datta received B.Tech. degree in Information Technology from University of Kalyani in 2003 and M.Tech degree in Information Technology from University of Calcutta in 2005. He is currently working as an Assistant Consultant with Tata Consultancy Services Limited and having 7 years of experience in the field of Software Design and Development area. Simultaneously, he is continuing his PhD work at Department of Engineering & Technological Studies under University of Kalyani. His research interest is on Parallel Computing & Architectures, Fault-tolerant computing. He is also a member of IEEE.

Mallika De received the B. Sc. Degree in Physics from Calcutta University in 1973 and the M. Sc. Degree in Applied Mathematics from Jadavpur University in 1996. She received the Advanced Diploma in Computer Science and M. Tech in Computer Science in the year 1980 and 1985 respectively from Indian Statistical Institute, Calcutta. Her Ph.D. degree in Engineering was awarded in the year 1997 from Jadavpur University. She is a senior faculty of the Department of Engineering & Technological Studies at University of Kalyani, where she is serving for last 28 years. Her research interest includes Parallel Algorithms & Architectures, Fault-tolerant Computing, Image processing, Soft Computing and Quantum Cellular Automata. She is also a member of IEEE. Dr. De has authored/co-authored 28 refereed Journal Articles and nearly 30 Conference papers. She has worked as paper reviewer for few International Conferences such as Advanced Computing & Communication, High Performance Computing and Asian Test Symposium.