Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 10, Issue. 3, March 2021, pg.01 – 13

A Recursive Deterministic Routing Algorithm for Two Dimensional Mesh Network

Abdulelah G.F.Saif
Computer Science Department & King Khalid University, KSA
absaif@kku.edu.sa
DOI: 10.47760/ijcsmc.2021.v10i03.001

Abstract— Network performance in parallel systems, actually, depends on network topology and the used routing algorithm. Routing can be classified as centralized, distributed, deterministic or adaptive. Two Dimensional 2D Mesh is one of the popular general purpose networks due to its node degree is fixed and does not increase as the network size increases, symmetry, embedding of other networks, recursive structure, and excellent scalability. In this paper, we developed an optimal deterministic recursive routing algorithm for 2D Mesh network without wraparound link and enhanced it for 2D Mesh network with wraparound link (Torus). To measure the efficiency of the algorithms, they are applied on 4*7 2D Mesh network, and on 4*7 Torus, respectively. As a result, these routing algorithms are optimal in terms of time and space.

Keywords— parallel systems, interconnection networks, embedding, routing

I. INTRODUCTION

In distributed systems, network performance, actually, depends on network topology and the used routing algorithm [1, 5, 6, 7, 8, 9, 10, 11]. An optimal routing algorithm in any network would route information or message from a source to a destination along a shortest path [1, 11].

Routing can be classified in several ways. The routing algorithm can be centralized or distributed [1, 2, 10]. In centralized routing, a single processing element determines the shortest path form a given source node to a given destination node. Then, the message is sent along that path. Finding the shortest path by a single central processing element is very slow and leads to a major bottleneck. On the other hand, if every source node takes the responsibility of computing the shortest path to the destination and sends that path along with the message to guide it through the intermediate nodes, the bottleneck problem is avoided, but traffic is increased and routing remains relatively slow.

In distributed routing, however, all intermediate nodes on the shortest path cooperate to find the shortest path using the destination address. Therefore, each intermediate node needs only the destination address to determine which neighbor falls on the shortest path to a given destination. In the routing algorithm, each intermediate node determines, in constant time, which of its neighbors will receive the message. The routing of a message can be viewed as a sequence of changes made on the source address label to become the destination address label. These changes are done at every intermediate node on the path. When the message is received by an intermediate node, it will consider itself as a new source.

Routing can also be classified as deterministic or adaptive [2]. In deterministic routing, a path that massage follows depends only on its source and destination nodes. In adaptive routing, for a given source and destination, the path that a massage follows depends on the dynamic conditions of the network such as faulty or congested channels.

2D mesh is one of the popular general purpose networks due to its node degree is fixed and does not increase as the network size increases, recursive structure, and excellent scalability. In this paper, we developed a point-to-point, optimal, deterministic and recursive routing algorithm for 2D Mesh Network, with and without wraparound link. Routing in this mode is optimal in time and space.

The paper is organized as: section 2 meshes, section 3 routing algorithm for 2D Mesh without wraparound link, section 4 routing algorithm for 2D Mesh with wraparound link(Torus).

II. MESHES

In a mesh network, the nodes are arranged in a k dimensional lattice of width w, giving a total of w^k nodes or w*w in the case of 2D mesh. Usually k=1 (linear array) or k=2 (2D array/2D mesh). Communication is allowed only between neighboring nodes. All interior nodes are connected to 2k other nodes [3]. A two-dimensional mesh illustrated in Fig.1(a) is an extension of the linear array to two-dimensions. Each dimension has p nodes with a node identified by a two-tuple (i, j). Every node (except those on the periphery) is connected to four other nodes whose indices differ in any dimension by one. A variety of regularly structured computations map very naturally to a 2D mesh. For this reason, 2D meshes were often used as in parallel machines [4]. Some data transfers in 2D mesh may require 2((w*w)½-1) links to be traversed. This can be reduced by using wraparound connections between nodes on same row or column as in Fig.1(b) or when k=3 (three dimensions) as in Fig.1(c) [3].

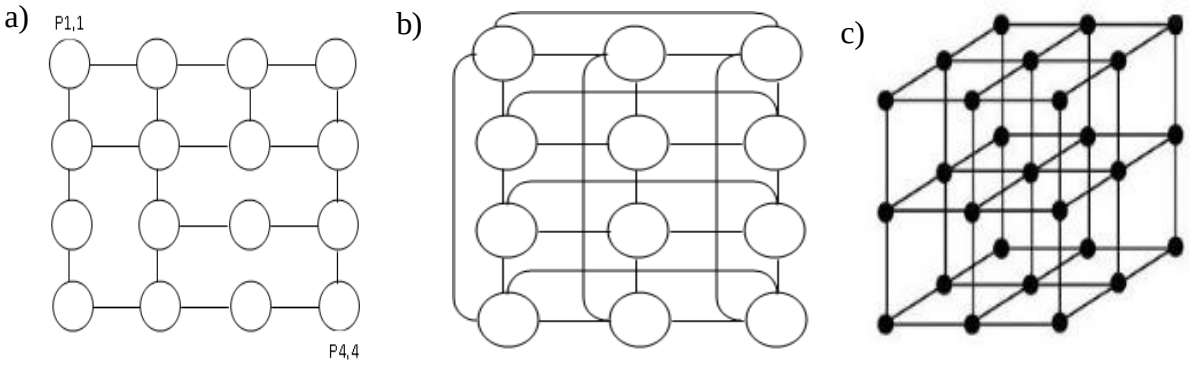

Fig.1 Meshes (a) k=2 , w=4 without wraparound, (b) k=2 , w=3 with wraparound, and (c) k=3 ,w=3 with wraparound.

III. ROUTING ALGORITHM FOR 2D MESH WITHOUT WRAPAROUND LINK

In order to explain this algorithm, we use the numbering scheme as follows. Each node in the 2D mesh is identified by a pair (x.y), where x denotes the row in which the node exists, and y denotes the column of the node in the row as in Fig.2. A node with the address 1.1 is the first node that exists at row 1. 1.2 refers to the second node that exists at row 1, and so on.

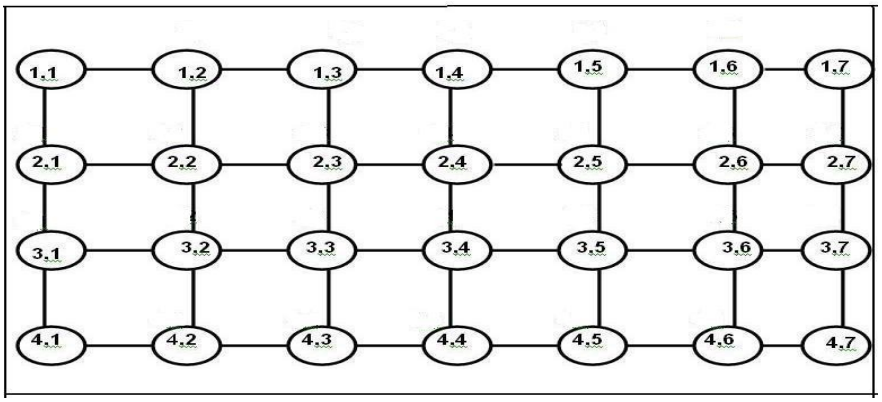

Fig.2 Addressing nodes in 2D mesh.

In the designed recursive routing algorithm for 2D mesh as shown in Fig.3, the movement from one node to another can be done by using the following three cases:

CASE 1: If (xs = xd) : moveHorizontal(xs,ys, xd,yd).
Here, we have two directions, moveHorizontal/Left-to-Right and moveHorizontal/Right-to- Left.

CASE 2: If (xs < xd) : moveDown(xs,ys, xd,yd)
Here, we have one direction, moveDown.

CASE 3: If (xs > xd) : moveUp(xs,ys,xd,yd).
Here, we have one direction, moveUp.

Where, xs is the row of source node, ys is the column of source node in this row, xd is the row of destination node and yd is the column of destination node in this row. One of the previous cases will be called recursively until the destination has been reached. When we will using these algorithm, we will start by calling moveHorizontal(xs,ys, xd,yd) and from which the progress continues until the destination is reached.


```
moveHorizontal( xs, ys, xd, yd)
    {
        if (xs == xd)
        {
            if (yd > ys)
            {
              moveHorizontal(xs, ys + 1, xd, yd)
            }
            else if (yd < ys)
            {
              moveHorizontal(xs, ys - 1, xd, yd)
            }
            else
            {
              Destination_Reached
            }
        }
        else
            move_Down(xs, ys, xd, yd)
    }

move_Down( xs,  ys,  xd,  yd)
    {
        if (xs < xd)

            move_ Down (xs + 1, ys, xd, yd)

        else
            move_Up(xs, ys, xd, yd)
    }

move_Up( xs,  ys,  xd,  yd)
    {
        if (xs > xd)
            {
                move_Up(xs - 1, ys, xd, yd)
            }
            else
            {
                moveHorizontal(xs, ys, xd, yd)
            }
    }
```

Fig.3 The Recursive Routing algorithm for 2D Mesh M(i,j) without wraparound link.

A. Examples on the Routing Algorithm, and Discussions

The following examples explain the above routing cases. We consider a network of a 4 * 4 2D Mesh; i.e. M(4,7) as shown in Fig.4.

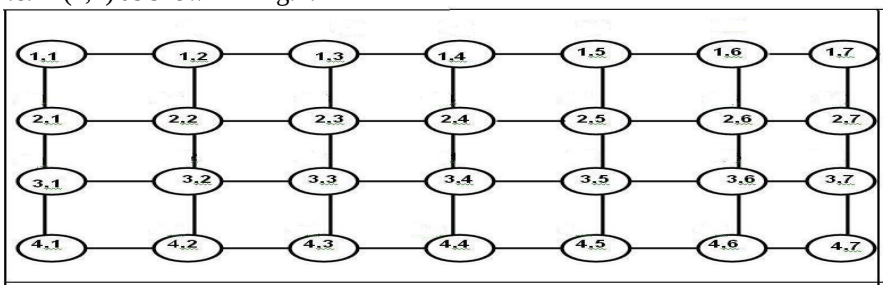

Fig.. 4 2D mesh M(4,7).

moveHorizontal-Left-to-Right

Let (2,1) be the address of the source node and (2,7) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 1 (moveHorizontal- Left-to-Right). To reach the destination, the algorithm passes through six steps; i.e., (2,1)→(2,2)→(2,3)→(2,4)→(2,5)→(2,6)→(2,7) as shown in Fig.5.

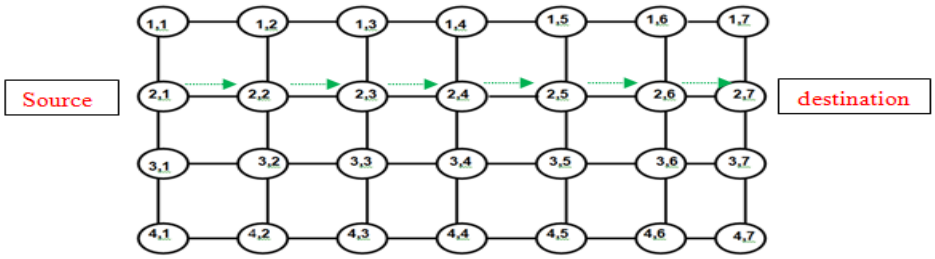

Fig.. 5 moveHorizontal-Left-to-Right.

moveHorizontal-Right-to-left

Let (2,7) be the address of the source node and (2,1) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 1 (moveHorizontal- Right-to-Left). To reach the destination, the algorithm passes through six steps; i.e., (2,7)→(2,6)→(2,5)→(2,4)→(2,3)→(2,2)→(2,1) as shown in Fig.6.

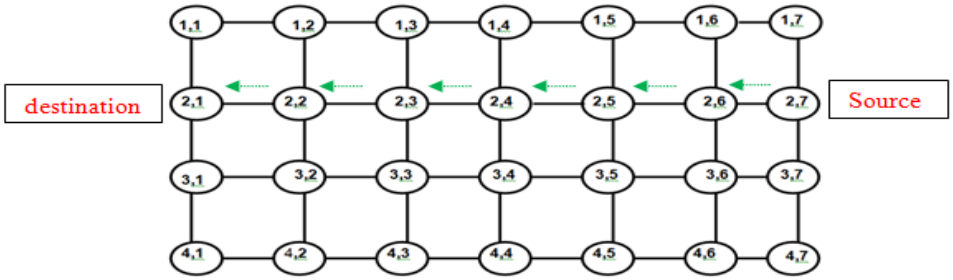

Fig.6 moveHorizontal-Right-to-Left.

moveDown

Let (1,4) be the address of the source node and (4,4) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 2 (moveDown). To reach the destination, the algorithm passes through three steps; i.e.(1,4)→(2,4)→(3,4)→(4,4) as shown in Fig.7.

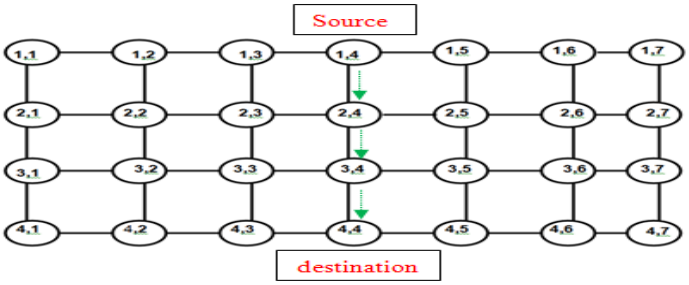

Fig.7 moveDown.

moveUp

Let (4,4) be the address of the source node and (1,4) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 3 (moveUp). To reach the destination, the algorithm passes through three steps; i.e. (4,4)→(3,4)→(2,4)→(1,4) as shown in Fig.8.

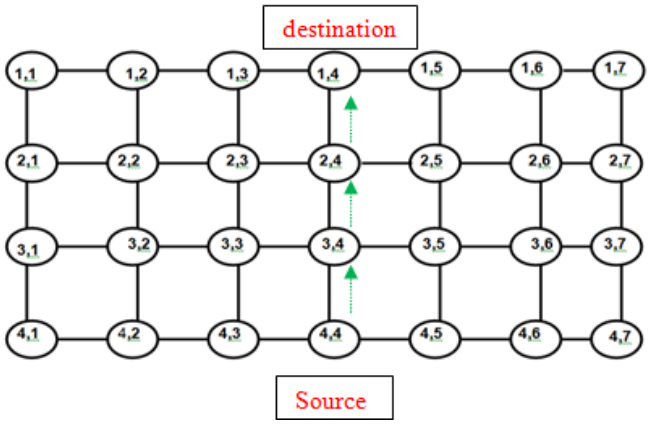

Fig.8 moveUp.

moveDown/moveHorizontal-Left-to-Right

Let (1,2) be the address of the source node and (4,5) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 2 (moveDown), then CASE 1 (moveHorizontal-Left-to-Right). To reach the destination, the algorithm passes through six steps; i.e., (1,2)→(2,2)→(3,2)→(4,2)→(4,3)→(4,4)→(4,5) as shown in Fig.9.

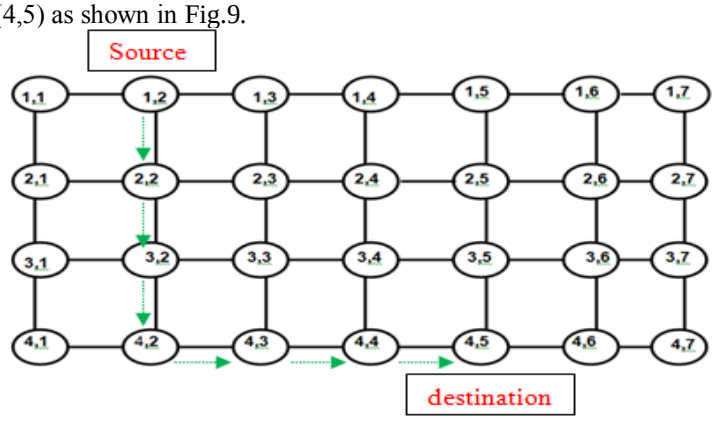

Fig.9 moveDown/moveHorizontal-Left-to-Right.

moveDown/moveHorizontal-Right-to-Left

Let (1,5) be the address of the source node and (4,2) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 2 (moveDown), then CASE 1 (moveHorizontal- Right-to-Left). To reach the destination, the algorithm passes through six steps; i.e., (1,5)→(2,5)→(3,5)→(4,5)→(4,4)→(4,3)→(4,2) as shown in Fig.10.

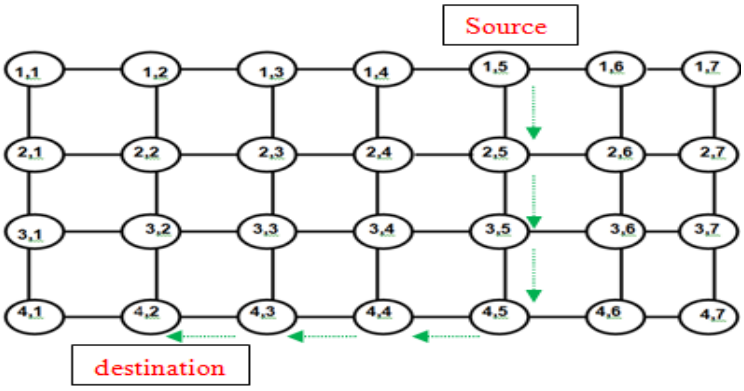

Fig.10 moveDown/moveHorizontal-Right-to-Left.

moveUp/moveHorizontal-Left-to-Right

Let (4,2) be the address of the source node and (1,5) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 3 (moveUp), then CASE 1 (moveHorizontal- Left-to-Right). To reach the destination, the algorithm passes through six steps; i.e., (4,2)→(3,2)→(2,2)→(1,2)→(1,3)→(1,4)→(1,5) as shown in Fig.11.

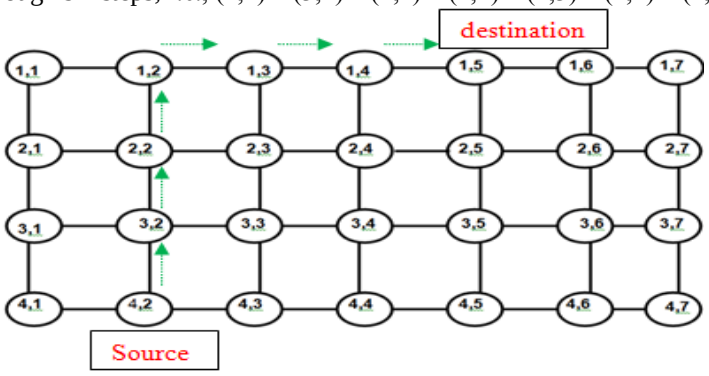

Fig.11 moveUp/moveHorizontal-Left-to-Right.

moveUp/moveHorizontal-Right-to-Left

Let (4,5) be the address of the source node and (1,2) the address of destination node. Applying the algorithm in Fig.3 will trigger CASE 3 (moveUp), then CASE 1 (moveHorizontal- Right-to-Left). To reach the destination, the algorithm passes through six steps; i.e., (4,5)→(3,5)→(2,5)→(1,5)→(1,4)→(1,3)→(1,2) as shown in Fig.12.

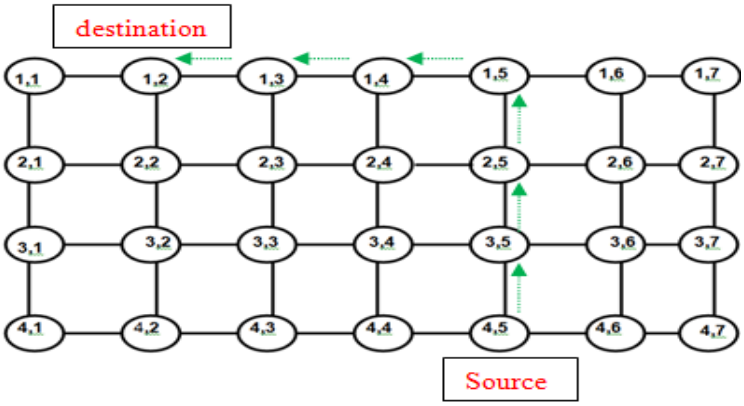

Fig.12 moveUp/moveHorizontal-Right-to-Left.

IV. Routing Algorithm for 2D Mesh with Wraparound Link(Torus)

This algorithm is the enhancement of the previous algorithm of 2D mesh without wraparound link for torus. In order to explain this algorithm, we use the numbering scheme similar to the above (numbering scheme for 2D mesh without wraparound link) as follows. Each node in the torus is identified by a pair (x.y), where x denotes the row in which the node exists, and y denotes the column of the node in the row as in Fig.13. A node with the address 1.1 is the first node that exists at row 1. 1.2 refers to the second node that exists at row 1, and so on.

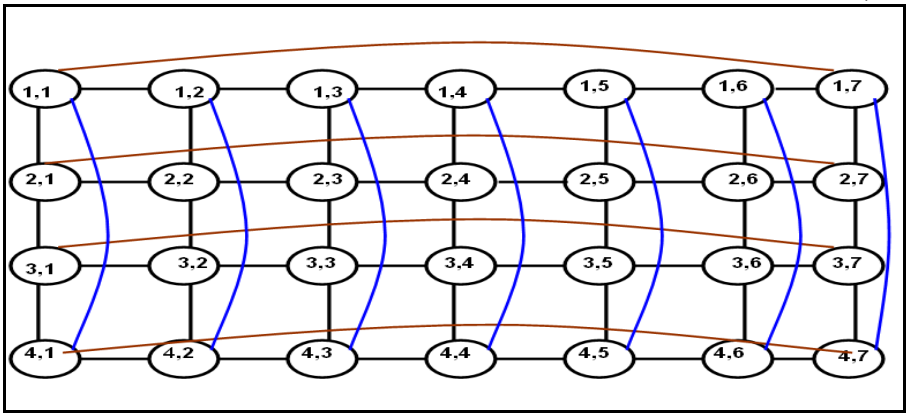

Fig.13 Addressing nodes in torus.

In the designed routing algorithm for torus as shown in Fig.14, the movement from one node to another can be done by using the following three cases:

CASE 1: If (xs = xd) : moveHorizontal(xs,ys, xd,yd).

Here, we have four directions, moveHorizontal/Left-to-Right across the original links, moveHorizontal/Right-to- Left across the original links, moveHorizontal/Left-to-Right across wraparound link and moveHorizontal/Right-to- Left across wraparound link.

CASE 2: If (xs < xd) : moveDown(xs,ys, xd,yd)

Here, we have two directions, moveDown across the original links and moveDown across wraparound link.

CASE 3: If (xs > xd) : moveUp(xs,ys,xd,yd).

Here, we have two directions, moveUp across the original links and moveUp across wraparound link.

Where, xs is the row of source node, ys is the column of source node in this row, xd is the row of destination node and yd is the column of destination node in this row. One of the previous cases will be called recursively until the destination has been reached. When we will using these algorithm, we will start by calling moveHorizontal(xs,ys, xd,yd) and from which the progress continues until the destination is reached.

A. Important Notes about This Algorithm:

Let the address of the last column (column_last_address) equals to 4*i-1, the address of the first column (column_first_address) equals to 1 and the address of the last row (Row_last_address) equals to 2*i, where i=1.2,….

Conditions in moveHorizontal(xs,ys,xd,yd) function:

1. if **(**

(ys + (column_last_address - yd)) < yd – ys

//This part of the condition checks, if distance between the source and destination across the wraparound link is less than the distance across the original links when the move from left to right.

OR

(((column_last_address - ys + 1) + (yd - 1)) <ys - yd)

//This part of the condition checks, if distance between the source and the destination across the wraparound link is less than the distance across the original links when the move from right to left.

)

If the above condition is true, then the movement is on the wraparound link, otherwise the movement is on original links i.e. as with moveHorizontal(xs,ys,xd,yd) function in the 2D mesh without wraparound link.

2. if(ys <= (column_last_address) / 2)

//Divide the path from the column_first_address to the column_last_address into two divisions and check which division ys fall in.

If this condition is true, then ys fall in the left division, otherwise ys fall in the right division.

Condition in moveDown(xs,ys, xd,yd) function.

1. if ((Row_last_address - xd+1) < xd - xs)

/This condition checks, if distance between the source and the destination across the wraparound link is less than the distance across the original links when the move from top to down.

If this condition is true then the movement is on the wraparound link, otherwise the movement is on original links i.e. as with moveDown(xs,ys,xd,yd) function in the 2D mesh without wraparound link.

Condition in moveUp(xs,ys, xd,yd) function.

1. if (((Row_last_address - xs + 1) + (xd - 1)) < xs - xd)

//This condition checks, if distance between the source and the destination across the wraparound link is less than the distance across the original links when the move from bottom to up.

If this condition is true then the movement is on the wraparound link, otherwise the movement is on original links i.e. as with moveUp(xs,ys,xd,yd) function in the 2D mesh without wraparound link.


```
moveHorizontal(xs,ys,xd,yd)
  {
   if (xs == xd)
    {
     if (
          (ys + (column_last_address - yd)) < yd - ys
                    OR
          (((column_last_address - ys + 1) + (yd - 1)) <ys - yd)
        )
         {
          if(ys <= (column_last_address) / 2)
            {
             if (ys == 1)
              {  //move right across the wraproundlink
                 moveHorizontal(xs,ys +column_last_address,xd, yd)
              }
             else
              {   //move left across the orginal links until ys=1
                  moveHorizontal(xs, ys - 1, xd, yd)
              }
            }
         else
            {
             if (ys == column_last_address)
              {   //move left across the wraproundlink
                  moveHorizontal(xs,ys -(column_last_address-1),xd,yd)
              }
             else
              {   //move right across the orginal links until ys=
                  column_last_address.
                  moveHorizontal(xs, ys + 1, xd, yd)
              }
            }
         }
        else
         {   //this part works as moveHorizontal(xs,ys,xd,yd) function in 2D
                       mesh without wraparound link.
          if (yd > ys)
            {
                moveHorizontal(xs, ys + 1, xd, yd)
            }
           else if (yd < ys)
                {
                   moveHorizontal(xs, ys - 1, xd, yd)
                }
                else
                {
                   Distination_Reached.
                }
        }
    }
    else
         move_Down(xs, ys, xd, yd)
}
```



```
move_Down(xs, ys,xd,yd)
 {
 if (xs < xd)
  {
   if ((Row_last_address - xd+1) < xd - xs)
    {
        if (xs == 1 )
         {   // move down across the wraparound link
             move_Down(xs + Row_last_address -1, ys, xd, yd)
         }
        else
         {   //move up across the original links untill xs=1
             move_Down(xs - 1, ys, xd, yd)
         }
    }
  else
    {   //move dwon as with 2D mesh without wraparound link
        move_Down(xs + 1, ys, xd, yd)
    }
  }
else
  move_Up(xs, ys, xd, yd)
}
move_up(xs, ys, xd, yd)
 {
   if (xs > xd)
    {
     if (((Row_last_address - xs + 1) + (xd - 1)) < xs - xd)
      {
       if (xs == Row_last_address)
        {   // move up across the wraparound link
            move_up(1, ys, xd, yd)
        }
       else
        {   //move up across the original links untill xs= Row_last_address.
            move_up(xs + 1, ys, xd, yd)
        }
      }
     else
       {   //move up as with 2D mesh without wraparound link
           move_up(xs - 1, ys, xd, yd)
       }
    }
   else
    {
      moveHorizontal(xs, ys, xd, yd)
    }
 }
```

Fig.14 Routing algorithm for 2D Mesh M(i,j) with wraparound link(Torus).

B. Examples on the Routing Algorithm, and Discussions

The following examples explain the above routing cases. We consider a network of a 4 * 4 2D Torus; i.e. M(4,7) as shown in Fig.15.

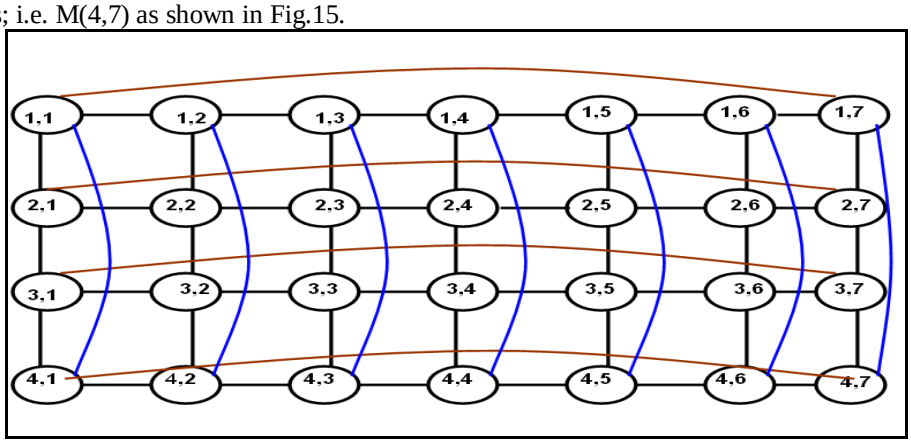

Fig.15 2D Torus M(4,7).

moveHorizontal-Left-to-Right

Let (1,2) be the address of the source node and (1,6) the address of destination node. Applying the algorithm in Fig.14 will trigger CASE 1 (moveHorizontal- Right -to-left across the original link, then moveHorizontal-Left-to-Right across the wraparound link, then moveHorizontal- Right -to-left across the original link). To reach the destination, the algorithm passes through three steps; i.e., (1,2)→(1,1)→(1,7)→(1,6) as shown in Fig.16.

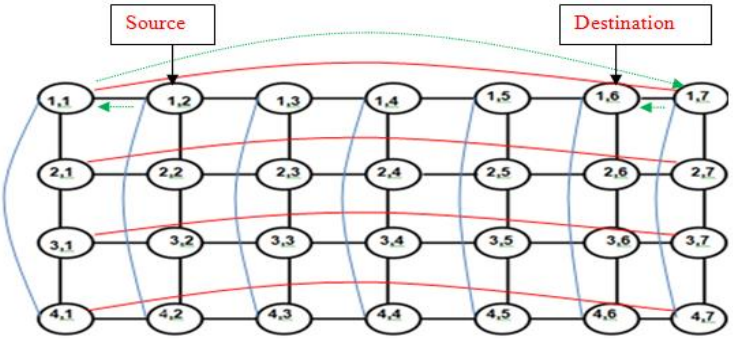

Fig.16 The path from node (1,2) to node (1,6) (horizontal).

moveHorizontal-Right-to-Left

Let (1,6) be the address of the source node and (1,2) the address of destination node. Applying the algorithm in Fig.14 will trigger CASE 1 (moveHorizontal-Left -to- Right across the original link, then moveHorizontal-Right -to-Left across the wraparound link, then moveHorizontal-Left-to-Right across the original link). To reach the destination, the algorithm passes through three steps; i.e., (1,6)→(1,7)→(1,1)→(1,2) as shown in Fig.17.

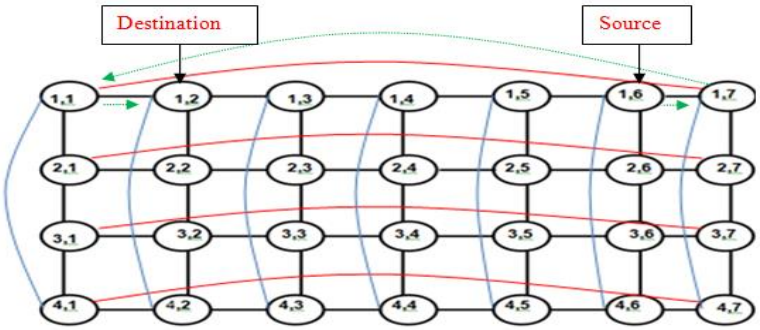

Fig.17 The path from node (1,6) to node (1,2) (horizontal).

moveDown

Let (1,2) be the address of the source node and (4,2) the address of destination node. Applying the algorithm in Fig.14 will trigger CASE 2 (moveDown across the wraparound link). To reach the destination, the algorithm passes through one step; i.e., (1,2)→(4,2) as shown in Fig.18.

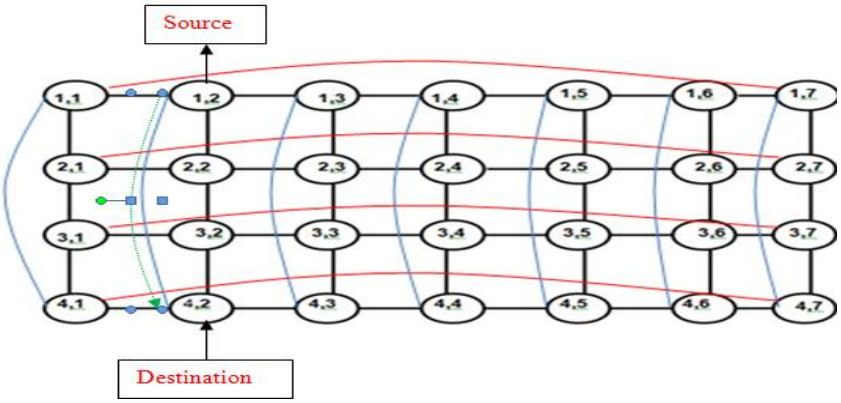

Fig.18 The path from node (1,2) to node (4,2).

moveUp

Let (4,2) be the address of the source node and (1,2) the address of destination node. Applying the algorithm in Fig.14 will trigger CASE 3 (moveUp across the wraparound link). To reach the destination, the algorithm passes through one step; i.e., (4,2)→(1,2) as shown in Fig.19.

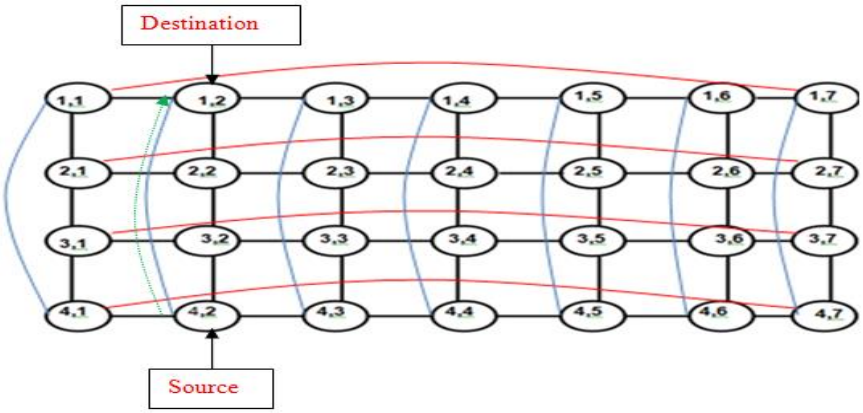

Fig.19 The path from node (4,2) to node (1,2).

V. DISCUSSION AND CONCLUSION

Network performance in distributed systems, actually, depends on network topology and the used routing algorithm. Routing can be classified as centralized, distributed, deterministic or adaptive. Two Dimensional 2D Mesh is one of the popular general purpose networks due to its node degree is fixed and does not increase as the network size increases, symmetry, embedding of other networks, recursive structure, and excellent scalability. In this paper, we developed an optimal deterministic recursive routing algorithm for 2D Mesh network without wraparound link and enhanced it for 2D Mesh network with wraparound link (Torus). To measure the efficiency of the algorithms, they are applied on 4*4 2D Mesh network, and on 4*4 2D Torus, respectively. As a result, these routing algorithms are optimal in terms of time and space.

For future work, we will enhance a recursive deterministic routing algorithm for 2D mesh into adaptive one, deadlock-free.

REFERENCES

[1] Sharieh A., Qatawneh M., Almobaideen W., and Sleit A., "Hex- Cell: Modeling, Topological Properties and Routing Algorithm", European Journal of Scientific Research, Vol.22, No. 3, pp. 457-468, 2008.

[2] Tahir K., "Performance Analysis of XY Routing Algorithm Using 2-D Mesh (M×N) Topology", Master project, university of Victoria, 2011.

[3] Available via https://www.cm.cf.ac.uk/Parallel/Year2/section5.html (cited 15.10.2019).

[4] Grama A., Gupta A., Karypis G., and Kumar V., "*Introduction to Parallel Computing*", Addison Wesley, pp. 12-72, 2003.

[5] Thiem C., Nonmember, Kenji K.," LEF: An Effective Routing Algorithm for Two-Dimensional Meshes", IEICE TRANS. INF. & SYST., Vol.E102–D, No.10, October 2019.

[6] Dipika D., John J., Shirshendu D., Hemangee K. Kapoora," Cost Effective Routing Techniques in 2D Mesh NoC using On-Chip Transmission Lines", Journal of Parallel and Distributed Computing · September 2018.

[7] Akash P., Pardeep K., Nitin N.," Level Based Routing Using Dynamic Programming for 2D Mesh", Cybernetics and Information Technologies, Vol. 17, No 2, Sofia, 2017.

[8] Aleksandr Yu. Romnov, Evgeny V.Lezhnev, Aleksandr Yu. Glukhikh, Aleksandr A. Amrikanov," Development of routing algorithms in networks-on-chip based on two-dimensional optimal circulant topologies", Heliyon Journal, Vol. 6, 2020.

[9] Rose G. Kunthara, Neethu K, Rekha K. James, Simi Z. Sleeba ,John Jose," DoLaR: Double Layer Routing for Bufferless Mesh Network-on-Chip", 2019 IEEE Region 10 Conference (TENCON 2019).

[10] Parag P., Jayesh k. Dalal, Sumant K.," Performance Comparison of XY, OE and DY Ad Routing Algorithm by Load Variation Analysis of 2-Dimensional Mesh Topology Based Network-on-Chip", BVICAM's International Journal of Information Technology, 2012.
[11] Qatawneh M. , Hebatallah K..," New Routing Algorithm for Hex-Cell Network", International Journal of Future Generation Communication and Networking ", Vol. 8, No. 2, pp. 295-306, 2015.

