A Recursion-based Broadcast Paradigm in Wormhole Routed Mesh/Torus Networks

Xiaotong Zhuang Vincenzo Liberatore*

College of Computing
801 Atlantic Drive
Georgia Institute of Technology
Atlanta, GA 30332-0280
xt2000@cc.gatech.edu

*Dept EECS
Case Western Reserve University
10900 Euclid Avenue
Cleveland OH 44106.
vxl11@eecs.cwru.edu

Abstract: A novel broadcast technique for wormhole-routed mesh and torus parallel computers based on recursion is presented in this paper. It works by partitioning the graph into several subgraphs similar to the original one, and identifying a *characteristic low-dimensional subgraph* from these subgraphs. The source message is first scattered in the characteristic low-dimensional subgraph of the original graph, then through a O(1) number of message transfer and sharing operations, the characteristic low-dimensional subgraphs in each subgraph get the full source message. This procedure continues recursively until the minimum subgraph (a single node) gets all the source message. We have applied this general paradigm to several different cases including the one-port/all-port model in mesh/torus with 2 or higher dimension. The network topology can be square or non-square, the source node can be located in the corner or not. Comparing to the previous results, our paradigm reduces broadcast latency and is simpler. We also present an analytical comparison of the algorithm against the optimum.

Keyword: Mesh/Torus, wormhole routing, broadcast algorithm, Massive parallel computer

1 Introduction

Massive parallel computers (MPC) are playing an increasing important role in scientific computation and high speed computing applications such as weather forecast and high performance servers. MPCs are usually organized as an ensemble of nodes, each of them equipped with its own processor, local memory, and other supporting devices, The nodes are interconnected using a variety of topologies. Among them, the most commonly used are meshes, tori, hypercubes and trees.

Wormhole routing is a fundamental routing mechanism in modern parallel computers, mainly due to its low communication latency, which, in the absence of link contention, is almost insensitive to routing distance[1]. Wormhole routing has been adopted by many new generation parallel computers like Cray T3D,T3E, Intel Touchstone, and MIT J-machine[6][7][8], in which fast switching is a critical objective.

Broadcast is frequently invoked as one of the most essential communication operations in massive parallel computer algorithms like parallel graph algorithms, fast Fourier transform, and barrier synchronization. Several algorithms have been put forward by researchers to effectively achieve fast broadcast [2][3][4][5][9][10].

The basic broadcast algorithms are Recursive Division (RD) and Scatter Collect (SC)[9][10]. In Recursive Division, the message is duplicated at an exponential rate, and during each round the whole message is transferred, which results in a relatively small number of transfer but a long transfer latency per transfer. On the contrary, the Scatter-Collect algorithm starts by scattering the original message along one row and then to each column. Finally, SC collects message flints in each row/column in circular style. It needs more transfers but less time per transfer. Performance analyses show that RD is optimal for length messages, but as message length becomes longer, its performance substantially degrades. SC is good only for sufficiently long messages, which makes its application limited. Other tree-based approaches utilize multiple spanning trees in parallel for transmission, as in Johnson and Ho[4] and Bermond et al[5]. Since the number of spanning trees is fixed and all the nodes must exist in the original topology, tree-based approaches are more suitable for store-forward networks. A recent paper by Yu-Chee Tseng [2] introduced the Network Partition (NP) approach for wormhole routing. NP combines RD, SC with other algorithms, and it can adapt to different message length by tuning a certain parameter d. Also, NP is applicable to the all-port model. However, the choice of the parameter d according to different message lengths makes the implementation complicated, and due to the same reasons, it should not be used in all-port model. The work by San-Yuan Wang, Yu-Chee Tseng[3] focuses on tori under the all-port model. They developed a detailed study on mathematical foundations and algorithms. In that algorithm, the message is transferred to lines and then to planes. However, their performance metric is the steps needed to complete the broadcast, which is only a rough estimate of latency.

We design an algorithm which we call Recursion Based (RB) algorithm. By identifying the *characteristic low-dimensional subgraph* of an interconnection graph, we get a general purpose algorithm applicable to many network topologies. Our results show considerable improvements over all other algorithms for middle-sized messages. For 2 or more dimensional mesh/torus, the characteristic low-dimensional subgraph is quite simple, which makes the implementation easy.

The rest of this paper is organized as follows: Preliminaries are given in Section 2. Section 3 presents the recursive algorithm in general form. Section 4 and Section 5 give its application to 2 dimensional mesh/torus network. Section 6 presents its application to non-square 2 dimensional mesh/torus. Section 7 extends it to 3-dimenstional mesh/torus. The all-port model is considered in Section 8. Section 9 provides several analytical results about the broadcast algorithm. Conclusions are drawn in Section 10.

2 Preliminaries

2.1 System model

In wormhole routing, each message is divided into small *flits*. The header guides the whole message traveling through the network in a pipelined fashion. If the header gets stalled, all the following flits will be stalled along the path. The major advantage of wormhole routing is the transfer latency is almost insensitive to the distance [1].

In a wormhole-routed network, the underlying network is organized as a number of inter-connected routers. Router connection determines the network topology. There are external channels between any two neighboring nodes. Each node is linked by several internal channels to a local processor that has a local memory and other supporting devices. Fig 1 gives an example.

Fig 1

Interconnections can be categorized along three dimensions: the topology of the underlying network, the property of the external channels, and the property of the internal channels.

1) *Topology of the network* A number of topologies have been studied including meshes, tori and hypercubes. Since wormhole routing technology is distance insensitive, meshes and tori have become popular for their simple structure. Some super-computers use high dimensional mesh/torus to satisfy their performance requirements.
2) *The external channels between neighboring routers* There are two kinds of links between routers and their neighbors: unidirectional and bidirectional. Unidirectional channels can transmit messages only in one direction, while the bidirectional channels can transmit messages in both directions between the two neighboring nodes.
3) *The internal channels between a router and its local processing unit* Some internal channels can be used for input, while others are for output. The *port model* refers to the number of internal channels or the concurrent transmission ability of the processing node. In the case of

an *all-port* model, every external channel has its corresponding internal channel, which permits the node to send and receive on all external channels simultaneously from its neighbors. The *one-port* model has only one pair of input/output internal channels.

The following graph illustrates a bidirectional node with four pairs of external channels and working in the one-port model. Such unit can be used as a node in a 2D torus.

Fig 2

2.2 Assumptions

We make the following assumptions throughout the paper:

1. Each input/output channel can only be occupied by one flow (All the algorithms discussed in this paper are contention-free). The router can simultaneously direct several transmission flows provided the channels permit.
2. The processing unit can send and receive in all its internal channels at a speed no less than the external channel, i.e. internal channels are not the bottlenecks.
3. In the case of meshes and tori, if not otherwise stated, the network uses XY-routing; i.e. a message is routed within a row to the column that contains the destination node and then to the desired row in this column.

A message transferring from one node to another occupies the entire path between the two nodes until the transfer finishes.

2.3 Latency calculation

The latency for sending L bytes message from a source node to a destination node with distance d can be formulated as a linear function $T_s+dT_f+LT_c$ of L and d, where T_s is the startup latency, T_f is the time to transfer one flit, T_c is the time to transfer one byte. Observe that, T_f is proportional to the length of each flit, which can be chosen as a very small value, regardless of the length of the message. So, we simply write the latency for one transmission as T_s+LT_c[1]. Broadcast algorithms are sequence of point-to-point transmissions. For the purpose of comparison, we denote the total latency of a broadcast algorithm as $\alpha T_s+\beta LT_c$, so that a simple comparison of α and β allows us to compare different broadcast algorithms. Observe that α and β can be calculated separately: α is the total number of concurrent transfers in a broadcast algorithm (each concurrent point-to-point transmission contribute one T_s) and βL is the sum of the maximal message lengths among all the concurrent transfers.

3 The recursive algorithm

3.1 Message exchange operations

In our paper, the original message is always divided into m equal-sized *flints*. Flints will not be further divided during the operations of broadcast, i.e, they are basic units of the transmission of our algorithm. Note here the difference between flint and flit. Flints may be further divided into flits during the wormhole routing to make the transfer time independent of distance. However, from our point of view, flits are system specific. The broadcast procedure consists of a series of *steps,* during which some of the nodes will act concurrently, either to send or to receive (or both) message flints. For a better understanding of the algorithm, we hereby define two operations on the message flints contained in a node.

1. *Message transfer.* One node sends out all or part of the flints it has to another node, so the receiving node gets part of or all flints from the first node. To simplify the description of the algorithm, we assume that the first node does not have the transferred flints after the message

transfer, although this is not true in fact. However, if the broadcast algorithm can work under this assumption, then it will work even without it.

2. *Message sharing.* Two nodes send message flints to each other. After message sharing, they get flints from each other without losing their own.

We will show later on how message transfer and sharing are implemented. Both of them can be implemented by O(1) point-to-point transmissions.

3.2 Algorithm description

Suppose the original connection topology is a graph G(V,E). First, we identify a *characteristic low-dimensional subgraph* of G which we call H. The subgraph H can be any graph that meets certain recursion requirements. In mesh/torus, we often choose the diagonals to be graph H, and thus we will frequently refer to the algorithm as the *Diagonal Algorithm*. Graph H will also called the H-graph of G.

Graph G is divided into a set of *G-graphs*: G_1, G_2,…G_p, where G_i (i=1..p, p≥2 is a constant) satisfies:

1. $\cup\ G_i = G$
2. $G_i \cap G_k = \varnothing$ $(1 \le i \ne k \le p)$
3. The G_i 's are isomorphic to each other.
4. Each G_i contains a subgraph H_i, which is similar to H and has 1/p the size of H.

Here, the $\cup$ and $\cap$ operations mean the union and intersection of the node sets of the graphs. Furthermore, to work recursively without channel contentions, the H graph must satisfy certain requirements, which will be specified later on.

The presence of H_i in the subgraph G_i originates our recursive algorithm on the graph. Obviously, each subgraph G_i's can also contain p smaller subgraphs. They will be called the G-graphs of each G_i.

The algorithm proceeds in the following two major stages.

Stage 1: Spread the source message into the H-graph of G, so that each node in graph H receives an equal portion of the source message. These portions add up to the whole source message.

Stage 2: Stage 2 consists of a series of *recursive rounds*. Each recursive round works in all the subgraph G_i 's and transfers the message to the H-graphs of all their G-graphs. We will show that each recursive round can be completed within O(1) point-to-point communications.

During each recursive round, we are working simultaneously on all the G-graph of the same size, then during the next round, we are working on the G-graph of previous G-graph. Recursively, the algorithm continues until the G-graph becomes a single node and completes within O(log(#Node(G))) rounds.

3.3 Latency calculation

We will show that stage 1 can be performed using a split-scattering algorithm. Split scattering is similar to the first stage of Scatter-Collect algorithm [10]. An original message is partitioned and half of its flints are transferred to another node, so that each of the two nodes get ½ of the original message, then the two nodes act like new source to further split and scatter the message to 4 nodes, each of them contain ¼ of the original. The procedure continues until all the nodes get some portions of the message. So, stage 1 needs log (#Node(H)) steps, where #Node(H) is the number of nodes in H. Then, the number of all steps is α=log (#Node(H))+log (#Node(G)) * O(1). Here, log(#Node(G)) is the total number of recursive rounds in stage 2 and we'll implement each round in O(1) concurrent transfers or sharings. Since H is a subgraph of G, α=O(log(#Node(G))).

The transfer size of message portions in stage 1 decreases at an exponential rate starting from L. In stage 2, the size of the H-graph is decreasing exponentially after each recursive round, or the flint size in each node of H-graph increases exponentially, which is proportional to the transfer size of message portions in each recursive round. Therefore, $\beta=O(1)$.

In the following chapters, we show that both α and β can be kept small for various topology networks and channel models and the algorithm is contention free in all cases. The analyses in section 4 to 7 are based on one-port model, section 8 presents its application in square mesh/torus under the all-port model.

4 2D square mesh/torus

4.1 Algorithm description

We begin with the discussion of mesh and torus topology. We consider the two topologies together because the wrap-around links in torus networks are not used in the algorithm if the source node is located in a corner of the mesh/torus(tori are treated as mesh). In the next section, we will show that the recursion algorithm works when the source node is not located in a corner without any latency penalty. For square mesh/torus, the H-graph is the two diagonals, and G_1 to G_4 are four sub-squares equally dividing the mesh/torus.

We now study the case of 2^nx2^n mesh and the source is located in the corner. Later we will discuss the general 2^nx2^m mesh/torus.

First, as phase 1 in [3], stage 1 splits and scatters throughout one of the diagonals (Fig 3.a shows the first two steps of split and scatter). After n transfers, each node along the diagonal contains $1/2^n$ of the source message.

Each recursive round in stage 2 is conducted simply through two steps of sharing operation.

Step 1 Sharing flints vertically with a node symmetric to central horizontal line. Fig 3.b.

Step 2 Sharing flints horizontally with a node at 2^{n-1} distance away. Fig 3.c.

Note that in step 1 of the first recursive round (Fig 3.b), the sharing is not symmetric, i.e. one diagonal shares with the other empty diagonal or it gives a copy of its flints to the other diagonal. However in the step 1 of following recursive rounds, sharing is between two diagonals with flints (Fig3.d shows the step 1 of the second recursive round when flints are shared in each subgraph G_i's diagonals). Step 2's for all the recursive rounds are as in Fig 3.c, except that the sizes of the G-graphs (H-graphs) are different. In Fig 3.c, i.e. after first recursive round, the H-graphs of G_1 to G_4 contain all the message flints of the original message in their two diagonals.

The recursion ends with the G-graph (H-graph) becoming a single node. It's easy to see there's no contention for channels.

Fig 3

4.2 Cost calculation

We calculate α and β separately. Stage 1 needs n transfers. Recursive rounds is invoked n times with 2 transfers each time, thus α=3n. β is calculated according to the total message length in each parallel transfer. For stage 1, we have the cumulative value of:

$$\beta_1 = \sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n}$$

Recursive rounds will be invoked n times, and except for the first time, the two transfer lengths in one recursive round are different. So the transfer length β_{2i} for round i is given by:

$$\beta_{21} = \frac{1}{2^n} + \frac{1}{2^n} \text{ if i=1} \quad \text{and} \quad \beta_{2i} = \frac{1}{2^{n+1-i}} + \frac{1}{2^{n+2-i}} \text{ if i} \in [2,n]$$

Adding together, we have the following:

$$\beta_2 = \beta_{21} + \sum_{i=2}^{n} \beta_{2i} = \frac{1}{2^n} + \frac{1}{2^n} + \sum_{i=1}^{n-1}\left(\frac{1}{2^i} + \frac{1}{2^{i+1}}\right) = \frac{2}{2^n} + \sum_{i=1}^{n-1}\frac{1}{2^i} + \sum_{i=2}^{n}\frac{1}{2^i} = 1.5 - \frac{1}{2^n}$$

Thus the total value of β is given by: $\beta = \beta_1 + \beta_2 = 2.5 - \frac{1}{2^{n-1}}$ which is slightly less than 2.5.

4.3 Performance comparison

To provide a fair and meaningful comparison, we have chosen several other state-of-the-art algorithms. RD is optimal for short messages (which we prove in section 10), and SC is the best for long messages. From the paper of [2], we can see NP with different combination can achieve the lowest latency for a wide range.

Table 1

Since RD is the special case of NP-RD and SC is the special case of NP-SC for d=0, we just compare RB with NP-RD and NP-SC with d=0..n.

Fig 4

From the result, we can see that for almost all the message length from 1KB to 10KB, RB is the best among all algorithms. RD is most effective when message length is less than 1KB. SC is the best algorithm when message length is greater than 25KB.

5 Non-corner source in two Dimensional Mesh/Torus

This section shows that with the recursive algorithm the broadcast cost will not increase if the source node is not located in the corner. Previous algorithms have the same cost whether the source is in a corner or not. The algorithm in this section reduces to RB in the special case of corner source.

We consider a $2^n \times 2^n$ mesh. Every mesh node ν will be identified by a horizontal and a vertical coordinate expressed in binary. For example, the leftmost node at the bottom of a 4×4 mesh is (00,00), the rightmost node at the bottom is (11,00), and the rightmost node at the top by (11,11). An example is given in Table 1, where each node is represented by a tile. Two nodes are interconnected in the mesh if the corresponding tiles share an edge.

Table 2

Given a $2^n \times 2^n$ mesh (n>0), we define a *canonical submesh* as a set of nodes whose most significant bits of the two coordinates are the same. A canonical submesh can be identified by the most significant bit of its horizontal and vertical ordinates of any node in the submesh, e.g. in Table 2, we say the four lower- right nodes (10,01),(11,01), (10,00),(11,00) are in the canonical submesh (1,0), because their most significant bit in the horizontal coordinate is 1, and their most significant bit of vertical coordinate is 0. Hence, there are exactly 4 canonical submeshes in a non-trivial square mesh.

We also define three functions that map a node to another node.

1. The k^{th} *vertical symmetry mapping* V_k(ν) of ν, which is obtained by complementing the k least significant bits in ν's vertical coordinate, i.e. V_k((x,y))=(x,y⊕0…01…1)= (x,y⊕(2^k-1)), where ⊕ denotes bitwise XOR. For instance, V_2((010,110))=(010,101).
2. The k^{th} *horizontal symmetry mapping* H_k(ν) of ν, which complements the k^{th} least significant bit in ν's horizontal coordinate, i.e. H_k((x,y))=(x⊕0…010…0,y)= (x⊕2^{k-1},y). For example, the horizontal symmetry mapping H_3((010,110))=(110,110).
3. The k^{th} *central symmetry mapping C_k(ν)* of ν is obtained by changing the k^{th} least significant bit of ν both in the horizontal and in the vertical coordinate. For example, two central

symmetry mappings applied to the origin ν=(00,00) give $C_1(\nu)$=(01,01) and $C_2(\nu)$=(10,10). In general, a series of applications of central symmetry mapping maps (x,y) into ($x\oplus z, y\oplus z$) for some n-bit binary number *z*, and, conversely, there is a series of central symmetric mapping that maps (x,y) to ($x\oplus z, y\oplus z$) for any n-bit binary numbers *z*.

Suppose the source node is in (x_s ,y_s). The algorithm can be described as follows:

```
Stage 1: set  GOT_FLINT={($x_s$ ,$y_s$)}
         For  k=n downto 1 do
         begin
              For each node (x, y) in GOT_FLINT, transfer half of its flints to $C_k$((x,y))
              Add $C_k$((x,y)) to GOT_FLINT
         end
Stage 2: For k=n downto 1 do
         begin
              For each node (x, y) in GOT_FLINT, share its flints with $V_k$((x,y)) and add $V_k$((x,y)) to
              GOT_FLINT if $V_k$((x,y))$\notin$ GOT_FLINT,
              For each node (x, y) in GOT_FLINT, share its flints with $H_k$((x,y)) and add $H_k$((x,y)) to
              GOT_FLINT if $H_k$((x,y))$\notin$ GOT_FLINT,
         end
```

To show algorithm correctness, we first give some definitions and lemmas:

DEFINITION 1. $SET_k(x_s,y_s)$ $SET_k(x_s,y_s)$={ ($x_s\oplus z$, $y_s\oplus z$) | z is a k bit binary number}

The set $SET_k(x_s,y_s)$ contains 2^k nodes, since there are 2^k different z's, and it's easy to show that the 2^k nodes ($x_s\oplus z$, $y_s\oplus z$) are different.

LEMMA 1. *$SET_k(x_s,y_s)$= $SET_{k-1}(x_s,y_s)$ U $SET_{k-1}(x_s\oplus 2^{k-1}, y_s\oplus 2^{k-1})$ for k>1.*

This lemma is easy to prove, so we omit the proof here.

DEFINITION 2. $\Phi_{x_s,y_s,k}$ $\Phi_{x_s,y_s,k}$ is the mapping from {0,1,2…2^k-1} to $SET_k(x_s,y_s)$, $\Phi_{x_s,y_s,k}$: z $\mapsto$ ($x_s\oplus z$, $y_s\oplus z$)

Φ is a one-to-one mapping, which maps an integer between 0 and 2^k-1 to a node.

LEMMA 2. *$\Phi_{x_s,y_s,k}$ ({0,1,2…2^{k-1}-1}) = $SET_{k-1}(x_s,y_s)$ and $\Phi_{x_s,y_s,k}$ ({2^{k-1} … 2^k-1}) = $SET_{k-1}(x_s\oplus 2^{k-1}, y_s\oplus 2^{k-1})$.*

Proof: Let {0,1,2…2^{k-1}-1} be the set of all the k-1 bit binary numbers. For each k-1 bit binary number, $\Phi_{x_s,y_s,k} = \Phi_{x_s,y_s,k-1}$. SO, $\Phi_{x_s,y_s,k}$ ({0,1,2…2^{k-1}-1}) = $SET_{k-1}(x_s,y_s)$. Notice that set {2^{k-1} … 2^k-1}={0,1,2…2^{k-1}-1}$\oplus 2^{k-1}$ and $\Phi_{x_s,y_s,k}$ is defined as z $\mapsto$ ($x_s\oplus z$, $y_s\oplus z$). So $\Phi_{x_s,y_s,k}$($z\oplus 2^{k-1}$)=($x_s\oplus(2^{k-1}\oplus z)$, $y_s\oplus(2^{k-1}\oplus z)$)=(($x_s\oplus 2^{k-1}$)$\oplus z$, ($y_s\oplus 2^{k-1}$)$\oplus z$) for z$\in${0,1,2…2^{k-1}-1}. Considering the fact that $\Phi_{x_s,y_s,k}$= $\Phi_{x_s,y_s,k-1}$ for k-1 bit binary numbers, $\Phi_{x_s,y_s,k}$($z\oplus 2^{k-1}$)= $\Phi_{x_s\oplus 2^{k-1},y_s\oplus 2^{k-1},k-1}$(z) for z$\in${0,1,2…$2^{k-1}$-1}. Hence, $\Phi_{x_s,y_s,k}$ ({2^{k-1} … 2^k-1}) = $\Phi_{x_s,y_s,k}$ ({0,1,2…2^{k-1}-1}$\oplus 2^{k-1}$) = $\Phi_{x_s\oplus 2^{k-1},y_s\oplus 2^{k-1},k-1}$({0,1,2…$2^{k-1}$-1})= $SET_{k-1}(x_s\oplus 2^{k-1}, y_s\oplus 2^{k-1})$. □

DEFINITION 3. Collectively owned We say that a flint is *collectively owned* by two nodes means both of them have part of the flint and the two parts add up to the whole flint.

LEMMA 3. *At the end of stage 1, a message is evenly distributed into 2^n different flints.*

Proof: Stage 1 is the same as the first stage in SC. Referring to [2][9], the message is evenly distributed into 2^n different flints. □

We will mark the flints as 0,1,2…2^n-1.

LEMMA 4. *At the end of stage 1, flint number z is collectively owned by $\Phi_{x_s,y_s,n}(z)$ and $\Phi_{x_s,y_s\oplus(2^n-1),n}(z)$.*

Proof: As the definition of central symmetry mapping, each node in $SET_n(x_s,y_s)$ gets flints during stage 1. The one-to-one mapping $\Phi_{x_s,y_s,n}(z)$ maps flint number z to the corresponding node that owns it. Hence, flint number z is collectively owned by $\Phi_{x_s,y_s,n}(z)$ and $\Phi_{x_s,y_s\oplus(2^n-1),n}(z)$, although there is no flint in $\Phi_{x_s,y_s\oplus(2^n-1),n}(z)$. □

LEMMA 5. Suppose that before a recursive round k (2^k x 2^k size mesh/torus) of stage 2, a message is evenly distributed into 2^k different flints, which are marked as $0,1,2\ldots2^k-1$, and the flint number z is collectively owned by $\Phi_{x_s,y_s,k}(z)$ and $\Phi_{x_s,y_s\oplus(2^k-1),k}(z)$.

Then, after one recursive round, we have that:

1. There are four sets whose elements have flints: $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b)$ U $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))$, a=0 or 2^{k-1}; b=0 or 2^{k-1}.
2. $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b) \cap SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))=\varnothing$ for k>1, a=0 or 2^{k-1}; b=0 or 2^{k-1}.
3. Each of the four sets in 1 has 2^k different nodes.
4. The 2^k flints in each set are different from each other and located in different nodes of the set.
5. For fixed a and b, all the nodes in one set are in the same canonical submesh $((x_s\oplus a)>>(k-1)\ ,\ (y_s\oplus b)>>(k-1))$, where >> denotes *shift right*
6. If a node (x, y) is in $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b)$, then $(x,\ y\oplus(2^{k-1}-1))$ is in $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))$, a=0 or 2^{k-1}; b=0 or 2^{k-1}.
7. In the canonical submesh $((x_s\oplus a)>>(k-1)\ ,\ (y_s\oplus b)>>(k-1))$, (a=0 or 2^{k-1}; b=0 or 2^{k-1}), the whole message can be divided again into 2^{k-1} different flints marked as $0,1\ldots2^{k-1}-1$, where flint $z'=0,1\ldots2^{k-1}-1$ is collectively owned by $\Phi_{x_s\oplus a,y_s\oplus b,k-1}(z')$ and $\Phi_{x_s\oplus a,y_s\oplus b\oplus(2^{k-1}-1),k-1}(z')$ in the canonical submesh.

Proof:

1. After the vertical sharing V_k, $u_1=\Phi_{x_s,y_s,k}(z)$ has flint z and $u_2=\Phi_{x_s,y_s\oplus(2^k-1),k}(z)$ has flint z as well. Then, after horizontal sharing H_k, $u_3=\Phi_{x_s\oplus2^{k-1},y_s,k}(z)$ and $u_4=\Phi_{x_s\oplus2^{k-1},y_s\oplus(2^k-1),k}(z)$ also get a full copy of flint z, $(z=0,1\ldots2^k-1)$. So, there are 4 nodes u_1, u_2, u_3, u_4 containing flint z. It is immediate that the 4 nodes are different.
   Therefore, the nodes containing flints belong to four sets:
   $u_1\in SET_k(x_s,y_s)$, $u_2\in SET_k(x_s,y_s\oplus(2^k-1))$, $u_3\in SET_k(x_s\oplus2^{k-1},y_s)$, $u_4\in SET_k(x_s\oplus2^{k-1},y_s\oplus(2^k-1))$
   We further divide each of the four sets into two sets based on lemma 1 and lemma 2, one containing flints $0,1..\ 2^{k-1}-1$, the other containing flints from 2^{k-1} to 2^k-1(Table 2):
   Table 3
   From the table, we can identify the four sets:
   $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b)$ U $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))$, a=0 or 2^{k-1}; b=0 or 2^{k-1}, which proves the first claim.
2. If $(x_s\oplus a\oplus z\ ,\ y_s\oplus b\oplus z) = (x_s\oplus a\oplus z'\ ,\ y_s\oplus b\oplus z'\oplus(2^{k-1}-1))$,i.e. they represent the same node, $x_s\oplus a\oplus z=x_s\oplus a\oplus z'$ and $y_s\oplus b\oplus z= y_s\oplus b\oplus z'\oplus(2^{k-1}-1)$. From the first equation, we get z=z', from the second equation, we get $z=z'\oplus(2^{k-1}-1)$. These equations can be satisfied only in the special case, k=1. So, $SET_{k-1}(x_s\oplus a,\ y_s\oplus b) \cap SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))=\varnothing$ when k>1.
3. $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b)$ and $SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))$ both contain 2^{k-1} different nodes (by Definition 1) and $SET_{k-1}(x_s\oplus a,\ y_s\oplus b) \cap SET_{k-1}(x_s\oplus a\ ,\ y_s\oplus b\oplus(2^{k-1}-1))=\varnothing$

4. From the table, we find that of $SET_{k-1}(x_s \oplus a, y_s \oplus b)$ and $SET_{k-1}(x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1))$, one contains flints 0 to $2^{k-1}-1$, the other contains flints 2^{k-1} to 2^k-1. There are totally 2^k different nodes in $SET_{k-1}(x_s \oplus a, y_s \oplus b)$ U $SET_{k-1}(x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1))$, so the flints in each set are different from each other and in different nodes.
5. The nodes of $SET_{k-1}(x_s \oplus a, y_s \oplus b)$ and $SET_{k-1}(x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1))$ can be written as:
   $(x_s \oplus a \oplus z', y_s \oplus b \oplus z')$ and $(x_s \oplus a \oplus z', y_s \oplus b \oplus (2^{k-1}-1) \oplus z')$, $(z'=0,1.. 2^{k-1}-1)$
   Since z' and $2^{k-1}-1$ are both no more than k-1 bit long, the most significant bit of the coordinates of the nodes are given by $x_s \oplus a$ and $y_s \oplus b$. Therefore, they are all in the canonical submesh $((x_s \oplus a)>>(k-1), (y_s \oplus b)>>(k-1))$.
6. A node in $SET_{k-1}(x_s \oplus a, y_s \oplus b)$ can be written as $(x_s \oplus a \oplus z', y_s \oplus b \oplus z')$, $z'=0,1.. 2^{k-1}-1$. So, $(x_s \oplus a \oplus z', y_s \oplus b \oplus (2^{k-1}-1) \oplus z')$ is in $SET_{k-1}(x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1))$.
7. Because $\Phi_{x_s \oplus a, y_s \oplus b, k-1}$ is a one-to-one mapping from {0,1.. $2^{k-1}-1$} to $SET_{k-1}(x_s \oplus a, y_s \oplus b)$ and $\Phi_{x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1), k-1}$ is a one-to-one mapping from {0,1.. $2^{k-1}-1$} to $SET_{k-1}(x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1))$, we define flint $z'(0,1…2^{k-1}-1)$ as the combination of the flint in node $\Phi_{x_s \oplus a, y_s \oplus b, k-1}(z')$ and the flint in node $\Phi_{x_s \oplus a, y_s \oplus b \oplus (2^{k-1}-1), k-1}(z')$. The 2^{k-1} double-sized flints constitute the whole message and they are in the same canonical submesh $((x_s \oplus a)>>(k-1), (y_s \oplus b)>>(k-1))$. □

LEMMA 6. *No contention occurs during the algorithm.*
Proof: Notice that at any time in stage 1, only one node is on each horizontal or vertical line. Assuming that we first transfer horizontally then vertically, no collision will happen. Similarly, in each recursive round of stage 2, both horizontal and vertical sharings are on different parallel lines. So, the algorithm is contention free. □

THEOREM 1: *The algorithm broadcasts the source message to every node in the mesh.*
Proof: We prove that for $0 \le k \le n$, each of the $2^k x 2^k$ size canonical submesh contains all the message flints of the original message. We prove this by induction on k. Lemma 3 and Lemma 4 are the base case for k=n. Lemma 5 claims that when it holds for t=k, it's also correct for t=k-1.

When k reaches 0, the canonical submesh becomes single node, i.e each node has got a copy of the original message, and so the message has been broadcast to every node, which concludes the proof.□

THEOREM 2: *The total latency for the non-corner situation is the same as that of the corner situation.*
Proof: Firstly, from the algorithm, the number of transmission (α) does not depend on the source (x_s, y_s). The flint size of each concurrent transmission, which builds up to β, is also independent of the source node. So the total latencies are the same for both corner and non-corner situations.□

6 Non-square meshes/tori

For generalization purpose, we now consider the $2^n x 2^m$ (n>m) non-square meshes/tori and prove that it needs only little additional cost to complete the broadcast.

First, we begin with n=m+1. The difference is that we must spread the message to one diagonal in each of the two meshes, then, shift horizontally. After that, the recursion steps are the same in each square submeshes.

Fig 5

If n=m+2, we illustrate it as the following:

Fig 6

Two sharing operations are used to reduce the graph into two subgraphs of the n=m+1 situation. In general, let n=m+k, where k≥1. We need 2^{k-1} sharings (each with a diagonal 2^{k-1} blocks away) to reduce from a 2^{m+k}x2^m mesh to two 2^{m+k-1}x2^m submeshes because of the channel contentions.

The total latency is rewritten as: $T(k) = T_{scatter}(k) + T_{reduce}(k) + T_0$, where $T_{scatter}(k)$ is the time to spread the message flints along the diagonals as in Fig 5, $T_{reduce}(k)$ is the time to share and transfer the whole message flints to all the 2^k 2^mx2^m square meshes. T_0 is the time to complete the broadcast from the 2^mx2^m square mesh containing all the flints like the two ones in Fig 5. By a simple calculation, we get:

$$T(k) = T_{scatter}(k) + T_{reduce}(k) + T_0 = (3m + k + 2^k - 1)T_s + (2.5 + \frac{k-2}{2^{m+1}} - \frac{1}{2^n})LT_c$$

We can observe that both α and β are kept small for a typically small k.

7 High-dimensional meshes/tori

The broadcast problem in three dimension or higher meshes/tori has a fast solution using the RB algorithm. In this section, we only consider the 3D mesh, and applications to higher dimensional meshes can be handled in a similar manner.

7.1 Algorithm description

For a 2^nx2^nx2^n mesh, we denote the nodes as 3 bit binary numbers illustrated in Fig7.a. In a 3D mesh/torus, the H-graph is the four main diagonals in the cube. Graph G is divided into 8 small cubes--G_1 to G_8, and we will transfer and share the whole message flints into 8 subgraphs in 3 steps.

Stage 1: create one main diagonal (000, 111) by split and scatter. Then split again along the Y direction to another main diagonal (010, 101), i.e. transfer half flints to it. Now, the 2^{n+1} nodes in the two diagonals have 1/ 2^{n+1} of the source message each.

Fig 7

After the initial stage, each recursive round is done in 3 steps.

Step 1: As shown in Fig7.a the diagonal (000,111) to (010,101) share flints with the main diagonal from (011,100) to (001,110) along the X-direction respectively. Note that the sharing is also asymmetric, as in section 4, when we invoke step 1 for the first time.

Step 2: As shown in Fig7.b, let Z_{00} , Z_{01} , Z_{10} , Z_{11} be the middle nodes of four vertical (Z direction) edges. Those nodes specify a horizontal plane. Now every node containing message flints shares them with a node at a distance of $2^n/2$ and located vertically down (or up) from it, so that all the nodes containing message flints below (above) this plane share flints with a node above (below) the plane.

Step 3: As shown in Fig7.b, let Y_{00} , Y_{01} , Y_{10} , Y_{11} be the middle nodes of four Y direction edges. Those nodes specify a vertical plane. Now every node containing a message flints share them with a node at $2^n/2$ distance along the Y direction. so that all the nodes contain message flints before (behind) this plane share flints with a node behind (before) the plane.

Using the calculation similar to section 4, we can get the value α and β in this algorithm: α =4n+1 and β=2.5-1/2^n-1/$2^{(n+1)}$.

7.2 Performance Comparison

For comparison, we only present the calculation of the SC, RD, and NP combined algorithms in 3-dimensional mesh/torus.

For RD_{3d}, we have α=3n, β=3n. For SC_{3d}, we have α=3n+3*2^n-3 , β=2-2/2^(3n).

For the NP algorithm, we define the DCN as 2^d x 2^d x 2^d (h=2^d) sub-3-dimensional mesh. The nodes along one main diagonal of each DCN are in the DDN. The DDNs are diluted 2^{n-d} x 2^{n-d} x 2^{n-d} 3D meshes. We can apply RD or SC to broadcast in the diluted 3D mesh. In summary, for NP-RD_{3d} α=3n+2^d-1, β=2+(3n-d-2)/2^d and for NP-SC_{3d} α=3n+3*2^(n-d)+2^d-4, β=2+2d/2^d-2/2^(3n-2d)

Latency comparisons (Fig 8) show there is a little bit more advantage in a 3D mesh than in a 2D mesh for RB over the other 3 algorithms.

Fig 8

8 Application to the square mesh/torus under all-port model

The same paradigm can be applied to the all-port model. For simplicity, we give out our discussion for the case when the source is in a corner node. Similar to section 5, the algorithm also works for a non-corner source without overhead.

8.1 Algorithm description

In the all-port model, a node can simultaneously receive and send as many messages in its input/output channels as in its external channels.

The H-graph is still the two diagonals of the mesh. However G is divided into 16 smaller square submeshes, which are denoted as G_1, G_2,…, G_{16} . We need 3 steps to share all the message flints to each G_i. The algorithm is designed carefully to prevent any channel contention.

Fig 9

Fig 9.a illustrates the first step, with the message flints on the two diagonals marked as eight segments, from 1 to 8. After step 1, the (segments) flints distribution is as in Fig 9.b. Fig 9.c illustrates the second step, after which, the (segments) flints distribution is as in Fig 9.d. The last step is in Fig 9.e, which ends up with each of the 16 submeshes G_1 –G_{16} getting all the 8 segments.

8.2 Cost calculation

The stage 1 (split scatter) for all-port model is the same as what is introduced in [2], which needs: $\alpha = \lceil (n+1)*\log_5 2 \rceil +1$, and β = 1/4+1/2^n,where $\lceil\ \rceil$ is the ceiling function.

On the whole, for this algorithm, $\alpha = \lceil (n+1)*\log_5 2 \rceil$+1.5*(n-1)+2, β = 17/12+37/3/2^(n+1).

8.3 Performance Comparison

Comparison is with the D-node[11] algorithm, and the improved NP-D[2].

For D-node, α = n+2 β = n+2.

For NP-D, α =n-d+2^d+3+$\lceil (d+1) \log_3 2 \rceil + \lceil d \log_5 2 \rceil$ β =1.25+(n-d+2+$\lceil (d+1)\log_3 2 \rceil$)/2^d

Fig 10

Obviously, the RB algorithm provides greater advantages in the all-port model than in the one-port model, largely due to the fact that it can share flints into 16 subgraphs with only 3 steps.

9 Performance analysis

In this section, we estimate the distance of the current algorithms from the optimal one. The following arguments are all based on the one-port model.

PROPOSITION 1: *For any algorithm, $\alpha \geq 2n$.*

REMARK: While RD's α(=2n) is optimal, its β(=2n) is large.

Proof: In each step, the number of nodes containing flints will be at most doubled. In the best case, the number of nodes containing flints increases exponentially. So, $\alpha \geq \log_2(2^n x 2^n)=2n$. □

PROPOSITION 2: *In the one-port model, if the source node sends out the whole message in k blocks then $\alpha \geq \log_2 N+k-1$, where N is the number of nodes.*

Proof: After k-1 transfers, at least one block will still remain in the source node. To broadcast this block of flints will need at least $\log_2 N$ steps as what is stated in proposition 1. Therefore, $\alpha \geq \log_2 N+k-1$. □

PROPOSITION 3: *In the one-port model, if the source node sends out the whole message in less than $\log_2 N$ blocks then $\beta \geq 2$, where N is the number of nodes.*

Proof: After the source node sends out all its blocks, there will be one node that has not received any block from the source node (the number of nodes with message blocks at most doubled in each step). We separate the whole broadcast into two stages. In the first stage, the source sends out all its blocks, so $\beta \geq 1$ in this stage. In stage two, the un-transferred node receives all the blocks, so $\beta \geq 1$. On the whole, $\beta \geq 2$. □

THEOREM 3: *For any $\varepsilon>0$, there is a broadcast algorithm with $\beta \leq 1+\varepsilon$.*

Proof: We first line up all the nodes in the mesh in such a way that all the nodes appear in this line once, which is easily done by concatenating rows one by one. Note that there are $N=2^{2n}$ nodes in this line. The source message is equally divided into $M>>2^{2n}$ flints. The value of M can be very large. Then the source sends out the flints one after another. Intermediate nodes receive one flint and send again to the following node. When the first flint reaches the last node in the line, most flints are still in the source. At this time, the cumulated value of β is written as $\beta_1=N/M$. In the next stage, the last node will receive all the flints in transmission, i.e. $\beta_2=1$. Totally, $\beta=\beta_1+\beta_2=1+N/M=1+\varepsilon$. It follows that this scheme produces a small β approaching 1. □

REMARK: The small value of β is achieved by dividing the message in an exponential number of flints. Dividing a message into a large number of flints results in a large value of α.

Based on these theorems, we have the good reason to believe, although still can not prove it, that α and β should be well balanced to achieve a low latency for middle-sized message, which is the common case in real systems.

10 Conclusion

This paper presents a widely applicable solution for the broadcast problem in a wormhole-routed mesh/torus. The recursive sharing-into-subgraph process produces good results with both small α and β. Unlike other studies in this type of problem, recursion based algorithms have the same type of results in different cases for different underlying networks, and different length of messages. Moreover, the framework can be applied to a wide range of topologies. Topologies studied in this paper include 2 and 3 dimensional mesh/torus and both one-port and all-port model. We have provided theorems to show upper and lower bounds on its performance.

We believe that the paradigm will be easy to use it in high-dimensional mesh/torus with reduced delays. However the usefulness of the recursion-based algorithm on more complicated topologies are still unclear, esp. as related to the problem of identifying the characteristic low-dimensional subgraph, so that the whole message flints can be sent to all of the characteristic low-dimensional subgraph in a O(1) time. However, as long as it can be found, the algorithm will give rise to the same optimal asymptotic complexity.

The deficiency in this algorithm is its assumption of contention free and synchronized transfer, which make the problem tractable. Further study can combine the algorithm with other deadlock-free algorithms [12][13] and consider the resolution of *depth-contention problem[6]*.

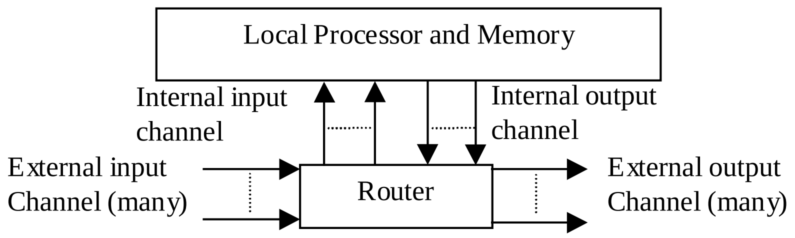
Local Processor and Memory
Internal input channel
Internal output channel
External input Channel (many)
Router
External output Channel (many)

Fig 1. The model for a single network node

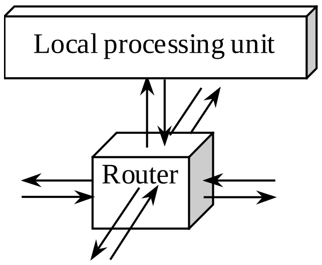

Fig 2. Example of a node in a 2D torus, one-port model

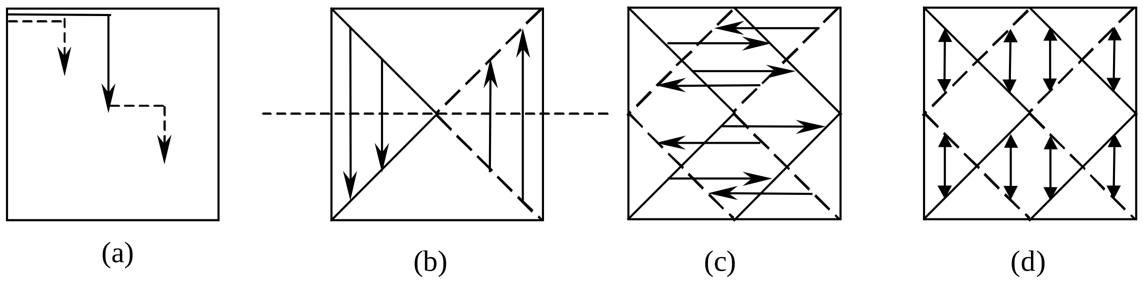

Fig 3. Broadcast steps in 2D square mesh/torus. (a) Stage 1. (b) step 1 for the first recursive round. (c) step 2 for the first recursive round. (d) step 1 for the second recursive round

Algorithms	α	β
RD	2n	2n
SC	2n+2^(n+1)-2	2-1/2^2n
NP-RD	2n+2^d-1	2+(2n-d-2)/2^d
NP-SC	2n+2^(n-d+1)+2^d-3	2+d/2^d-2/2^(2n-d)
RB (ours)	3n	2.5-1/2^(n-1)

Table 2: Comparisons between RB and several well-known algorithms.

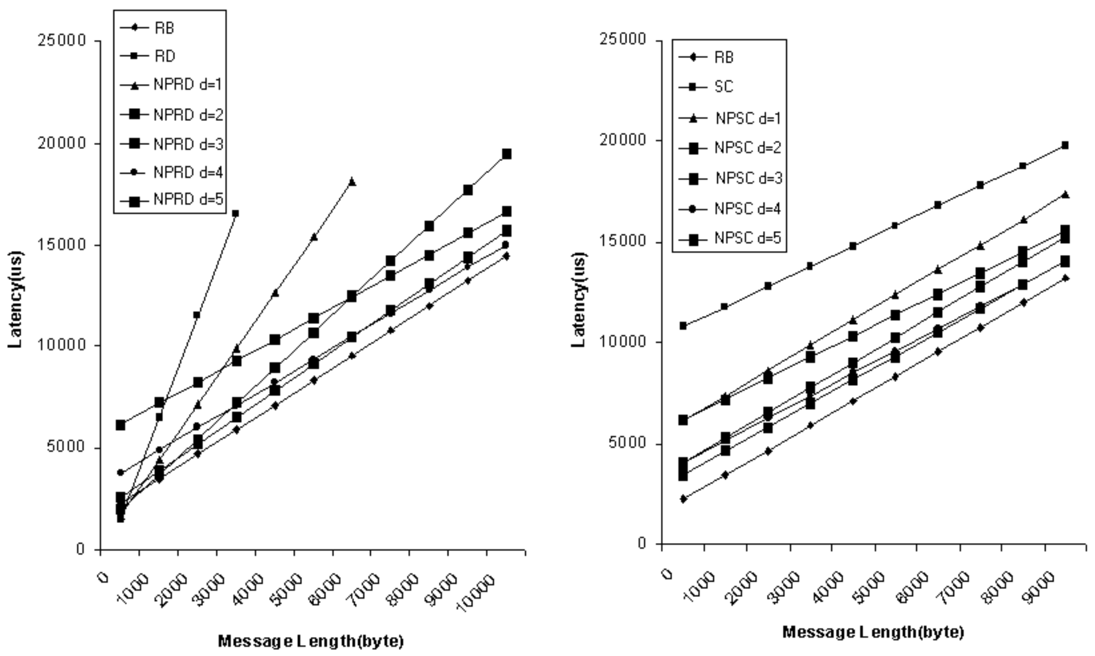

Fig 4 Performance comparison (left, with RD and NPRD. right, with SC and NPSC), parameters: n=5;Ts=150;Tc=0.5;one-port model.

(00,11)	(01,11)	(10,11)	(11,11)
(00,10)	(01,10)	(10,10)	(11,10)
(00,01)	(01,01)	(10,01)	(11,01)
(00,00)	(01,00)	(10,00)	(11,00)

Table 2: A 4×4 mesh and the corresponding node coordinates.

	Flints 0 to 2^{k-1}-1	a , b	Flints 2^{k-1} to 2^k-1	a, b
$SET_k(x_s,y_s)$	$SET_{k-1}(x_s,y_s)$	0,0	$SET_{k-1}(x_s \oplus 2^{k-1}, y_s \oplus 2^{k-1})$	$2^{k-1},2^{k-1}$
$SET_k(x_s,y_s \oplus (2^k-1))$	$SET_{k-1}(x_s,y_s \oplus 2^{k-1} \oplus (2^{k-1}-1))$	$0,2^{k-1}$	$SET_{k-1}(x_s \oplus 2^{k-1},y_s \oplus (2^{k-1}-1))$	$2^{k-1},0$
$SET_k(x_s \oplus 2^{k-1},y_s)$	$SET_{k-1}(x_s \oplus 2^{k-1},y_s)$	$2^{k-1},0$	$SET_{k-1}(x_s,y_s \oplus 2^{k-1})$	$0,2^{k-1}$
$SET_k(x_s \oplus 2^{k-1},y_s \oplus (2^k-1))$	$SET_{k-1}(x_s \oplus 2^{k-1}, y_s \oplus 2^{k-1} \oplus (2^{k-1}-1))$	$2^{k-1},2^{k-1}$	$SET_{k-1}(x_s,y_s \oplus (2^{k-1}-1))$	0,0

Table 3: Divide each of the four sets into 2 sets.

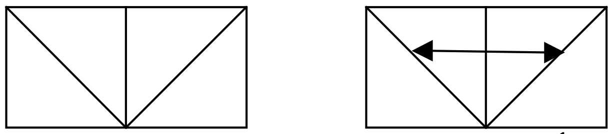

Fig 5: Sharing in a non-square mesh of 2^{m+1}x2^m

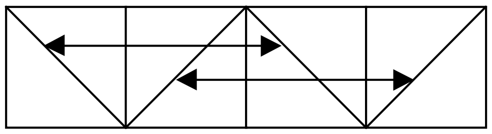

Fig 6: Sharing in a non-square mesh of 2^{m+2}x2^m

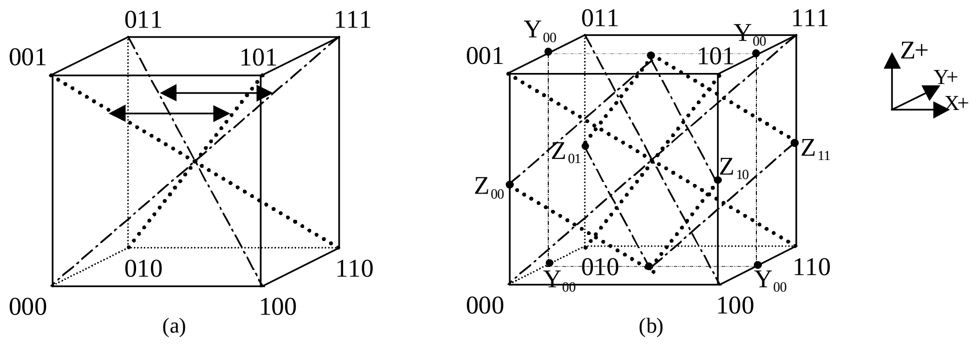

Fig 7: RB steps in 3 dimensional mesh/torus

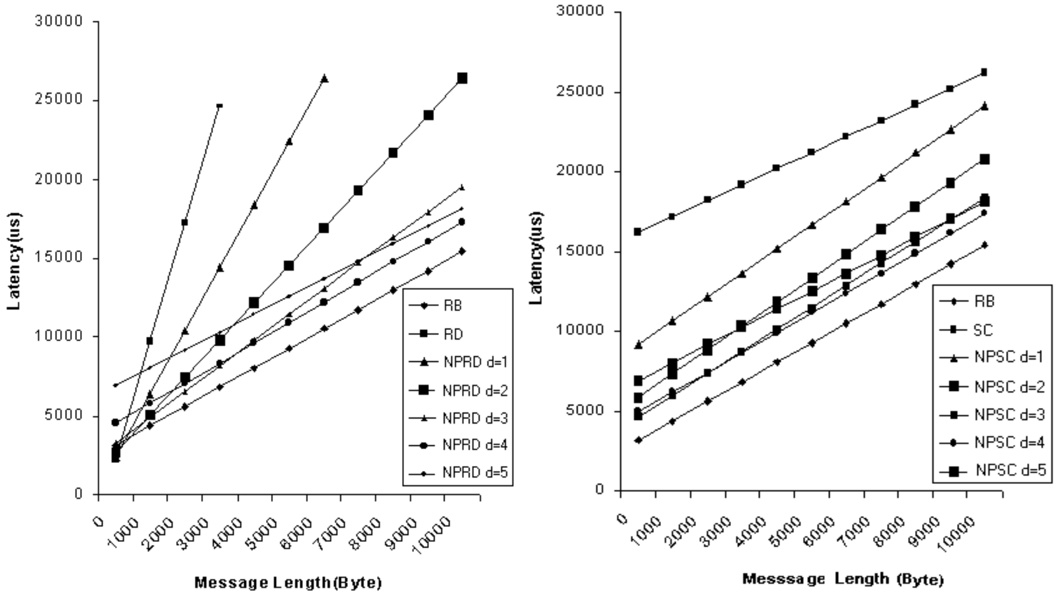

Fig 8: Performance comparison, (left, with RD and NPRD. right, with SC and NPSC),parameters: n=5;Ts=150;Tc=0.5;one-port model 3D mesh.

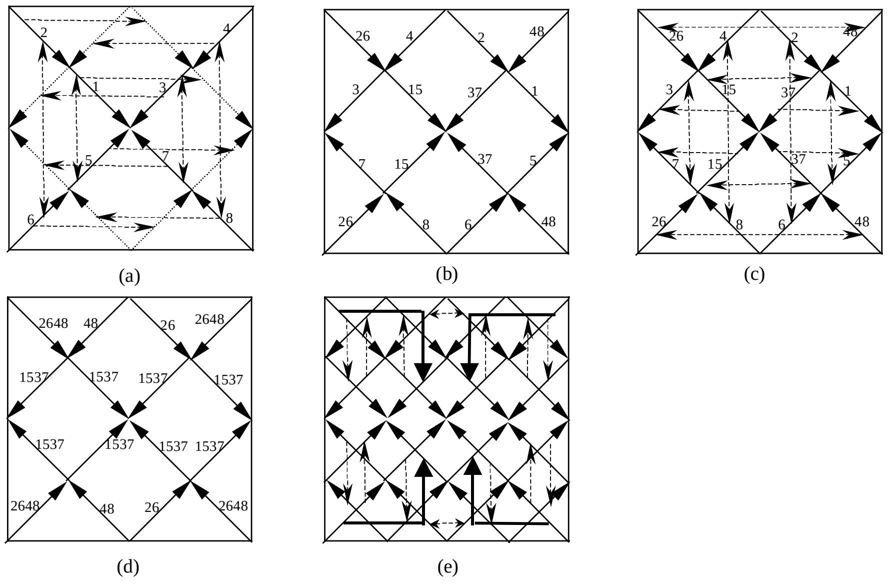
Fig 9: RB steps in all-port model for mesh/tours

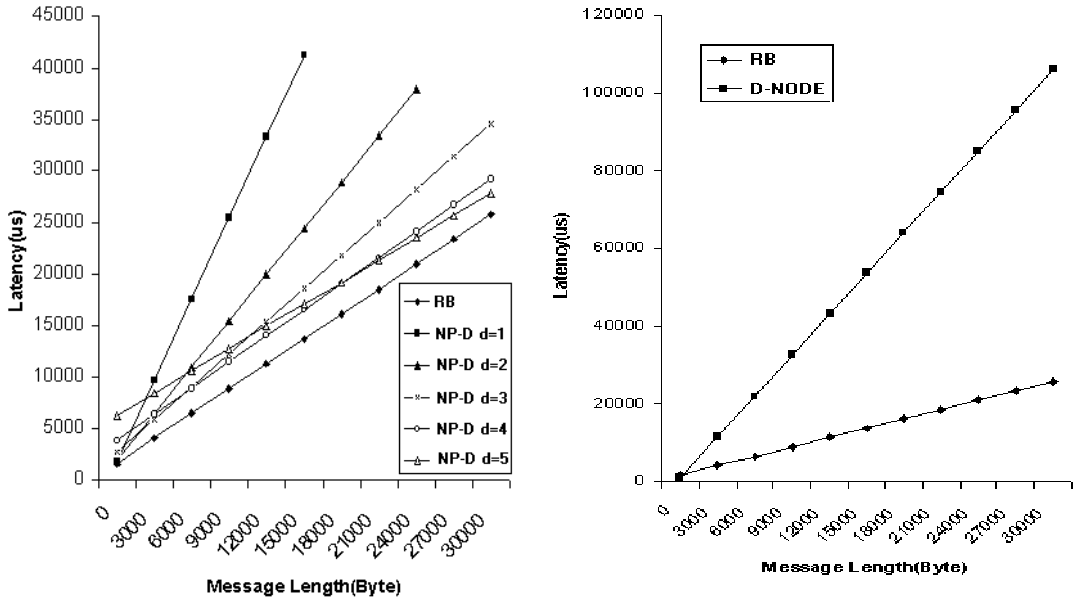

Fig 10: Performance comparison, (left, with NP-D. right, with D-NODE)parameters: n=5;Ts=150;Tc=0.5;all-port model mesh.