Multi-Node Broadcasting in a Wormhole-Routed 2-D Torus Using an Aggregation-then-Distribution Strategy

Yuh-Shyan Chen and Che-Yi Chen
Department of Computer Science and Information Engineering
Chung-Hua University
Hsin-Chu, 30067, Taiwan
Email: chenys@csie.chu.edu.tw

Yu-Chee Tseng
Depart. of Computer Science and Information Engineering
National Central University
Chung-Li 32054, Taiwan
Email: yctseng@csie.ncu.edu.tw

Abstract

This paper presents an efficient multi-node broadcasting algorithm in a wormhole-routed 2-D torus, where there are an unknown number of s source nodes located on unknown positions each intending to broadcast a message of size m bytes to the rest of network. The torus is assumed to use the all-port model and the popular dimension-ordered routing. Most existing results are derived based on finding multiple edge-disjoint spanning trees in the network. The main technique used in this paper is an aggregation-then-distribution strategy. First, the broadcast messages are aggregated into some positions of the torus. Then, a number of independent subnetworks are constructed from the torus. These subnetworks, which are responsible for distributing the messages, can well exploit the communication parallelism and the characteristic of wormhole routing. It is shown that such an approach is more appropriate than those using edge-disjoint trees for fixed-connection network such as tori. This is justified by our performance analysis.

1. Introduction

A massively parallel computer (MPC) consists of a large number of identical processing elements interconnected by a network. One basic communication operation in such a machine is broadcasting. Two commonly discussed instances are: *one-to-all* broadcast and *all-to-all* broadcast, where one or all nodes need to broadcast messages to the rest of the nodes. A more complicated instance is the *many-to-all* (or *multi-node*) broadcast, where an unknown number of nodes located in unknown positions each intending to perform a broadcast operation. The focus of this paper is the multi-node broadcast problem, whose applications can be found in parallel graph algorithms, parallel matrix algorithms, fast Fourier transformation, parallel compilation, and cache coherence. In addition to multi-node broadcasting, many collective communication patterns, such as one-to-all broacasting, multicasting, all-to-all broadcasting, complete exchange, scatter, gather, and reduction, have all intensive attention recently, [5] [3].

The multi-node broadcast problem has been studied on a variety of interconnection networks [6] [7] [8][10][12][2]. Saad and Schultz [6] [7] initially defined this problem and proposed a simple routing algorithm for hypercubes. Stamoulis and Tsitsiklis [8] proposed to use *n edge-disjoint spanning trees* in an *n*-cube to solve this problem. A distributed approach to improve the load imbalance problem in [8] was presented by Tseng [10] for hypercubes and star graphs. Efforts were made by Varvarigos [12] to solve the more complicated problem where each source node may have several messages (of the same length) to broadcast. Recently, Susanne *et al.* [2] propose a scheme called *s-to-p broadcasting*, where the authors tried to align the broadcast messages into a regular pattern before they are distributed.

The aforementioned results are all based on finding edge-disjoint spanning trees in a network and are appropriate for *non-fixed connection networks* [1]. One problem with this is that the number of edge-disjoint trees that could be offered by a network is fixed [1]. The other problem is that the characteristic of wormhole routing, which is assumed in this paper, is not well exploited [11]. In this paper, we consider 2-D tori, which have been adopted by Cray T3D and T3E and are *fixed-connection* networks. The recently popular wormhole routing technology is assumed. In the literature, sending a packet involves two costs: *start-up time* and *transmission time*. Attempts to minimize both these costs are made.

Our approach is based on an *aggregation-then-distribution* strategy. First, the *network-partitioning* techniques proposed in [11] is used to get multiple independent subnetworks (which are different from edge-disjoint spanning trees) in a torus. The number of independent subnetworks are actually an adjustable parameter. Given a multi-node broadcast problem with an unknown number of *s* source nodes located on unknown positions in an $n \times n$ torus each intending to broadcast an m-byte message, our approach can solve it efficiently in time $O(\lceil \log_5 n \rceil T_s + \max\{\lceil \log_5 \lceil \frac{n}{h} \rceil, h\} \rceil \frac{s}{h} m T_c)$, where h is the number of independent subnetworks. It is shown that this number has outperformed the aforementioned schemes using edge-disjoint-spanning-trees.

The rest of this paper is organized as follows. The basic ideas are in Section 2. Section 3 presents our scheme. Timing analyses and comparisons are in Section 4. Conclusions are drawn in Section 5.

2. Basic Idea

2.1. System Model

A *massively parallel computer* (MPC) is formally represented as $G = (V, C)$, where V denotes the node set and C specifies the channel connectivity. Each node contains a separate router to handle its communication tasks. In this paper, we consider G as a 2-dimensional torus $T_{n_1 \times n_2}$ with $n_1 \times n_2$ nodes. Each node is denoted as $P_{i,j}$, $1 \leq i \leq n_1$, $1 \leq j \leq n_2$ and P_{i_1,i_2} has an edge connected $P_{(i_1 \pm 1) \bmod n_1, i_2}$ along dimension one and an edge to $P_{i_1,(i_2 \pm 1) \bmod n_2}$ along dimension two. Each edge is considered consisting of two directed communication links pointing in opposite directions.

The *wormhole routing* model is assumed [4]. Under such model, each packet is partitioned into smaller units called *flits*, which are sent in a pipelined manner. In the absence of congestion, the communication latency in the networks is proportional to factor of sum of message length and routing distance. Specifically, the time required to deliver a packet of L bytes from a source node to a destination node can be formulated as $T_s + LT_c$, where T_s is the start-up time containing the channel setup and software overhead, and T_c represents the transmission time per data byte. In this paper, attempts will be made to counter the trade-off between the start-up and the transmission costs.

In addition, we adopt the *all-port* model, in that a node can simultaneously send and receive messages along all outgoing and incoming links, and the *dimension-ordered routing* [10], in that every message must travel links in a strictly increasing order in terms of link dimensions.

2.2. Network Partitioning

Our work is based on partitioning the torus into some subnetworks. In the following, we review some definitions, which are based on the work in [11].

Consider a torus $T_{n_1 \times n_2}$. Suppose h is an integer which divides both n_1 and n_2. We define k *data-distribution network* $DDN_k = (V_k, C_k)$, $k = 0..h-1$ as follows:

$$\begin{aligned} V_k &= \{p_{i,j} | i = ah + k, j = bh + k, \text{ for all } a = 0..(\tfrac{n_1}{h}) \\ &\quad \text{-1 and } b = 0..(\tfrac{n_2}{h}) - 1\} \\ C_k &= \{\text{all channels at rows } ah + k \text{ and column } bh + k\} \end{aligned}$$

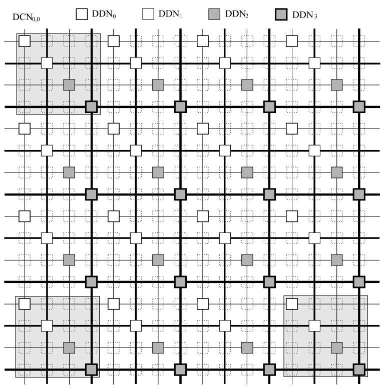

Figure 1: Network partitioning scheme.

Intuitively, each DDN is a *dilation-h* torus of size $(\frac{n_1}{h}) \times (\frac{n_2}{h})$, in the sense that each edge is dilated by a path of h edges. An example is shown in Fig. 1 with four *dilated*-4, 4×4 tori embedded in a 16×16 torus.

Also, we partition the $T_{n_1 \times n_2}$ into $\frac{n_1 \times n_2}{h^2}$ *data collecting network* $DCN_{a,b} = (V_{a,b}, C_{a,b})$, $a = 0..n_1 - 1$, $b = 0..n_2 - 1$, as follows:

$$\begin{aligned} V_{a,b} &= \{p_{i,j} | i = a \times h + x, j = b \times h + y \text{ for all } x, y = 0 \\ &\quad ..h - 1\} \\ C_{a,b} &= \{\text{the set of edges induced by } V_{a,b} \text{ in } T_{n_1 \times n_2}\} \end{aligned}$$

Intuitively, these DDNs are obtained by evenly slicing the torus into $\frac{n_1 \times n_2}{h^2}$ blocks, each being a square $h \times h$ submesh. Fig. 1 illustrates this definition when $h = 4$.

3. The Multi-Node Broadcasting Scheme

This section presents a multi-node broadcasting scheme. The work is constructed by a proposed *aggregation-then-distribution* strategy. Two phase, aggregation and distribution phases, are separately discussed.

3.1. The Aggregation Phase

3.1.1. Step 1: Diagonal-Based Data-Aggregation Operation

This subsection describes the diagonal-based data-aggregation, or namely data-aggregation, operation. The size of DDN_0, DDN_1,..., DDN_{h-1} and DCN_0, DCN_1,..., DCN_{k-1} are initially determined. It is a trade-off problem to determine the value of h and k. Basically, the higher value of h is, the lower latency will be.

Given a node $P_{i,j}$ and an integer k, define $D(P_{i,j}, k)$ to be a sequence of k nodes as follows. For instance, the main diagonal in a square $T_{n \times n}$ torus passing node $P_{0,0}$ is the sequence $D(P_{0,0}, n)$. If node $P_{i,j} = P_{0,0}$ then the sequence of the diagonal nodes is $P_{1,1}, P_{2,2}, ...$, and $P_{n-1,n-1}$. A torus can be

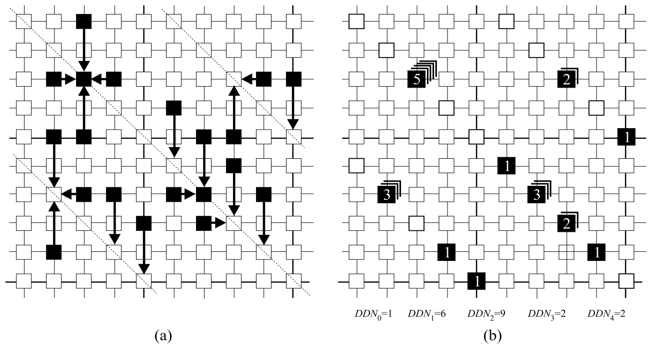

Figure 2: Example of data aggregation operation.

viewed as n diagonals $D(P_{i,0},n)$ or L_i, $i = 0..n-1$. The purpose of data-aggregation operation is to aggregate n diagonals into $\widetilde{L}_j$ diagonals, where $j = 0..\left\lceil \frac{n}{h} \right\rceil - 1$ and $\widetilde{L}_j = L_{j'}$ for $j' = j * \left\lceil \frac{n}{h} \right\rceil$. In other words, data are aggregated into main diagonal in every DCNs.

The time complexity of data-aggregation operation is totally depended on value of h. During each data-aggregation operation, every node $P_{i,j}$ in main diagonal of each DCN are aggregated messages from nodes $P_{i-1,j}, P_{i+1,j}, P_{i,j-2}$, and $P_{i,j+2}$. This also implies that $\widetilde{L}_j$ or $L_{j'}$ diagonals aggregates messages from $L_{-2+j'}, L_{-1+j'}, L_{1+j'}$ and $L_{2+j'}$. Clearly, all of the communications are congestion-free. The communication latency is determined by the size of h, not value of n. If $h > 4$, data-aggregation operation can be recursively executed within $\lceil \log_5 h \rceil$ time units. Generally, we have following result if $h > 4$.

Lemma 1 *Diagonal-based data-aggregation operation can be recursively performed on a $T_{n\times n}$ within*

$$\lceil \log_5 h \rceil T_s + \sum_{i=1}^{\lceil \log_5 h \rceil - 1} 5^{i-1} m T_c = \lceil \log_5 h \rceil T_s + \frac{5^{\lceil \log_5 h \rceil} - 1}{4} m T_c$$

, where $h > 4$.

Further, each data-aggregation operation may aggregate messages from partial nodes by four neighboring nodes if it is one of source nodes. For instance, consider 20 source nodes intending to send message to rest of network as shown in Fig. 2(a). After data-aggregation operation, the result is shown in Fig. 2(b). Assume that $h = 5$, the number of copies of distinct messages in DDN_0, DDN_1, DDN_2, DDN_3, and DDN_4 are 1, 6, 9, 2, and 2. Obviousely, it is not load balance.

3.1.1. Step 2: Balancing-Load Operation

Prefix-Sum Procedure: The main function of prefix-sum procedure is to exchange the information of amount of

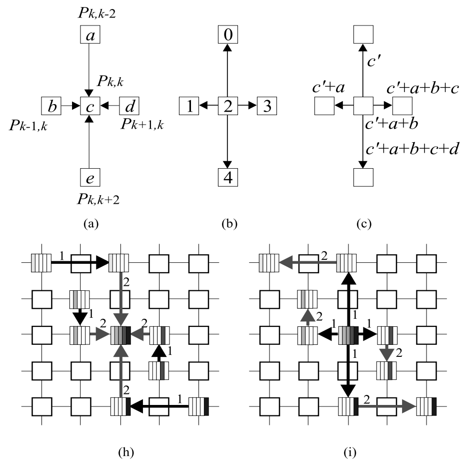

Figure 3: Prefix-sum procedure.

collected messages to calculate the prefix-sum value. Using prefix-sum value allows us to do the balancing-load operation well. This procedure only propagates control message across the 2-D tori network. The prefix-sum procedure needs (1) forward stage, and (2) backward stage. For ease of presentation, a simple prefix-sum procedure on five nodes is initially explained.

1. Basic forward stage: Node $P_{k,k}$ containing message c receives messages a, b, d, and e from nodes $P_{k,k-2}$, $P_{k-1,k}$, $P_{k+1,k}$, and $P_{k,k+2}$, as illustrated in Fig. 3(a). Note that node $P_{k,k}$ must keeps values of a, b, c, d, and e on each forward stage to calculate partial prefix-sum in future backward stage.

2. Basic backward stage: Assume that node $P_{k,k}$ gets a local partial prefix-sum value c' (from previous backward stage), then node $P_{k,k}$ must sends backward value c' plus partial prefix-sum to nodes $P_{k,k-2}$, $P_{k-1,k}$, $P_{k+1,k}$, and $P_{k,k+2}$, according to the order as shown in Fig. 3(b). That is, node $P_{k,k}$ sends value of $c', c'+a, c'+a+b, c'+a+b+c$, and $c'+a+b+c+d$ to nodes $P_{k,k-2}$, $P_{k-1,k}$, $P_{k,k}$, $P_{k+1,k}$, and $P_{k,k+2}$, respectively, as illustrated in Fig. 3(c).

Owing to the fact that all nodes with messages are located in the diagonal of tori (due to applying data aggregation operation), so a diagonal-based recursive prefix-sum procedure is needed. Nodes $P_{k-2,k-2}, P_{k-1,k-1}, P_{k+1,k+1,}$ and $P_{k+2,k+2}$ are located in a diagonal, two communication steps are needed as illustrated in Fig. 3(h) and 4(i). First communication step is let diagonal nodes $P_{k-2,k-2}, P_{k-1,k-1}, P_{k+1,k+1,}$ and $P_{k+2,k+2}$ send corresponding messages to $P_{k,k-2}$, $P_{k-1,k}$, $P_{k+1,k}$, and $P_{k,k+2}$ as shown in Fig. 3(a). This communication step is congestion-free and takes one time step. Second communication step is to perform the basic forward stage. Similarly, the backward stage on diagonal do a reverse work. Collectively, each of the diagonal-based forward and backward stages takes 2 time

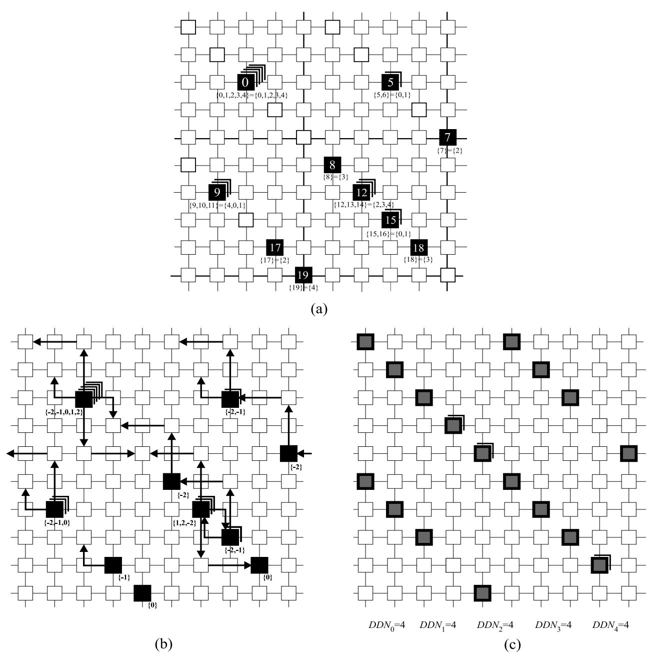

Figure 4: Example of data tuning procedure.

steps. Recursively perform the basic forward and backward stages for $\lceil \log_5 n \rceil$ times. This indicated that the recursive forward and backward stages on diagonal need time cost of $2\lceil \log_5 n \rceil (T_s + T_c)$. Each incoming data must be kept for the backward stage, so there needs $5\lceil \log_5 n \rceil$ extra memory. Consequently, the total time cost of prefix-sum procedure is $4\lceil \log_5 n \rceil (T_s + T_c)$. For example, a prefix-sum procedure is operated in Fig. 4(a).

Data Tuning Procedure: The purpose of data tuning procedure is to balance the work load among all *DDN*s. We divide the task into two parts; (1) finding a destination list, (2) performming data-movement operation.

1. Finding a destination list: Given that prefix-sum value and number of messages are α and β. The original destination list is $\{\alpha, \alpha+1, \ldots, \alpha+\beta\}$. If number of *DDNs* is h, let the destination list become $\mathcal{F} = \{\alpha \bmod h, (\alpha+1) \bmod h, \ldots, (\alpha+\beta) \bmod h\}$. For instance in Fig. 4(a), if $h = 5$, one node whose prefix sum is 9 and number of collected messages is 3, the original destination list is {9, 10, 11} then $\mathcal{F}$ is {4, 0, 1}. Since this node is located in DDN_1, so the three messages should be moving from DDN_1 to DDN_0, DDN_1, and DDN_4. This task can be further accomplished as follows. Let $P_{k,k'}$ located in diagonal of DDN_i, change destination list $\mathcal{F}$ as $\mathcal{F}'$, where $\mathcal{F}'$ obtained as follows. For every $t \in \mathcal{F}$, let $j = t - i$, if $j > \frac{h}{2}$ then let $j = j - h$. For instance, a destination list $\mathcal{F}$ ={4, 0, 1}, so $\mathcal{F}'$ ={-2,-1,0}, where $i = 1$ and $h = 5$. Each element of $\mathcal{F}'$ represents

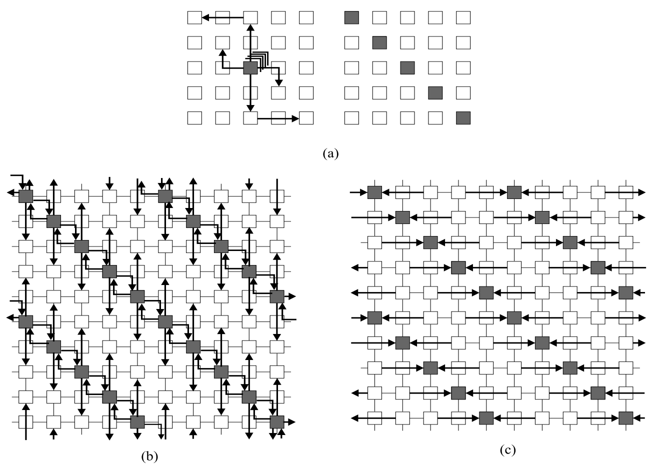

Figure 5: Data movement pattern.

the offset value from each *DDN*. Further, all of rest destination lists $\mathcal{F}'$ are displayed in Fig. 4(a).

2. Data-movement operation: Based on information of destination list $\mathcal{F}'$, a congestion-free data-movement operation is performed to balancing the load among all the *DDN*s. Suppose that $P_{k,k'}$ located in diagonal in every *DDN* and each node has messages to be exchanged with nodes $P_{k+l,k'+l}$ when $l = \pm 1$ and ± 2. Every node $P_{k,k'}$ exchanges one message with node $P_{k+l,k'+l}$, $l = \pm 1$ and ± 2, within two time steps as shown in Fig. 5(a). Figs. 5(b) and 5(c) illustrates the first and second communication pattern. There is no communication congestion. Each node $P_{k,k'}$ exchanges only one message with node $P_{k+l,k'+l}$, where $l \in \mathcal{F}'$. A hierarchial congestion-free communication pattern is worked within $2\lceil \log_5 h \rceil$ time steps. For instance, according to destination list $\mathcal{F}'$, data-movement operations are performed as shown in Figs. 4(b) and 4(c) when $l = \pm 1$ and ± 2. Each communication stage only propagates message m at most, this stage needs $2\lceil \log_5 h \rceil (T_s + mT_c)$ time units.

The time complexity of data tuning procedure is $2\lceil \log_5 h \rceil (T_s + mT_c)$.

3.2. Distribution Phase

3.2.2. Step 1: Alignment Operation

The purpose of the alignment phase lies in a preparation process for the distribution phase of multi-node broadcasting. Two procedures are needed.

1. Alignment procedure to main diagonal: All possible message are collected into the main diagonal of corresponding *DDN*. This task can be easily achieved by recursively performing the diagonal-based data redistribution operation as introduced in section 3.1. Intuitively, it takes $\lceil \log_5(\lceil \frac{n}{h} \rceil) \rceil (T_s + \widetilde{m} T_c)$ time cost, where $\widetilde{m} = \frac{s}{h} m$.

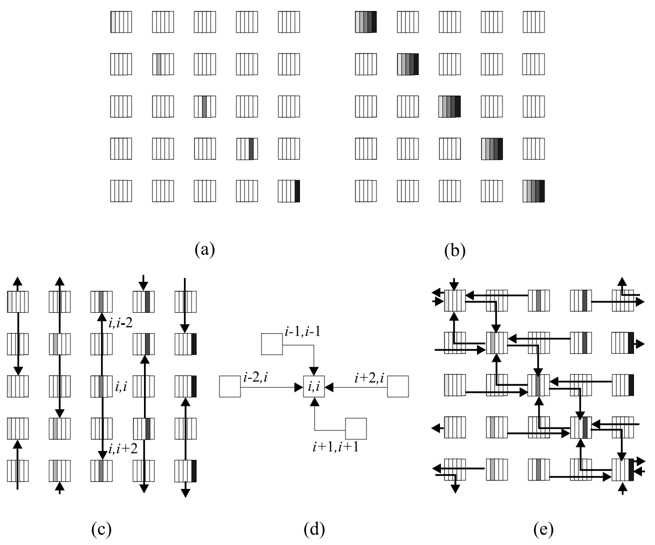

Figure 6: The all-to-all-diagonal-broadcasting procedure.

2. All-to-all broadcasting procedure on diagonal: This procedure is to collect messages of each node in the main diagonal from other nodes in the same diagonal. For instance, consider that a $T_{5\times5}$, each node in main diagonal of a *DDN* will keeps different message as shown in Fig. 6(a). After executing the all-to-all-diagonal-broadcasting operation, all nodes in main diagonal will keeps messages from all other nodes as illustrated in Fig. 6(b). The all-to-all-diagonal-broadcasting procedure is to recursively perform a data-distribution and data-collection stages as follows. For ease of presentation, we explain it by an example on a $T_{5\times5}$ as follows, (a) (Data-distribution) each node $P_{i,i}$ in diagonal distributes its own message to two neighboring nodes $P_{i,i-2}$ and $P_{i,i+2}$ according to a data-distribution pattern as shown in Fig. 6(c). (b) (Data-collection) each node $P_{i,i}$ in diagonal then collect four messages from four neighboring nodes $P_{i-1,i-1}$ and $P_{i+1,i+1}$ $P_{i-2,i}$ and $P_{i+2,i}$ according to a data-collection pattern as illustrated in Fig. 6(d). Therefore, five nodes can contain all other nodes' messages in the same diagonal as shown in Figs. 6(e) and 6(b). Repeatedly perform above data-redistribution and data-collection operations, thus an all-to-all-diagonal-broadcasting can be applied on the diagonal of *DDN* with any size. The time complexity is $2\lceil\log_5\lceil\frac{n}{h}\rceil\rceil(T_s+\widetilde{m}T_c)$, where $\widetilde{m}=\frac{s}{h}m$.

3.2.2. Step 2: Broadcast Operation

Now every node in diagonal of each *DDN* contains same broadcast message. The next step is to perform the dilated-diagonal broadcasting algorithm [9] on each *DDN* in parallel. The main diagonal $D(P_{0,0},\lceil\frac{n}{h}\rceil)$ has own all source packets data, and the broadcasting is based on a recursive structure. The main diagonal will send messages to four diagonal and let them also have the same messages. That is, each node of main diagonal sends messages to four neighbor diagonal nodes. Therefore, The time complexity of broadcast phase is $\lceil\log_5\lceil\frac{n}{h}\rceil\rceil(T_s+\widetilde{m}T_c)$, where $\widetilde{m}=\frac{s}{h}m$.

3.2.2. Step 3: Data Collection Operation

Each *data collecting network* (which is an $h\times h$ mesh), the diagonal nodes have received packets (M_0, M_1, ..., M_{h-1}). Every packet contains whole *DCN* messages. Each packet (M_0, M_1,..., M_{h-1}) have messages. These messages should be propagated to every node of the *DCN*. This is implemented in two stages: *row broadcasting* followed by *column broadcasting*.

The row broadcasting stage can be done by applying a recursive scheme. We evenly partition *DCN* into three parts. The node who is located in diagonal send its own messages to two nodes which located in trisection part of row of the *DCN* and recursive propagate to sub-trisection until the distance is one. This take $\lceil\log_3 h\rceil$ communication phases and incurs cost

$$T_1=\lceil\log_3 h\rceil(T_s+\widetilde{m}T_c).$$

Every node collects the partial messages from the row broadcasting stage. The messages are belong its column nodes, every node will concurrence send separate message to other nodes with pipelined scheme. We first embed a logical (directed) ring on each column of the *DCN*. This is done by first visiting even nodes downward the column and then odd nodes upward the column. This gives a dilation-2 embedding. With this embedding, every node then pipelines propagate its own message following the ring of the $h\times h$ *DCN*. The column broadcasting stage runs within

$$T_2=(h-1)(T_s+\widetilde{m}T_c).$$

Summing T_1 and T_2, time complexity of data collecting operation is

$$T=T_1+T_2=(\lceil\log_3 h\rceil+h-1)(T_s+\widetilde{m}T_c),\text{where } \widetilde{m}=\frac{s}{h}m.$$

4. Performance Analysis

Now we discuss performance analysis and comparison under two strategies. One is our proposed aggregation-then-distribution strategy. Another one is the well known result of *edge-disjoint-spanning-trees-based* approach [8][10][12].

Lemma 2 *The aggregation phase can be executed in a $T_{n\times n}$ torus within*

$$O((4\lceil\log_5 n\rceil+3\lceil\log_5 h\rceil+2\lceil\log_5\frac{n}{h}\rceil)T_s$$
$$+\left[(2\lceil\log_5 h\rceil+\frac{5^{\lceil\log_5 h\rceil}-1}{4})m+(4\lceil\log_5 n\rceil+2\lceil\log_5\frac{n}{h}\rceil)\right]$$
$$T_c)$$

Table 1: The comparsion table.

Strategy	Start-up comp.	Trans. comp.
EDSTs [12]	$O(2\lfloor \frac{n}{2} \rfloor T_s)$	$O(2\lfloor \frac{n}{2} \rfloor \cdot \frac{sm}{4} T_c)$
Ours	$O(\max\{\lceil \log_5 n \rceil, h\} T_s)$	$O(\max\{\lceil \log_5 \lceil \frac{n}{h} \rceil \rceil, h\} \frac{s}{h} m T_c)$

, where m is one unit message size.

Lemma 3 *The distribution phase can be executed in a $T_{n\times n}$ torus within*

$$O((4 \left\lceil \log_5 \left\lceil \frac{n}{h} \right\rceil \right\rceil + \lceil \log_3 h \rceil + h - 1) T_s$$
$$+ (4 \left\lceil \log_5 \left\lceil \frac{n}{h} \right\rceil \right\rceil + \lceil \log_3 h \rceil + h - 1) \widetilde{m} T_c)$$

, where $\widetilde{m} = \frac{s}{h} m$.

Theorem 1 *The multi-node broadcasting algorithm with aggregation-then-distribution strategy can be executed in a $T_{n\times n}$ torus within*

$$O(\lceil \log_5 n \rceil T_s + \max(\left\lceil \log_5 \left\lceil \frac{n}{h} \right\rceil \right\rceil, h) \widetilde{m} T_c)$$

, where $\widetilde{m} = \frac{s}{h} m$ and m is one unit message size.

Notably, four edge disjoint spanning trees can be constructed in a 2-D tori $T_{n\times n}$ [1], where height of spanning tree is D+2 and D=$2\lfloor \frac{n}{2} \rfloor$ is the diameter in $T_{n\times n}$. The time complexity of multi-node broadcasting in 2-D tori using edge-disjoint spanning trees based approach is $O(\lfloor \frac{n}{2} \rfloor T_s + (2\lfloor \frac{n}{2} \rfloor \cdot \frac{sm}{4}) T_c)$ illustrated in Table 1. Table 1 illustrates that multi-node broadcasting using aggregation-then-distribution strategy is more efficient than multi-node broadcasting using *edge-disjoint spanning trees-based* approach. The overall communication latency is depended on the transmission complexity. Observe that the transmission complexity of our scheme is $O(\lceil \log_5 \lceil \frac{n}{h} \rceil \rceil \frac{s}{h} m T_c)$. The transmission complexity of our scheme is better than the *edge-disjoint spanning trees-based* approach owing to $O(\lceil \log_5 \lceil \frac{n}{h} \rceil \rceil \frac{s}{h} m T_c) < O(2\lfloor \frac{n}{2} \rfloor \cdot \frac{sm}{4}) T_c)$. As a result, the multi-node broadcasting using the aggregation-then-distribution strategy is truly efficient than multi-node broadcasting using *edge-disjoint-spanning-trees-based* approach.

5. Conclusions

In this paper, we have shown how to solve the multi-node broadcast problem in a 2-D torus using an *aggregation-then-distribution* strategy. The underlying assumptions are wormhole and dimension-ordered routing, which are currently used. The main technique is to partition the torus into a certain number of independent subnetworks. Timing analyses have shown that this scheme is promising. Work is currently underway to develop multi-node broadcasting of personalized messages and to extend to higher dimensional tori and other networks.

References

[1] M. Cosnard and D. Trystram. *Parallel algorithms and Architectures*. Thomaon Computer Press, Boston MA 02116, 1995.

[2] S. E. Hambrusch, A. A. Khokhar, and Y. Lin. Scalable s-to-p broadcasting on message-passing mpps. *IEEE Transactions on Parallel and Distributed Systems*, 9(8):758–768, August 1998.

[3] R. Kesavan and D. K. Panda. Multiple multicast with minimized node contention on wormhole *k*-ary *n*-cube networks. *IEEE Transactions on Parallel and Distributed Systems*, 10(4):371–393, 1999.

[4] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*. Morgan Kaufmann Publishers, San Mateo, California, 1992.

[5] D. F. Robinson, P. K. McKinley, and B. H. Cheng. Optimal multicast communication in wormhole-routed torus networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1029–1042, October 1995.

[6] Y. Saad and M. Schultz. Data communication in hypercubes. *Journal of Parallel and Distributed Computing*, 6(1):115–135, February 1989.

[7] Y. Saad and M. Schultz. Data communication in parallel architectures. *Parallel Computing*, 11:131–150, 1989.

[8] George D. Stamoulis and John N. Tsitsiklis. An efficient algorithm for multiple simultaneous broadcasts in the hypercube. *Information Processing Letter*, 46:219–224, July 1989.

[9] Y. C. Tseng. A dilated-diagonal-based scheme for broadcast in a wormhole-routed 2d torus. *IEEE Transactions on Computers*, 46(8):947–952, August 1997.

[10] Y. C. Tseng. Multi-node broadcasting in hypercubes and star graph. *Journal of Information Science and Engineering*, 14(4):809–820, 1998.

[11] Y. C. Tseng, S. Y. Wang, and C. W. Ho. Efficient broadcasting in wormhole-routed multicomputers: A network-partitioning approach. *IEEE Transactions on Parallel and Distributed Systems*, 10(1):44–61, January 1999.

[12] E. A. Varvarigos and D. P. Bertsekas. Pratial multinode broadcast and partial exchange algorithms for d-dimension meshes. *Journal of Parallel and Distributed Computing*, 23:177–189, 1994.