# Depth Contention-Free Broadcasting on Torus Networks*

Yomin Hou
Department of Computer and Information Science
National Chiao-Tung University
Hsinchu,Taiwan, R.O.C.
gis81556@cis.nctu.edu.tw

Chien-Min Wang
Institute of Information Science
Academia Sinica
Taipei, Taiwan, R.O.C.
cmwang@iis.sinica.edu.tw

Lih-Hsing Hsu
Department of Computer and Information Science
National Chiao-Tung University
Hsinchu,Taiwan, R.O.C.
lhhsu@cis.nctu.edu.tw

## ABSTRACT

For distributed memory parallel computers, broadcast operations are widely used in a variety of applications. In this paper, we propose an efficient algorithm for broadcasting on an all-port wormhole-routed 2D torus. The underlying network is assumed to support only the dimension-ordered unicast routing. By taking the advantage of the all-port model and the distance insensitivity of the wormhole routing, the proposed algorithm can be proved to be depth contention-free, and needs only $d$ steps for broadcasting on a $2^d \times 2^d$ torus. Furthermore, the software latency is also reduced by appropriately overlaying the messages sent out by the same node in each step. The timing analysis and the computer simulation conducted in this paper clearly show the performance improvement of the proposed algorithm.

## Keywords

All-port, broadcast, depth contention-free, torus, wormhole routing.

## 1. INTRODUCTION

A *distributed memory parallel computer* consists of a large number of identical processing elements and an interconnection network. Each processing element has its own processor, local memory, and other supporting devices. Such a system is also known as a *massively parallel computer* (MPC). Processors in a MPC communicate by sending messages through the interconnection network. Communication operations may be either point-to-point or collective, depending on whether exactly two or more than two processors participate. One of the most fundamental communication operations is *broadcast*, in which the same message is delivered from a source node to all nodes in the network. Efficient broadcast communication is useful in message-passing applications, and is also necessary in several other operations, such as replication and barrier synchronization [2], which are supported in data parallel languages.

There are many different kinds of interconnection networks being used to build parallel computers. The most popular ones are $k$-ary $n$-cubes and their variants, such as rings, meshes, tori, and binary $n$-cubes. Early systems that used the store-and-forward switching often adopted a hypercube topology because its relatively dense interconnection network resulted in shorter message paths. However, many new-generation wormhole-routed MPCs use low-dimensional mesh and torus topologies. These topologies are simpler and more easily to construct than hypercubes. Although they exhibit larger internode distances, the relative distance-insensitivity of wormhole routing obviates this problem. Dally [3] had shown that low-dimensional networks have lower latency and higher hot-spot throughput than high-dimensional networks with the same bisection width.

Broadcasting on an interconnection network can be supported by either hardware or software. *nCUBE*-2 [1] is an example that supports broadcast in hardware, but it can not prevent deadlocks if two or more broadcast operations are performed simultaneously. Most existing MPCs do not support broadcast in hardware. In these environments, broadcast must be supported in software by sending multiple unicast messages. The simplest way to implement broadcast is to send a separate copy of the message directly from the source to every other node. However, this strategy is unacceptable for its poor performance. An alternative approach is to use a *broadcast tree* to improve the performance. In each message-passing step of a broadcast tree, each node holding a copy of the message forwards it to some subset of the other nodes that have not yet received it. The number of messages a node can send out concurrently is determined by the system's *port model*. In an *all-port* system, each node can send and receive messages to and from all its neighbors at the same time.

In this paper, we address the problem of broadcasting on an all-port wormhole-routed two-dimensional (2D) torus. Our method is intrigued from the idea of recursively decomposing a 2D torus to a number of smaller *building blocks*. Following the decomposition, the broadcast message is also forwarded to every node. It can be proved that the proposed broadcast algorithm is depth contention-free and needs only $d$ steps for broadcasting on a $2^d \times 2^d$ torus.

The remainder of the paper is organized as follows. Section 2 describes the specific architectural characteristics of the systems considered in this paper, and Section 3 illustrates the issues and the problems involved in supporting efficient broadcast communi-

*This work was supported by National Science Council, Republic of China, under Grant NSC87-2213-E-001-005.


ICS 98 Melbourne Australia

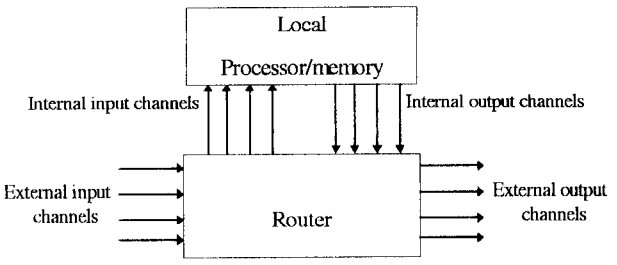


**Fig. 1. The all-port node architecture.**

cation in such systems. In Sections 4, related works are briefly discussed. Section 5 gives the details of the proposed broadcast algorithm. The performance of the proposed algorithm is evaluated in Section 6. Finally, Section 7 concludes this paper.

## 2. THE SYSTEM MODEL

The time required to move data between nodes is critical to the system performance. It can be evaluated by *communication latency*, which is the interval from the time the source node begins to send out a message until the destination node has received the message. Communication latency is composed of two parts: *software latency* and *network latency*. The software latency, including *sending latency* and *receiving latency*, is the time required for the system to handle the message at both the source and destination nodes. The network latency equals the elapsed time after the head of a message has entered the network at the source until the tail of the message emerges from the network at the destination. In addition to the time for transmitting a message through channels, network latency also contains the *blocking time*, which includes all possible delays encountered during the lifetime of a message. For example, there may be delays due to *channel contention*, i.e., some channel is required by two or more unicasts simultaneously. For software-supported broadcast, the total latency, called *broadcast latency*, is the interval from the time the source node begins to send the message until the last node has received the message. Since several message-passing steps may be required for broadcasting, broadcast latency is severely affected by the number of message-passing steps and the communication latency in each step. The way to minimize broadcast latency depends on the particular system architecture. The system architectures under consideration in this paper are 2D tori that can be characterized by three properties described below.

First, the *wormhole routing switching strategy* [9] is used. With wormhole routing, each message is divided into a number of flits. The header flit(s) carries the address information and governs the route while the remaining flits of the message follow in a pipeline fashion. One of the attractions of wormhole routing is that the storage requirement for each router is significantly less than that of the store-and-forward routing. Another attraction of wormhole routing is that its network latency is much lower than that of store-and-forward routing. In the absence of channel contention, the network latencies of wormhole routing are relatively independent of the distance between the source and destination nodes.

Second, the *all-port* architecture is utilized. In wormhole-routed MPCs, communications among nodes are handled by a separate *router*, as shown in Fig. 1. *External channles* connect the router to neighboring routers, and *internal channels* connect to its local processor. The *port model* refers to the number of internal channels at each node. If each node possesses exactly one pair of internal input/output channels, the system is called a *one-port* architecture. In a one-port architecture, a local processor must transmit messages sequentially, and messages that are destined to the same node have to be received sequentially. In the case of an *all-port* system, every external channel has a corresponding internal channel, thus allowing the node to send and receive messages to and from all its neighbors concurrently.

Finally, these systems use the deterministic *dimension-ordered routing algorithm* [9], which reserves links in a strictly increasing order of dimensions when sending messages. To provide shortest routing paths, *virtual channels* have to be added to prevent deadlock in a torus network [5]. Fig. 2 illustrates how virtual channels can be arranged along a single dimension with even width $k$. The situation is similar when $k$ is odd. There are three sets of virtual channels: $p$-channels, $l$-channels, and $h$-channels. The $p$-channels route messages that will eventually use the wraparound channel in the same dimension. The $l$-channels and $h$-channels are used after the wraparound channel has been traversed. They are also used by messages that will not use the wraparound channel in the current dimension. The $l$-channels are directed towards lower-address neighboring nodes, while higher-address neighbors are reached through $h$-channels. By accounting for the merits of the dimension-ordered routing algorithm, the designer of unicast-based collective operations may be able to eliminate channel contention, so that the performance can be improved.

## 3. THE PROBLEM

The most important issue for an efficient broadcast algorithm is to

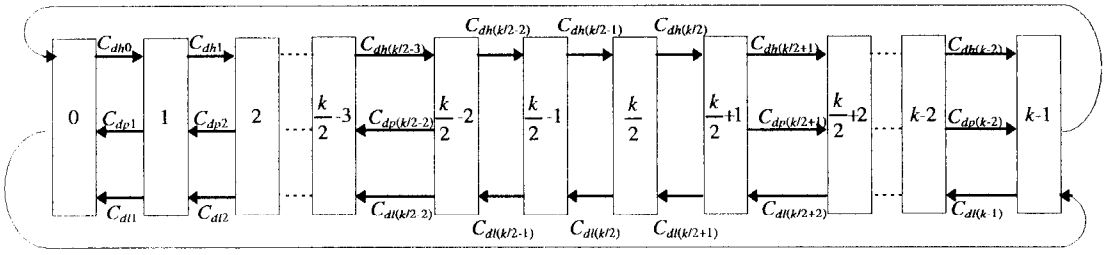


**Fig. 2. Virtual channels in one dimension of a torus.**

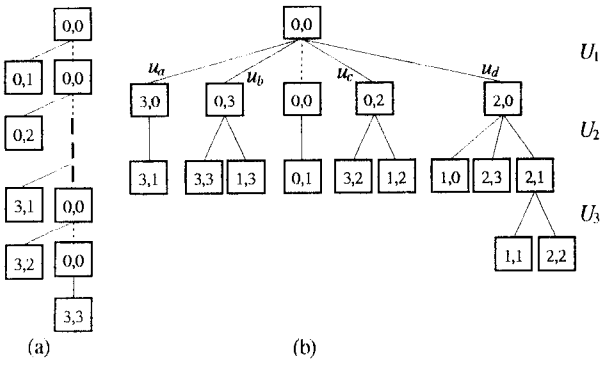


**Fig. 3. Examples of broadcasting on a 4×4 torus.**

minimize the broadcast latency. As we had mentioned in Section 2, the broadcast latency is the interval from the time the source node begins to send the broadcast message until the last node has received the message. In wormhole-routed networks that support only the unicast communication, a broadcast operation must be implemented in software by sending several unicast messages. For instance, to implement the broadcast operation in a network with $N$ nodes, we have to generate ($N$-1) unicasts so that every nodes can receive the broadcast message. If all these ($N$-1) unicasts are sent by the source node, then the broadcast latency will be the sum of the communication latencies of the ($N$-1) unicasts in the worst case. Obviously, this strategy is unacceptable for its poor performance. By organizing these unicasts as a *broadcast tree*, better performance may be obtained. Fig. 3 shows the difference of these two implementations for broadcasting on a 4×4 torus. In Fig. 3(a), node (0, 0) generates 15 unicasts to send the broadcast message. In Fig. 3(b), an example of a broadcast tree is illustrated.

The performance of a broadcast tree depends on the system's architecture, especially the switching strategy, the port model, and the unicast routing algorithm. Exploiting the distance insensitivity of wormhole routing, the communication latencies of the unicasts in a broadcast tree are approximately the same in the absence of channel contention. For this reason, in the absence of channel contention, the broadcast latency is approximate the time for performing the longest sequence of unicasts in the broadcast tree. For example, the longest sequence of unicasts in Fig. 3(b) is first from (0, 0) to (2, 0), then from (2, 0) to (2, 1), finally from (2, 1) to either (1, 1) or (2, 2). However, if channel contention happens in some unicasts, their communication latencies may increase, and the broadcast latency may also increase and becomes unpredictable.

The total software latency for performing the longest sequence of unicasts is also an important part of the broadcast latency. Although the all-port model is utilized, a processor can only handle one message at a time before the message is ready to be sent to the router through the internal output channel. Therefore, appropriately overlaying the messages sent out by each node in each step may be able to decrease the software latency of the longest sequence of unicasts in the broadcast tree. Hence, the broadcast latency may also be decreased. Consider the example shown in Fig. 3(b). There are four unicasts from source node (0, 0) to nodes (3, 0), (0, 3), (0, 2) and (2, 0), denoted by $u_a$, $u_b$, $u_c$ and $u_d$, respectively, in the first message-passing step. If $u_d$ is the last one handled by node (0, 0) in the first step, then node (2, 0) will have to wait three more sending latencies before it starts to broadcast the message. This is because node (0, 0) has to spend the time for handling the other three unicasts. Since node (2, 0) is an intermediate node in the longest sequence of unicasts, a better choice is to handle $u_d$ first. In this way, node (2, 0) can continue to forward the message to other nodes as soon as possible.

From above discussion, it is obvious that an efficient broadcast algorithm should minimize the number of unicasts in the longest sequence, i.e., minimize the number of message-passing steps, prevent channel contention, and minimize the total software latency for performing the longest sequence of unicasts. To formally define the requirements of an efficient implementation for broadcasting on a dimension-ordered wormhole-routed network, some definitions are given below and a theorem for contention-free broadcast algorithm is also proposed.

A unicast operation in a broadcast tree can be denoted as an ordered quadruple $(u, v, p(u, v), t)$ [8], where $u$ and $v$ are the source and destination nodes, respectively, $p(u, v)$ is a path based on dimension-ordered routing, and $t$ is the message-passing step of the broadcast at which the unicast is performed. Two unicasts $(u, v, p(u, v), t)$ and $(x, y, p(x, y), \tau)$ are *contention-free* if they will not contend for the same channel at the same time. Clearly, if the two paths $p(u, v)$ and $p(x, y)$ are arc-disjoint, then $(u, v, p(u, v), t)$ and $(x, y, p(x, y), \tau)$ are contention-free.

**Definition 1**: An implementation $I(B)$ of a broadcast $B$ is a sequence of unicast sets $U_1, U_2, \ldots, U_k$, satisfying the following conditions.

1) $U_1 = \{(s_0, u, p(s_0, u), 1)\}$, where $s_0$ is the source node of $B$ and $u$ is one of the other nodes in the network.
2) For every unicast $(u, v, p(u, v), t) \in U_t$, $u \neq s_0$ and $1 < t \leq k$, there must exist a set $U_j$ with $j < t$ which has $(w, u, p(w, u), j)$ as a member for some node $w$.
3) For any two unicasts $(u, v, p(u, v), t) \in U_t$ and $(x, y, p(x, y), t) \in U_t$, $1 \leq t \leq k$, if $u = x$, then the first channel of $p(u, v) \neq$ the first channel of $p(x, y)$.
4) For every node $d_i$, $d_i \neq s_0$, there exist exactly one node $w$ such that $(w, d_i, p(w, d_i), j)$ appear in $U_j$ for some integer $j$, $1 \leq j \leq k$.

The first condition in Definition 1 states that only the source node is sending messages in the first step of the implementation. The second condition guarantees that a node, except the source node, has received the message before it may forward the message to another node. The third condition implies that the messages sent by the same node in a step must use different output channels. Finally, the last condition ensures that every node receives the broadcast message exactly once. In the example illustrated in Fig. 3, there are three unicast sets in the implementation. A broadcast implementation is said to be *stepwise contention-free* if the elements in each unicast set $U_i$ are pairwise contention-free, and *depth contention-free* if any two unicasts in the broadcast implementation are contention-free. Theorem 1 gives sufficient conditions for an implementation to be depth contention-free.

**Definition 2**: Given a broadcast implementation $I(B) = \{U_1, U_2, \ldots, U_k\}$, a node $v$ is in the reachable set of node $u$, denoted as $R_u$, if and only if $v = u$ or there exists $j$, $1 \leq j \leq k$, such that $(w, v, p(w, v), j) \in U_j$ for some node $w \in R_u$.

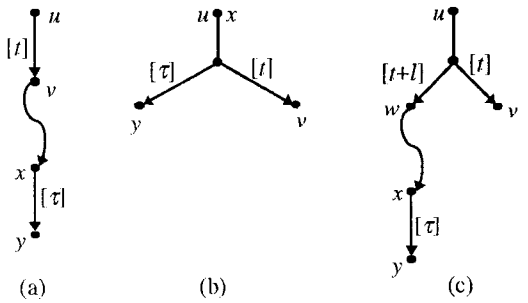


**Fig. 4. Illustrations of Conditions 1, 3 and 4 in Theorem 1.**

**Theorem 1**: Given a broadcast implementation $I(B)$, if at least one of the following four conditions holds for every pair of unicasts $(u, v, p(u, v), t)$ and $(x, y, p(x, y), \tau)$ in $I(B)$, where $t \le \tau$, then $I(B)$ is depth contention-free.

1) $x \in R_v$.
2) $p(u, v)$ and $p(x, y)$ are arc-disjoint.
3) $u=x$ and the first channel in $p(u, v)$ is also the first channel in $p(x, y)$.
4) $x \in R_w$, $(u, w, p(u, w), t+l) \in I(B)$ for some node $w$ and positive integer $l$, and the first channel in $p(u, v)$ is also the first channel in $p(u, w)$.

Proof: We shall show that contention does not arise between any pair of unicasts in the implementation. Consider two arbitrary unicasts $(u, v, p(u, v), t)$ and $(x, y, p(x, y), \tau)$, with $t \le \tau$.

*Condition* 1: If $x \in R_v$, then the $u$-to-$v$ unicast must be completed before the $x$-to-$y$ unicast begins, as shown in Fig. 4(a). Clearly, they are contention-free. (Note: $u \notin R_y$ since $t \le \tau$.)

*Condition* 2: If the two paths of the messages, $p(u, v)$ and $p(x, y)$, are arc-disjoint, then, by definition, the two unicasts must be contention-free.

*Condition* 3: If $u=x$ and the first channel in $p(u, v)$ is also the first channel in $p(x, y)$, then we can derive $t<\tau$ since $u$ can send only one message through the same internal output channel at a time as defined in Definition 1. Fig. 4(b) illustrates this situation. Note that $u$ sends the message to $v$ before sending it to $y$. Even if $\tau=t+1$ and the sending latency is 0, no contention will happen.

*Condition* 4: As showing in Fig. 4(c), node $u$ must complete sending the message to node $v$ before it can start to send the message to node $w$ because the first channel in $p(u, v)$ is also the first channel in $p(u, w)$. Since node $w$ is either an ancestor of $x$ or $x$ itself, clearly, node $v$ will receive the message prior to node $x$. This prevents channel contention. □

# 4. RELATED WORKS

The problem of implementing collective operations in wormhole-routed networks has been studied extensively. A recent survey can be found in [7]. In terms of the one-port model, Mckinley, Xu, Esfahanian and Ni [8] proposed a recursive doubling process for multicast communication in meshes and hypercubes. This process was also extended to tori [10], and proved to be depth contention-free. For the all-port model, Ho and Kao [6] developed an optimal broadcast algorithm for hypercubes. On an $n$-dimensional hypercube, it needs only $\theta(n/log_2(n+1))$ steps and is also proved to be depth contention-free.

On all-port meshes and tori, Tsai and Mckinley [11], [12] proposed the *extended dominating nodes* (*EDN*) approach for broadcasting. When broadcasting on a $2^d \times 2^d$ torus, their algorithm constructs $(d+1)$ *extended dominating sets* (*EDSs*), one for each level. The highest level EDS contains only the source node, while the level_0 EDS is the set of all nodes in the network. Each node in a level_$i$ EDS, $1 \le i \le d$, follows some pattern to send the message to nodes in the level_$(i-1)$ EDS. Their algorithm requires $d$ steps for broadcasting and is proved to be stepwise contention-free. However, through computer simulation, it can be shown that their algorithm is not depth contention-free.

For broadcasting on an all-port 2D torus with arbitrary size, Tseng [13] proposed a *dilated-diagonal-based scheme*. On a square $n \times n$ torus, $2\lceil \log_5 n \rceil + 1$ steps are required for Tseng's algorithm. On a nonsquare $n_1 \times n_2$ torus, $n_1 < n_2$, the number of steps depends on $n_1$. If $n_1$ is even, $(\lceil \log_5 n_1 \rceil + \lceil \log_5 \frac{n_1}{2} \rceil + \lceil \log_5 \frac{n_2}{n_1} \rceil + 2)$ steps are required; if $n_1$ is odd, one more step is required. Although Tseng's algorithm can handle 2D tori with arbitrary size, it can not guarantee to be depth contention-free either.

In this paper, we also deal with the problem of broadcasting on a $2^d \times 2^d$ all-port torus. The proposed algorithm requires $d$ steps and is proved to be depth contention-free. Moreover, the software latency is also minimized by the proposed algorithm.

# 5. THE PROPOSED DCF ALGORITHM

In an $n$-dimensional torus, each node has two neighboring nodes in each dimension. If the all-port model is utilized, a node can send messages to at most $2n$ nodes simultaneously. Thus, there exists a theoretical lower bound, $\lceil \log_{2n+1}(N) \rceil$, on the number of message-passing steps for broadcasting on an all-port $n$-dimensional torus, where $N$ is the total number of nodes in the network. However, this lower bound may not be achievable under the constrain of dimension-ordered routing. Because of the constrain of dimension-ordered routing, it is not always possible for every node to send the message to $2n$ new nodes in every step. In the literature, the EDN algorithm [12] and the Tseng's algorithm [13] are both efficient for broadcasting on an all-port wormhole-routed torus. They require $d$ and $2\lceil \log_5 2^d \rceil + 1$ steps respectively for broadcasting on a $2^d \times 2^d$ torus. Although they can be proved to be stepwise contention-free, channel contention may occur between two unicasts in different steps. To eliminate channel contention, a new algorithm is proposed in this section. The proposed algorithm also needs only $d$ steps for broadcasting on a $2^d \times 2^d$ torus. Moreover, it can be proved to be depth contention-free and the software latency is also reduced by appropriately overlaying

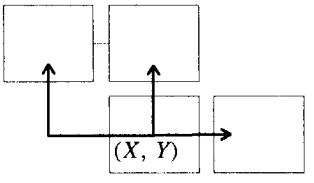


**Fig. 5. An example of the building block, Z_block.**

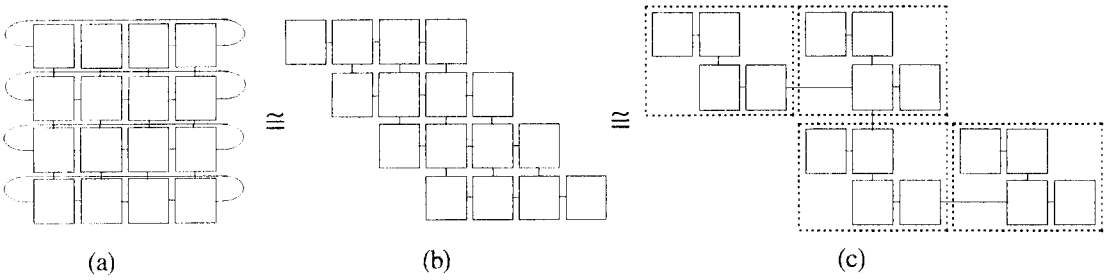


**Fig. 6. Decompose a 4×4 torus into four level_1 Z_blocks.**

the messages sent out by the same node in each step.

## 5.1 Building Blocks

The proposed broadcast algorithm is based on *building blocks*. A building block of a network consists of a set of units. Each unit is either a smaller building block or a node in the network. For convenience, a single node can be viewed as the degenerate case of a building block. Nodes in a building block are assumed to be able to communicate with each other through the paths determined by the unicast routing algorithm. One of the units in a building block is called the *block source unit*, who takes the responsibility of broadcasting messages to the other units. A building block is called a *level_1 block* if there is only one node in each unit. The block source unit of a level_1 block can also be called a *block source node*. A level_$i$ building block can be constructed recursively so that each unit in a level_$i$ building block is a level ($i$-1) building block. Fig. 5 shows an example of a building block, called *Z_block*. There are 4 units in a Z_block. The block source node, ($X$, $Y$), of a level_1 Z_block can broadcast a message to the other three nodes, ($X$-1, $Y$+1), ($X$, $Y$+1) and ($X$+1, $Y$), in one step.

A 4×4 torus can be decomposed recursively into four level_1 Z_blocks, as shown in Fig. 6. For clarity, some channels are not painted in Fig. 6. Since there are wraparound channels in a torus, a 4×4 torus can be viewed as shown in Fig. 6(b). Fig. 6(c) illustrates the recursive decomposition. Note that the four dashed units in Fig. 6(c) form a level_2 Z_block. The concept of the above decomposition can be generalized so that a $2^d \times 2^d$ torus can be viewed as a level_$d$ Z_block.

From the above example, it can be observed that a torus can be viewed as a high-level building block. The block source unit in the high-level building block can send the broadcast message to the other units. Hence, following the recursive decomposition of the high-level building block, the message can be sent to every nodes in the network. This broadcast algorithm can be applied regardless of the location of the source node since the torus is a node symmetric topology.

## 5.2 The DCF Building Block

The above subsection has illustrated the building-block-based approach for broadcasting. Although the Z_block can be used for broadcasting on a $2^d \times 2^d$ torus, it is not depth contention-free. Moreover, in addition to the network latency, it needs 3 sending latencies and 1 receiving latency for broadcasting a message from the block source unit to the other three units in a Z_block. In order to find a depth contention-free broadcast algorithm with fewer software latency, a different building block, as shown in Fig. 7, is proposed. It is called a *DCF_block*, which consists of 16

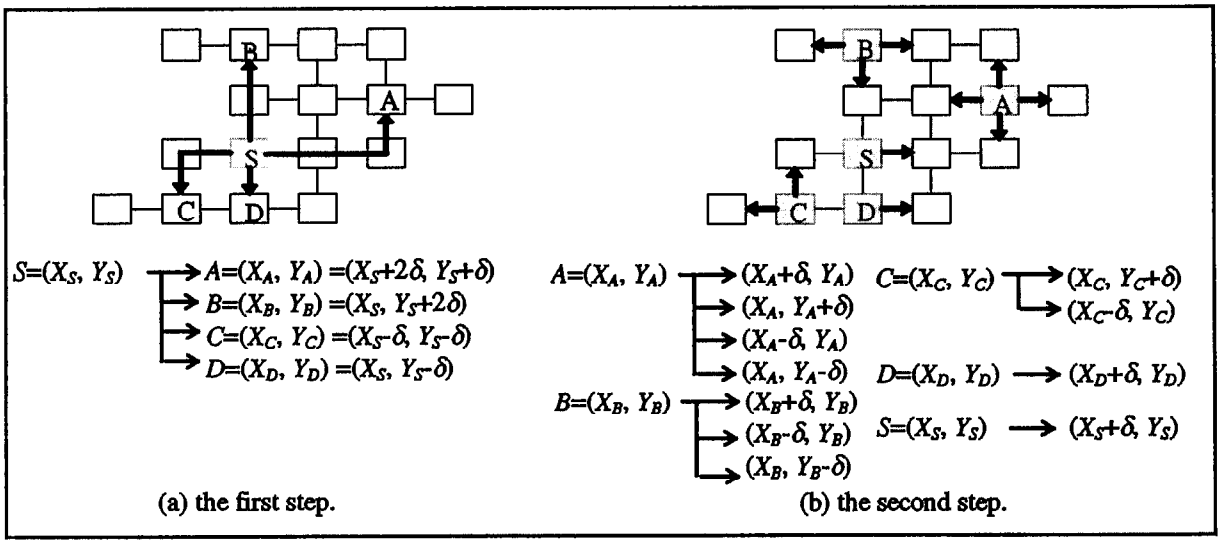


**Fig. 7. Broadcasting on the level_1 DCF_block.**

units. It takes two steps for broadcasting on a DCF_block. Fig. 7 illustrates the broadcast algorithm for a level_$i$ DCF_block, where $\delta=4^{(i-1)}$. In the first step, the source node $S$ in the block source unit sends the broadcast message to nodes $A$, $B$, $C$ and $D$ as shown in Fig. 7(a). In the second step, nodes $S$, $A$, $B$, $C$ and $D$ send the message to their neighboring units as shown in Fig. 7(b).

It can be observed that a 4×4 torus can be viewed as a level_1 DCF_block. Similarly, a 16×16 torus can be recursively decomposed into 16 level_1 DCF blocks, and be viewed as a level_2 DCF block. In general, a $4^k\times4^k$ torus can be viewed as a level_$k$ DCF block. For broadcasting on a level_$k$ DCF_block, it takes $k$ phases, and two steps in each phase. In the first phase, the source node $S$ sends the message to another 15 nodes, as shown in Fig. 7. After the first phase, each of the block source units of the 16 level_($k$-1) DCF_blocks has got a copy of the broadcast message and continue to broadcast it. Note that, all the block source units of level_($k$-$j$) DCF_blocks can get a copy of the broadcast message at the end of phase $j$, and then send the message to the block source units of level ($k$-$j$-1) DCF_blocks in the next phase. The messages sent by the same node in each step are handled in the order as shown in Fig. 7. We will show in Section 6 that the software latency can be reduced in this order.

To handle a 2D torus of size $(2\times4^k)\times(2\times4^k)$, we can define a level_2 DCF_block to be composed of 16 level_1 Z_blocks rather than 16 level_1 DCF_blocks. Hence, a $(2\times4^k)\times(2\times4^k)$ torus can be viewed as a level ($k$+1) DCF block. When broadcasting, ($k$+1) phases are needed. In each phase $j$, $1\le j\le k$, the detail of each step is the same as shown in Fig. 7, except $\delta=2\times4^{(k-j)}$. The only difference is in the last phase. There is only one step in the last phase as shown in Fig. 5. Therefore, it needs $2k$+1 steps for broadcasting on a $(2\times4^k)\times(2\times4^k)$ torus.

**Theorem 2**. The proposed algorithm delivers a message exactly once to every node in a $4^k\times4^k$ torus in $2k$ message-passing steps, for any $k\ge1$.

Proof: Since a $4^k\times4^k$ torus can be viewed as a level_$k$ DCF_block, we can prove this theorem by showing that every node in a level_$k$ DCF_block can receive the broadcast message exactly once in $2k$ steps.

The proof is by induction on $k$. For $k$=1, as shown in Fig. 7, every node can receive the broadcast message exactly once in 2 steps.

Assume that the result is true for $k=l$, $l\ge1$, i.e., every node in a level_$l$ DCF_block can receive the broadcast message exactly once in $2l$ steps. Now, consider broadcasting on a level_($l$+1) DCF_block. As shown in Fig. 7, after the two message-passing steps in the first phase, one of the nodes in each block source unit of the 16 level_$l$ DCF_blocks received the message exactly once. Then, they take the responsibility of broadcasting in each of the 16 level_$l$ DCF_blocks. According to the assumption, this can be done in $2l$ steps and every node can receive the broadcast message exactly once. Therefore, broadcasting on a level_($l$+1) DCF_block can be completed in 2($l$+1) steps, and every node can receive the broadcast message exactly once. By mathematic induction, this theorem must be true for any $k\ge1$. □

**Corollary 1**. The proposed algorithm delivers a message exactly once to every node in a $(2\times4^k)\times(2\times4^k)$ torus in $2k$+1 message-passing steps, for any $k\ge1$.

**Corollary 2**. The proposed algorithm delivers a message exactly once to every node in a $2^d\times2^d$ torus in $d$ message-passing steps.

**Theorem 3**. The proposed algorithm for a $4^k\times4^k$ torus network is stepwise contention-free.

Proof: Since a $4^k\times4^k$ torus can be viewed as a level_$k$ DCF_block, we can prove this theorem by showing that it is stepwise contention-free for broadcasting on a level_$k$ DCF_block.

First, it can be observed in Fig. 7 that unicasts in the same message-passing step for a DCF_block must be stepwise contention-free. Besides, the routing paths of the unicasts use only the channels within the DCF_block. Since DCF_blocks of the same level must be disjoint, channel contention can not happen between unicasts in different DCF_blocks. Therefore, the proposed algorithm must be stepwise contention-free. □

**Corollary 3**. The proposed algorithm for a $(2\times4^k)\times(2\times4^k)$ torus is stepwise contention-free.

**Theorem 4**. The proposed algorithm for a $4^k\times4^k$ torus is depth contention-free.

Proof: Since a $4^k\times4^k$ 2D torus can be viewed as a level_$k$ DCF_block, we can prove this theorem by showing that there is no contention for broadcasting on a level_$k$ DCF_block.

The proof is by induction on $k$. For $k$=1, as shown in Fig. 7, only two of the 15 unicasts may use the same channel. One is from $S$ to $(X+2,Y)$ in step 1, and the other one is from $S$ to $(X+1,Y)$ in step 2. Since these two unicasts are in different steps and their first channel is the same one, from Condition 3 in Theorem 1, no contention may happen. Hence, broadcasting on the level_1 DCF_block is depth contention-free.

Assume that broadcasting on the level_$l$ DCF_block is depth contention-free. Now, consider the case for the level_($l$+1) DCF_block. From the assumption, there is no contention between unicasts in the same level_$l$ DCF_block. Since channels in different level_$l$ DCF_blocks must be disjoint, there is no channel contention between unicasts in different level_$l$ DCF_blocks. To prove that broadcasting on the level_($l$+1) DCF_block is depth contention-free, we only need to show that no unicast in the first phase will contend for any channel with other unicasts.

From the reasons similar to the above discussion for $k$=1, we can prove that no contention may happen between any two of the 15 unicasts in the first phase for broadcasting on the level_($l$+1) DCF_block. In the following, it is proved that no unicast in the first phase will contend for any channel with unicasts in other phases. First, consider the unicast ($S$, $A$, $P(S,A)$, 1) in the first phase, where $S=(X,Y)$ is the source node and $A=(X+2\delta,Y+\delta)$. It can be observed that ($S$, $A$, $P(S,A)$, 1) passes channels of four level_$l$ DCF_blocks. These four level_$l$ DCF_blocks are denoted as $Q_a$, $Q_b$, $Q_c$ and $Q_d$, and the their source nodes are $S$, $(X+\delta,Y)$, $(X+2\delta,Y)$ and $A$, respectively. In the following, we shall show that there is no channel contention between ($S$, $A$, $P(S,A)$, 1) and unicasts in these four level_$l$ DCF_blocks.

1. From the algorithm shown in Fig. 7 and the constraint of the dimension-ordered routing, unicasts in $Q_a$ that may contend for the same channel with ($S$, $A$, $P(S,A)$, 1) must be generated by nodes passed by ($S$, $A$, $P(S,A)$, 1). Let $x$ be the source node of such an unicast. If $x=S$, the first channel of

the unicast must be the same as the first channel of $P(S,A)$. From Condition 3 in Theorem 1, no contention may happen. Suppose that $x \neq S$. There must exist an ancestor $W$ of $x$ that receives the message from $S$ and the first channel from $S$ to $W$ is the same as the first channel of $P(S,A)$. From Condition 4 in Theorem 1, there is no contention between ($S$, $A$, $P(S,A)$, 1) and the unicast. Hence, there is no contention between ($S$, $A$, $P(S,A)$, 1) and any unicast in $Q_a$.

2. Node $(X+\delta,Y)$ is an ancestor of the source nodes of all the unicasts in $Q_b$ and it will receive the broadcast message from node $S$ in step 2. Since the first channel from $S$ to $(X+\delta,Y)$ is the same as the first channel of $P(S,A)$, from Condition 4 in Theorem 1, there is no contention between ($S$, $A$, $P(S,A)$, 1) and unicasts in $Q_b$.
3. Since the source node of each unicast in $Q_c$ and $Q_d$ is in the reachable set of $A$, from Condition 1 in Theorem 1, there is no contention between ($S$, $A$, $P(S,A)$, 1) and unicasts in $Q_c$ and $Q_d$.

For the above discussion, we can conclude that there is no contention between ($S$, $A$, $P(S,A)$, 1) and other unicasts. The other unicasts in the first phase can also be proved to be contention-free with any other unicast in a similar way. Hence, there is no contention when broadcasting on a level $(l+1)$ DCF_block. Therefore, the proposed algorithm for a $4^k \times 4^k$ torus is depth contention-free.□

**Corollary 4**. The proposed algorithm for a $(2\times4^k)\times(2\times4^k)$ torus is depth contention-free.

## 6. PERFROMANCE EVALUATION

In this section, the performance of the proposed algorithm is evaluated through the timing analysis and computer simulation. For comparison, the performance of the EDN broadcast algorithm is also evaluated. From the results, the performance improvement of the proposed broadcast algorithm over the EDN algorithm can be clearly observed.

### 6.1 Timing Analysis

As we have proved in Section 5, the proposed algorithm needs only $d$ steps for broadcasting on a $2^d \times 2^d$ torus, and is depth contention-free. In order to better understand the performance of the proposed algorithm, the time required for broadcasting will be analyzed. In the following analysis, $t_s$ represents the sending latency incurred for each message; $t_r$ is the receiving latency; the time required for transmitting a flit on a channel is $t_c$; $L$ is the length of the broadcast message, in flits; and $h$ denotes the distance between the source and destination nodes of an unicast. Thus, in the absence of channel contention, the communication latency of an unicast is $t_s + ht_c + Lt_c + t_r$. Suppose that a node sends the broadcast message to $m$ different destination nodes in one step. Since the sending latencies of these $m$ unicasts must be serialized, in the absence of channel contention, the time required from the beginning of the first unicast to the receipt of the message by the $i$th destination is $it_s + h_i t_c + Lt_c + t_r$, where $h_i$ is the distance of the $i$th unicast handled by the node. Using this formula as a basis, the broadcast latency of the proposed algorithm and the EDN algorithm are analyzed in the following.

For the first step in a phase of the proposed algorithm, there are four unicasts from $S$ to $A$, $B$, $C$ and $D$. Suppose that $S$ begins to send the message at time instant 0, then $A$, $B$, $C$ and $D$ receive the message at time instants $(t_s + 3\delta t_c + Lt_c + t_r)$, $(2t_s + 2\delta t_c + Lt_c + t_r)$, $(3t_s + 2\delta t_c + Lt_c + t_r)$ and $(4t_s + \delta t_c + Lt_c + t_r)$, respectively. Once they receive the message, they begin the second step in the phase. Hence, the last unicasts sent out by $A$, $B$, $C$, $D$ and $S$ in the second step must be completed at time instants $(5t_s + 4\delta t_c + 2Lt_c + 2t_r)$, $(5t_s + 3\delta t_c + 2Lt_c + 2t_r)$, $(5t_s + 3\delta t_c + 2Lt_c + 2t_r)$, $(5t_s + 2\delta t_c + 2Lt_c + 2t_r)$ and $(5t_s + \delta t_c + 2Lt_c + t_r)$, respectively. Obviously, the time for completing a phase is $(5t_s + 4\delta t_c + 2Lt_c + 2t_r)$. Since there are $k$ phase for broadcasting on a $4^k \times 4^k$ torus, the broadcast latency is $\sum_{i=1}^{k}(5t_s + 4\delta_i t_c + 2Lt_c + 2t_r)$, where $\delta_i = 4^{k-i}$. This summation equals to $(5kt_s + \frac{4(4^k-1)}{3}t_c + 2kLt_c + 2kt_r)$. Similarly, the broadcast latency on a $(2\times4^k)\times(2\times4^k)$ torus can be derived to be $[\,(5k+3)t_s + \frac{2(4^{(k+1)}-1)}{3}t_c + (2k+1)Lt_c + (2k+1)t_r\,]$.

The time for the EDN broadcast algorithm can be analyzed in the

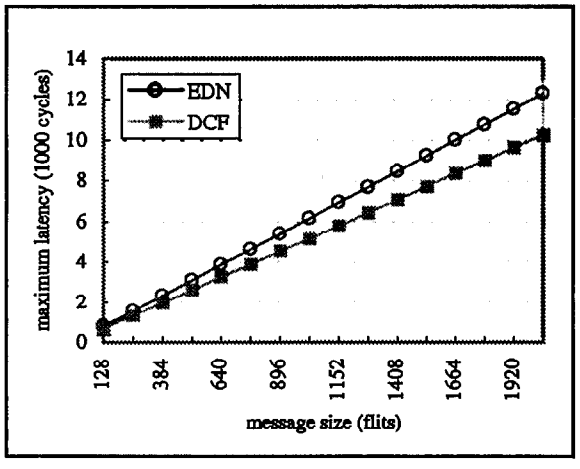


**Fig. 8. Broadcasting on a 32×32 torus when sending and receiving latencies are 0.**

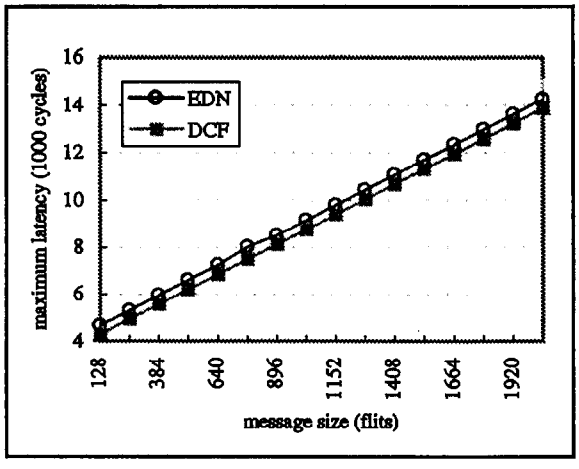


**Fig. 9. Broadcasting on a 32×32 torus when sending and receiving latencies are 200 cycles.**

same way. Without considering the possible channel contention, the broadcast latency is $(6kt_s + \frac{4(4^k-1)}{3}t_c + 2kLt_c + 2kt_r)$ on a $4^k \times 4^k$ torus, and $[\ 3(2k+1)t_s + \frac{2(4^{(k+1)}-1)}{3}t_c + (2k+1)Lt_c + (2k+1)t_r\ ]$ on a $(2\times4^k)\times(2\times4^k)$ torus [12]. Compared with the proposed algorithm, the EDN algorithm spends $k$ more sending latencice and the blocking time that may be caused by the possible channel contention.

## 6.2 Simulation Study

To investigate the performance improvement of the proposed approach, some experiments are made by simulating the network behavior of 2D tori. The performance measure is the maximum latency. Given a message size, node (0, 0) is assumed to broadcast the message to all the other nodes. The simulated channel rate is one cycle per flit. Fig. 8 and Fig. 9 show the simulation result of the proposed DCF algorithm and the EDN algorithm over different message lengths on a 32×32 torus. In Fig. 8, the sending latency and the receiving latency are both set to 0. In Fig. 9, the sending latency and the receiving latency are both set to 200 cycles. It can be observed that in both cases, the performance of the proposed algorithm is better than that of the EDN algorithm.

In Fig. 8, the effect of channel contention can be observed. The EDN algorithm takes about one more communication latency than the proposed algorithm because of the blocking time caused by channel contention. The difference becomes more significant as the message size increases. The benefit for overlaying the messages sent out by each node in each step can be observed in Fig. 9. The broadcast latency of the proposed algorithm is about 200 cycles fewer than that of the EDN algorithm, which is rather significant for small message sizes.

Compared the simulation results with the timing analysis, it can be noted that the broadcast latency of the proposed algorithm is very close to what we have analyzed because it is depth contention-free. However, the broadcast latency of the EDN algorithm is unpredictable due to the possible channel contention.

# 7. CONCLUSION

In this paper, we propose a new algorithm based on building blocks to broadcast messages on an all-port wormhole-routed 2D torus. The underlying network is assumed to support only the dimension ordered unicast routing. By taking the advantage of the all-port model and the distance insensitivity of wormhole routing, we have achieved the goal of $d$ steps for broadcasting on a $2^d \times 2^d$ torus, and no channel contention between any two constituent unicast messages. Furthermore, the software latency is also reduced by appropriately overlaying the messages sent out by the same node in each step. The timing analysis and the computer simulation clearly show the performance improvement of the proposed algorithm.

# REFERENCES


[1] *nCUBE 2 Supercomputers Manual, NCUBE Company*, 1990.

[2] Tim S. Axelrod, "Effects of synchronization barriers on multiprocessor performance," *Parallel Computing*, Vol. 3, No. 2, May. 1986, pp. 129-140.

[3] Willian J. Dally, "Performance Analysis of $k$-ary $n$-cube Interconnection Networks," *IEEE Transactions on Computers*, Vol. 39, No. 6, Jun. 1990, pp. 775-785.

[4] Willian J. Dally and Charles L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, Vol. 1, No. 3, 1986, pp.187-196.

[5] Willian J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, May. 1987, pp. 547-553.

[6] Ching-Ten Ho and Ming-Yang Kao, "Optimal Broadcast in All-Port Wormhole-Routed Hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 2, Feb. 1995, pp. 200-204.

[7] Philip K. McKinley, Yih-jia Tsai, and David F. Robinson, "Collective Communication in Wormhole-routed Massively Parallel Computers," *IEEE Computer*, Vol. 28, No. 12, Dec. 1995, pp. 39-50.

[8] Philip K. McKinley, Hong Xu, Abdol-Hossein Esfahanian, and Lionel M. Ni, "Unicast-Based Multicast Communication in Wormhole-Routed Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 12, Dec. 1994, pp. 1252-1265.

[9] Lionel M. Ni and Philip K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, Vol. 26, Feb. 1993, pp. 62-76.

[10] David F. Robinson, Philip K. McKinley and Betty H.C. Cheng, "Optimal Multicast Communication in Wormhole-Routed Torus Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 10, Oct. 1995, pp. 1029-1042.

[11] Yih-jia Tsai and Philip K. McKinley, "An Extended Dominating Node Approach to Collective Communication in Wormhole-Routed 2D Meshes," *Proc. Scalable High Performance Computing Conf.*, May. 1994, pp. 199-206.

[12] Yih-jia Tsai and Philip K. McKinley, "A Broadcast Algorithm for All-Port Wormhole-Routed Torus Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 8, Aug. 1996, pp. 876-885.

[13] Yu-Chee Tseng, "A Dilated-Diagonal-Based Scheme for Broadcast in a Wormhole-Routed 2D Torus," *IEEE Transactions on Computers*, Vol. 46, No. 8, Aug. 1997, pp. 947-952.