Circuit-Switched Broadcasting in Torus Networks

Joseph G. Peters and Michel Syska

Abstract—In this paper we present three broadcast algorithms and lower bounds on the three main components of the broadcast time for 2-dimensional torus networks (wrap-around meshes) that use synchronous circuit-switched routing. The first algorithm is based on a recursive tiling of a torus and is optimal in terms of both *phases* and intermediate switch settings when the start-up time to initiate message transmissions is the dominant cost. It is the first broadcast algorithm to match the lower bound of $\log_5 N$ on number of phases (where N is the number of nodes). The second and third algorithms are hybrids which combine circuit-switching with the pipelining and arc-disjoint spanning trees techniques that are commonly used to speed up store-and-forward routing. When the propagation time of messages through the network is significant, our hybrid algorithms achieve close to optimal performance in terms of phases, intermediate switch settings, and total transmission time. They are the first algorithms to achieve this performance in terms of all three parameters simultaneously.

Index Terms—Broadcasting, torus networks, circuit-switched routing, tilings, pipelining.

1 Introduction

Distributed memory multicomputer systems in which the processors communicate by exchanging messages over an interconnection network are an increasingly popular method for achieving cost-effective high-performance computing. The performance of a message-passing system is strongly dependent on the topology of the interconnection network and on the routing mechanism that is used to move information around the network. *Multidimensional tori (wrap-around meshes, toroidal meshes, k-ary n-cubes)* are currently popular interconnection networks because their low degrees permit efficient layouts and construction with standard components [10]. The relatively large diameters of tori are a disadvantage when *store-and-forward* routing is used because communication time for store-and-forward routing is proportional to the diameter of the network. Store-and-forward routing has been displaced by *circuit-switched* routing in many recent multicomputer systems such as the Cray T3D [24], Fujitsu AP1000 [30], INMOS T9000 [34], Intel Paragon [33], iWarp [6], [7], and nCUBE-2 [32]. Since the cost of circuit-switched routing is less dependent than store-and-forward routing on the diameter of a network, torus networks with circuit-switched routing are a practical choice.

Broadcasting is a one-to-all information dissemination problem in which information originating at one node of a network must be distributed to all other nodes of the network. The broadcasting problem has been studied for many different network topologies and routing strategies. Until recently, research on broadcasting concentrated on *unit cost* store-and-forward models in which each message transmission travels along one communication link and takes one unit of time. Much of the recent research on broadcasting has used *linear cost* models in which the propagation time of a message is proportional to the length of the message. Linear cost models have been used to study both store-and-forward routing [12], [16], [26], [31] and various types of circuit-switched routing including *direct connect* [23], *virtual cut-through* [17], and *wormhole* routing [8], [10], [28]. See [15] for a thorough survey of earlier work on unit cost models and [13] for a recent survey of store-and-forward routing under both unit cost and linear cost models.

In this paper, we present three new broadcast algorithms for 2-dimensional torus networks which use circuit-switched routing and a linear cost model. Our first algorithm, based on tilings of a torus, minimizes the number of sequential message transmissions, or *phases*, in a broadcast assuming that the start-up time to initiate message transmissions is the dominant cost. Our second algorithm, based on divide-and-conquer, and our third algorithm, based on *H-trees*, are intended for situations in which the total time for a broadcast can be reduced by splitting messages into packets for simultaneous transmission at the expense of an increase in the total cost to initiate transmissions.

We prove that our tiling algorithm is optimal when the messages are short or when the time to initiate a message transmission is much larger than the unit propagation time of a message along a link. The latter situation is the case in many current multiprocessor networks. When the start-up time for message transmissions is negligible or when messages are very long, optimal asymptotic performance can be obtained by simulating a store-and-forward algorithm based

• *J. Peters is with the School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6.E-mail: peters@cs.sfu.ca.*
• *M. Syska is with Laboratoire I3S - CNRS - URA 1376, Université de Nice - Sophia Antipolis, 06903 Sophia-Antipolis Cedex, France. E-mail: michel.syska@alto.unice.fr.*

Manuscript received June. 10, 1993; revised Apr. 30, 1994. This research was done while Mr. Peters was visiting CNRS, Université de Nice—Sophia Antipolis, and while Mr. Syska was visiting the School of Computing Science, Simon Fraser University.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number D95085.

on the arc disjoint spanning trees in [22]. However, any store-and-forward algorithm will use exponentially more phases than our first algorithm. Our divide-and-conquer and H-trees algorithms are new circuit-switching algorithms that achieve asymptotic performance very close to a store-and-forward algorithm for long messages using numbers of phases very close to our tiling algorithm. These algorithms are well suited to applications that involve the broadcasting of large amounts of data, such as linear algebra computations that broadcast large arrays of floating point numbers (see [11] for example). Both the divide-and-conquer and H-trees algorithms will outperform the store-and-forward algorithm in all cases except the extreme case in which the time to transmit a message is more than exponentially longer than the time to initiate a message transmission. They are the first algorithms to simultaneously achieve close to optimal performance (i.e., within small constant factors) in terms of both start-up time and propagation time through the network.

In the next section, we describe both store-and-forward and circuit-switched routing and define the linear cost model for both types of routing. We also survey previous work on broadcasting in torus networks. In the third section, we give lower bounds on the various components of the linear cost model. In Sections 4 and 5, we develop and analyze our new algorithms. We conclude with a comparison of our algorithms with store-and-forward algorithms and with our lower bounds.

2 Models of Communication

In a $p \times q$ 2-dimensional torus network, each node has a label (i, j) and four neighbors $(i, j+1)$, $(i, j-1)$, $(i+1, j)$ and $(i-1, j)$ where the first component of a label is an integer mod p and the second is an integer mod q. The diameter of a $p \times q$ torus is $D = \left\lfloor \frac{p}{2} \right\rfloor + \left\lfloor \frac{q}{2} \right\rfloor$. We will use the *link-bounded* [13] (or *shouting* [15]) model of communication in which a processor can use all of its communication links simultaneously. In contrast, the *processor-bounded* (or *whispering*) model permits the use of only one link at any given time. We also assume that the communication links are *full-duplex* so that messages can travel in both directions simultaneously. Thus, in a 2-dimensional torus network, each node has four ports for incoming messages and 4 ports for outgoing messages. We assume that a node can *switch through* a message by connecting an input port to an output port. In a 2-dimensional torus, as many as four messages can be switched through a node simultaneously.

When *store-and-forward* routing is used to send a message along a path of d links, the message is stored in buffers at intermediate nodes on the path. An intermediate node does not begin to send the message to the next node on the path until it has received the entire message. Thus, the transmission time for a message of length L to be sent distance d in the linear cost model is $d(\beta + L\tau)$ where β is the time to initiate a message transmission and $1/\tau$ is the bandwidth of the communication links. (We assume that all links have the same bandwidth.) The total time to complete a broadcast can be reduced by partitioning the message into packets and using a *pipelining* technique to send the packets consecutively along the communication links [26], [31], [16]. The time can be further reduced by distributing the broadcast over several *arc-disjoint spanning trees*. Using pipelining and k arc-disjoint spanning trees of maximum depth h, and choosing the packet size carefully, gives a propagation time of $\left(\sqrt{(h-1)\beta} + \sqrt{L\tau / k}\right)^2$ [3]. Combining this result with the four arc-disjoint spanning trees of depth $D + 1 = \left\lfloor \frac{p}{2} \right\rfloor + \left\lfloor \frac{q}{2} \right\rfloor + 1$ from [22] gives time $\left(\sqrt{D\beta} + \sqrt{L\tau / 4}\right)^2$ to broadcast in a $p \times q$ torus.

When *circuit-switched routing* is used, a header containing the destination address is sent through the network to "build" a path. At each intermediate node on the path, the input and output ports used by the header are connected. When the destination node receives the header, it sends an acknowledgment back to the source node establishing a direct connection between source and destination. The bytes of the message are then sent in pipeline fashion. Since the message is switched through intermediate nodes, there is no need to buffer the entire message. The links of the path can be released as the last byte passes through each node or by an acknowledgment from the destination node when the last byte is received. The former case is known as *direct connect* [23]. In a *wormhole* implementation of circuit-switching, the header establishes a path to the destination in the same way as in a direct connect implementation, but an acknowledgment is not sent back to the source node. Instead, the remaining bytes immediately follow the header in pipeline fashion with the last byte releasing the switches as it passes through. The Torus Routing Chip described in [9] uses wormhole routing for *point-to-point* (i.e., one-to-one) communications and can be used to build multidimensional tori [8].

In the linear cost model, when circuit-switched routing is used, the transmission time for a message of length L to be sent distance d is $\alpha + d\delta + L\tau$, where α is the time to initiate a new message transmission, δ is the time to switch an intermediate node, and $1/\tau$ is the bandwidth of the communication links. In most current machines, message transmissions are initiated in software and switching is done in hardware, so δ is usually much smaller than α. Furthermore, α is usually much larger than τ. For example, in the iPSC/860, the time to transmit $L \leq 100$ bytes over a distance d has been measured to be $(65 + 10d + 0.425L)$ μs [4], [5]. Store-and-forward routing can be simulated by circuit-switched routing by restricting d to be 1 for all transmissions, so the store-and-forward upper bounds stated above are also upper bounds for circuit-switched systems with $\beta = \alpha + \delta$.

There are other factors that affect the transmission time of a message. One factor is the propagation time for the header when setting up the path. Depending on the type of circuit-switching, there can also be propagation times for acknowledgments sent by the destination node at the beginning and at the end of a message transmission. A second factor is *router contention*. Since routers can "switch through" several paths by connecting pairs of ports, there can be contention when these paths are being set up. However, there is no router contention after the paths have been established, and no buffering of messages, so the propagation time of a message from source to destination is not affected by the number of nodes through which it is switched or the numbers of other messages that are being switched through those nodes. We will omit further mention of these factors since their effects are small and we can account for them by replacing the constants α, δ, and τ, and the parameter L, with slightly larger values. Seidel [27] discusses a longer list of factors that can affect real message passing systems, and concludes from experiments on the Intel Delta that most of them contribute very little to the overall cost of communication.

In the wormhole routing model, only the total switching time depends on the distance d, so it is possible that the last byte of a message has left the originator before the header reaches the destination. This could happen, for example, if the message is very short or d is large. This cannot happen in the direct connect model because the source and destination nodes are synchronized. The transmission times for wormhole routing and direct connect routing are similar when messages are long or the paths are short or both. When messages are short and the paths are long, the two routing methods behave quite differently. However, in this case, the dominant factor is α. The α factor is proportional to the number of phases, and this cannot be reduced by using wormhole routing instead of direct connect routing, so the transmission times will be similar. Since direct connect routing is easier to analyze than wormhole routing (because the source and destination nodes are synchronized in direct connect routing), we will adopt the direct connect model in this paper. We emphasize, however, that all of our algorithms can be used without modification with any type of circuit-switched routing, and that all of the bounds that we derive are valid for any type of circuit-switching.

Two currently available components that can be used to build torus networks with all of the properties described above are the iWarp cell [6], [7] and the T9000 Transputer [34]. The iWarp cell has four input and four output ports, each with a 40 Mbyte per second bandwidth. The bandwidth from memory to the communication controller is 160 Mbyte per second and the hardware supports multiplexing. The T9000 Transputer has a separate DMA controller for each input and each output channel, and four internal busses that provide multiport access to the on-chip cache.

When any type of circuit-switched routing is used, and communication patterns are arbitrary, deadlock is possible. In particular, if several headers are trying to establish routes and each has constructed a partial route containing links needed for the other routes, then none of the headers will reach its destination. Many papers on wormhole routing are more concerned with deadlock avoidance than with efficiency [10], [20], [18], [19]. The most common deadlock avoidance method is the use of *virtual channels* which use multiplexing to share physical links. Since our algorithm uses fixed, predetermined communication patterns, and the paths used during any phase are arc-disjoint, there is no possibility of deadlock. Since our algorithms are also synchronous, virtual channels offer no performance advantages that our algorithms can exploit, so we will ignore virtual channels.

The minimum phase wormhole broadcast algorithm in [2] shows that broadcasting can be done more efficiently in a processor-bounded system with circuit-switched routing than with store-and-forward routing. The tiling algorithm that we present in Section 4 is a minimum phase circuit-switched broadcast algorithm for link-bounded systems. A minimum phase circuit-switched broadcast algorithm for link-bounded systems is also presented in [25]. However, the model used in [25] is a unit cost model that ignores δ, L, and τ, and the algorithm is nonuniform in the sense that messages in the same phase can travel over different numbers of links. In contrast, the algorithm in Section 4 is uniform and simultaneously minimizes the α and δ terms.

3 Lower Bounds

The total transmission time for broadcasting in the linear cost circuit-switched model has three components: the *total start-up time*, the *total switching time*, and the *total propagation time*. These components are measured in terms of α, δ, and τ respectively. We can prove lower bounds on these components individually and in combination.

The total start-up time depends on the number of *phases* in a broadcast. A phase for a node that is sending a message starts when the node begins the initiation of the transmission and ends when the links over which the message was sent are released. For a node that is receiving a message, a phase starts when the header reaches the node and ends when the link on which the message arrived is released. Notice that the phases of the source and destination nodes of a message transmission do not start and end at exactly the same times. Furthermore, our definition of phase ignores the fact that a node can be sending and receiving different messages on each of its links and the starting and ending times for these transmissions can all be different. Thus, according to our definition, a node in a 2-dimensional torus could be in as many as eight different phases simultaneously.

To simplify our lower and upper bound analyses, we will assume that a node is in at most one phase at any given time. This assumption restricts the class of broadcast algorithms that we can analyze, but has no effect on the generality of our lower bounds. Since none of the algorithms described in this paper take advantage of the more general definition of phase, the assumption does not affect any of our upper bounds. Algorithms based on wormhole routing can take advantage of the more general definition of phase to reduce broadcast time, but the lower bounds in this section, and the upper bounds derived in later sections are all still valid for wormhole routing.

The movement of information in a broadcast is partially ordered because a node cannot send information before it has received it. Some of our lower bounds are based on determinations of minimum length critical paths in partial orders and others are based on analyses of bottlenecks in networks. None of our arguments rely on the definition of *phase*, so none of our lower bounds are affected by the simplifying assumption.

PROPOSITION 1. *In a vertex-transitive graph G with N nodes, degree Δ, edge connectivity λ, and diameter D, the minimum time to broadcast a message of length L is*

$$\max\left(\left\lceil \log_{\Delta+1} N \right\rceil \alpha, D\delta, \tfrac{L\tau}{\lambda}, \alpha + D\delta + \tfrac{L\tau}{\Delta}\right).$$

PROOF. First consider the total start-up time. When a source node initiates a message transmission, it incurs a cost of α and can transmit the message to at most Δ other nodes without incurring further start-up costs. Each of these Δ nodes can begin to forward the message to at most Δ more nodes as soon at it receives the header of the message from the source node, but each of these "second generation" transmissions incurs a start-up cost of α. After it has completed its first set of transmissions, the source node can also start to inform a "second generation" of Δ more nodes. These $\Delta + 1$ "second generation" transmissions can start at different times, but no "third generation" node can receive the header of the message before at least 2α time units have elapsed. Continuing in this way, it is easy to see that at most $(\Delta + 1)^k$ nodes can have the message after $k\alpha$ time units, and it follows that the total start-up time is at least $\lceil \log_{\Delta+1} N \rceil \alpha$.

The following arguments are simple extensions of the arguments developed in [16], [31] for store-and-forward routing in hypercubes.

The lower bound of $D\delta$ on total switching time is immediate since the message must travel along a path of length at least D from the source to at least one other node, and this path contains at least D switches.

If the edge connectivity of G is λ, then there is an edge cut of size λ which separates the source node from at least one other node. The bandwidth of this cut is $\frac{\lambda}{\tau}$, so the minimum propagation time for a message of length L through this edge cut is $\frac{L\tau}{\lambda}$.

The final term of the lower bound is the minimum time for the message to reach a node at distance D from the source. The header of the message cannot be received by this node in less than $\alpha + D\delta + \tau$ time units (assuming that the header has unit length). Since the node can receive information on all Δ of its edges simultaneously, it can receive Δ units of message by time $\alpha + D\delta + \tau$ and the remaining $L - \Delta$ units of message require at least $\frac{L-\Delta}{\Delta}\tau$ more time. □

COROLLARY 1. *In an $n \times n$ 2-dimensional torus, the minimum time to broadcast a message of length L is*

$$\max\left(2\left\lceil \log_5 n \right\rceil \alpha, \alpha + 2\left\lfloor \tfrac{n}{2} \right\rfloor \delta + \tfrac{L\tau}{4}\right)$$

When the messages are short or when α is much larger than δ and τ, the minimum time to broadcast in a 2-dimensional $p \times q$ torus approaches $\lceil \log_5 pq \rceil \alpha$.

4 AN OPTIMAL ALGORITHM FOR SHORT MESSAGES

The broadcast algorithm presented in this section can be viewed as a recursive tiling of a torus. We will take a direct approach to the development of our algorithm that avoids the sophisticated machinery that has been developed to study tilings. For an extensive treatment of tilings in general and tilings of the torus in particular, the reader is referred to [14], [29], respectively.

Fig. 1 shows two phases of a broadcast algorithm in a 2-dimensional mesh. We assume that the nodes have labels of the form (i, j). The originator of the broadcast is the black node in the center of the diagram, and we assume that it has label $(0, 0)$. In the first phase, the originator uses the knight's move paths (shown as heavy arrows) to broadcast the message to the four black nodes which have labels $(1, 2)$, $(-1, -2)$, $(2, -1)$, and $(-2, 1)$. Notice that the four paths are arc disjoint. In the second phase, each black node (x, y), including $(0, 0)$, sends the message to its four immediate neighbors $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$, and $(x, y - 1)$. The shading in Fig. 1 shows the "informed area" of the mesh after two phases. It is easy to see that this shaded area can be cut out and the links wrapped around from top to bottom and from left to right to form a 5×5 torus. The broadcast time is $2\alpha + 4\delta + 2L\tau$.

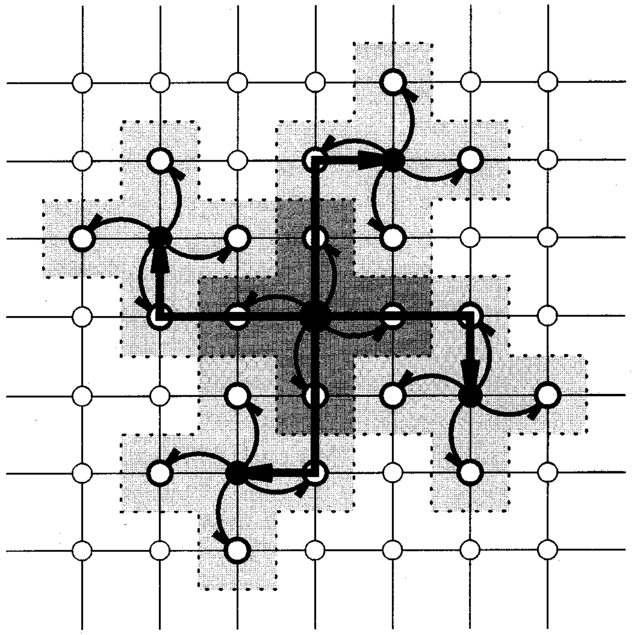


Fig. 1. Broadcasting in a mesh.

To reduce the complexity of the figures in this section, we will use a pictorial representation of broadcast algorithms. Fig. 2 corresponds to the shaded area of Fig. 1. The originator $(0, 0)$ is at the point in the center of the black cross. The four nodes with indices $(1, 2)$, $(-1, -2)$, $(2, -1)$, and $(-2, 1)$ that are informed by the originator during the first phase are at the centers of the small white crosses. The knight's move paths used in the first phase are shown as diagonal lines in Fig. 2 for clarity. The 20 nodes informed during the second phase are at the extreme points of the five small white crosses.

We can view the shaded area in Fig. 2 as a 5×5 "square" which we denote S_1. The four "corners" of S_1 are shown as black circles. The top "edge" of S_1 consists of nine line segments which have orientations "right-down-right-up-right-up-right-down-right" when proceeding from the top left corner to the top right corner. We will call this edge an E_1 edge with the clockwise orientation being understood.

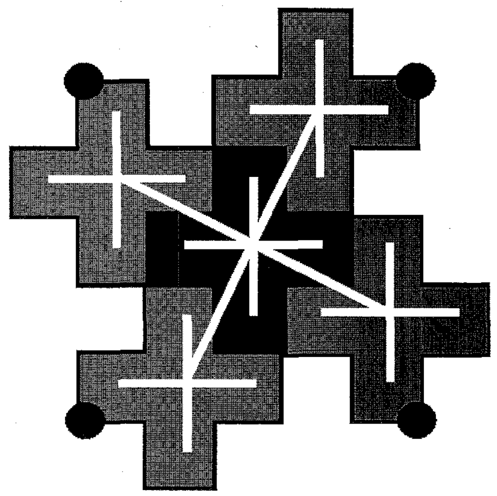


Fig. 2. A 5 × 5 torus drawn as an S_1 square.

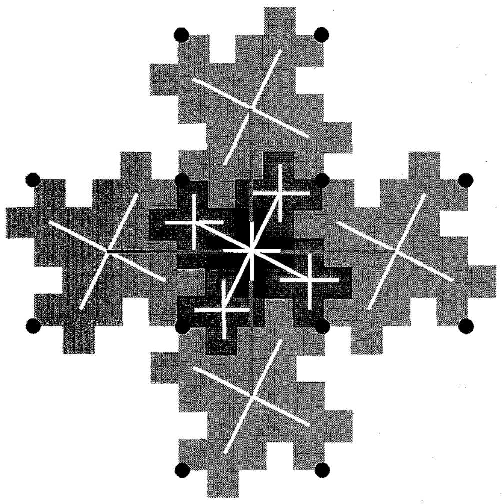


Fig. 3. A C_1 cross.

Clearly, the other three edges of S_1 are rotations of E_1. An E_1 edge has 180 degree rotational symmetry so the E_1 edges on the top and bottom of an S_1 fit perfectly into each other when wrapped around, as do E_1 edges on the left and right sides of an S_1. A 5 × 5 torus is obtained by identifying all four corners of an S_1 with a single point.

We can build bigger tilings by identifying the corners of several S_1 squares. Fig. 3 shows one possible tiling using five S_1 squares. Some of the details of the broadcast algorithm are omitted from the four outer S_1 squares to simplify the diagram. The tiling in Fig. 3 does not wrap around to form a torus, but we can view the shaded area as a "cross" in the same sense that S_1 is a square. Each corner of the cross is shown as a black circle and each "edge" of the perimeter of the cross is an E_1. We will denote this cross by C_1. We will use S_0 to denote a simple square that covers an area containing a single node and C_0 to denote a simple cross formed from five S_0s (such as the black cross in the middle of Fig. 2 or Fig. 3). A C_1 is formed by arranging five S_1s into the same pattern as the S_0s are arranged to form a C_0. Similarly, the symmetries of C_1s allow us to form the S_2 in Fig. 4 by arranging five C_1s into the same pattern as the C_0s are arranged to form an S_1 in Fig. 2.

We can view an S_2 as a square with corners indicated by the large black circles. Each of the four "edges" of S_2 consists of 9 E_1 edges (delimited by small black circles in Fig. 4) arranged in the same "right-down-right-up-right-up-right-down-right" pattern as nine simple edges are arranged in an E_1. It is, therefore, immediate that E_2 edges also have 180 degree rotational symmetry and that a 25 × 25 torus results when all four corners of an S_2 are identified with a single point. It is curious that the outlines of our S_k tilings, as drawn in Figs. 2 and 4, are Koch curves [21].

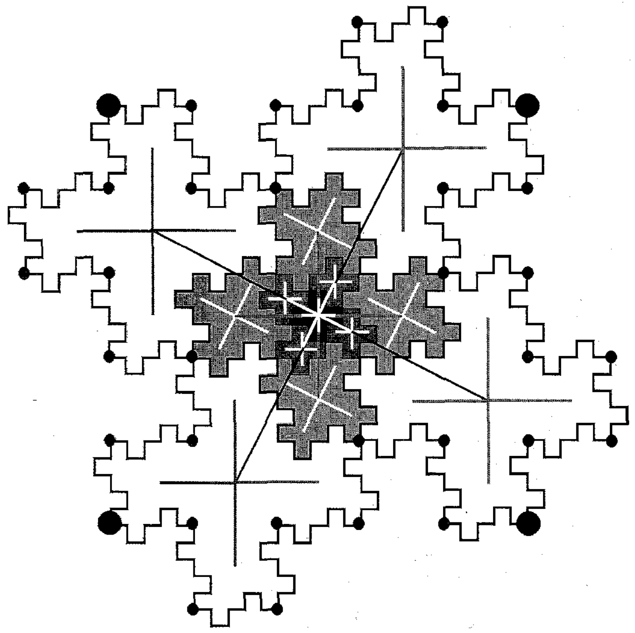


Fig. 4. A 25 × 25 torus drawn as an S_2 square.

THEOREM 2. *Broadcasting in a torus of size* $N = 5^k \times 5^k$, $k \geq 1$, *requires time at most* $(\log_5 N)\alpha + (\sqrt{N} - 1)\delta + (\log_5 N)L\tau$.

PROOF. Clearly, we can continue the construction described above to obtain a tiling for a torus of size $5^k \times 5^k$ for any $k \geq 1$. Furthermore, all of the data paths in use during any particular phase are arc disjoint. To calculate the time required by this broadcast algorithm, we need to determine the number of edges on each data path. To simplify notation in this proof, we will number the phases from last to first. During phase 1 (the last phase), the central node (x, y) of each C_0 broadcasts to its four immediate neighbors $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$, and $(x, y - 1)$. During phase 2, the central node of the central C_0 of each S_1 broadcasts to the central nodes of the four other C_0s in the same S_1. In particular, (x, y) broadcasts to $(x + 1, y + 2)$, $(x - 1, y - 2)$, $(x + 2, y - 1)$, and $(x - 2, y + 1)$ (see Fig. 2). In general, in an odd-numbered phase $2i + 1$, the center of the central S_i of each C_i is broadcasting to the centers of the four other S_is in the same C_i. Since S_is are $5^i \times 5^i$ "squares," the centers of two adjacent S_is in a C_i are distance 5^i apart (either horizontally or vertically). Similarly, in an even-numbered phase $2i$, the center (x, y) of the central C_i of each S_{i+1} broadcasts to the centers of the four other C_is in the same S_{i+1} and the data paths to these centers have $3 \cdot 5^i$ edges. In particular, during phase j, an informed node (x, y) will broadcast to the four nodes $(x + u_j, y + v_j)$, $(x - u_j, y - v_j)$, $(x + v_j, y - u_j)$, and $(x - v_j, y + u_j)$ where

$$u_j = \begin{cases} 5^{\frac{j}{2}-1} & \text{if even} \\ 0 & \text{if odd} \end{cases} \quad \text{and} \quad v_j = \begin{cases} 2 \times 5^{\frac{j}{2}-1} & \text{if even} \\ 5^{\frac{j-1}{2}} & \text{if odd.} \end{cases}$$

Since each informed node broadcasts to four other nodes during each phase, the number of phases in the broadcast is $log_5 (5^{2k}) = 2k$. The total time is therefore

$$\sum_{j=1}^{2k} \left[\alpha + (uj + vj)\delta + L\tau\right]$$

$$= 2k\alpha + \delta \sum_{i=1}^{k} \left[u_{2i-1} + u_{2i} + v_{2i-1} + v_{2i}\right] + 2kL\tau$$

$$= 2k\alpha + \delta \sum_{i=1}^{k} \left[4 \cdot 5^{i-1}\right] + 2kL\tau$$

$$= 2k\alpha + \left(5^k - 1\right)\delta + 2kL\tau.$$

□

Theorem 2 establishes that our algorithm is optimal in terms of both α and δ for tori for which both dimensions are the same even power of 5. We can use different tilings to obtain algorithms for other sizes of tori. These algorithms are optimal in terms of α and are usually optimal or close to optimal in terms of δ.

One way to create new tilings is to expand each node of a torus into a block of nodes. For example, Fig. 5 shows a C_0 cross in which each node has been expanded into a 2×2 block of nodes. The node in the lower left corner of each block is the "boss" of the block. We can make an expanded S_1 square by combining five of these expanded C_0s using the same pattern as in Fig. 2. To broadcast in this 10×10 torus, first perform a broadcast to the block bosses using the algorithm for an S_1 square. Since all the message paths are exactly twice as long as they were in the S_1 square, the total switching time for the two phases of the algorithm is doubled giving a time of $2\alpha + 8\delta + 2L\tau$. In the last phase, each boss broadcasts within its block. There are several ways to do this. The most efficient way is for the block boss (i, j) to route the message directly to $(i+ 1, j)$, indirectly to $(i + 1, j + 1)$ via $(i, j + 1)$, and indirectly to $(i, j + 1)$ via $(i - 1, j)$ and $(i - 1, j + 1)$. This adds $\alpha + 3\delta + L\tau$ to the cost. (The other ways to broadcast within a block either use an extra phase or use virtual channels.) We can improve on this result by changing the shape of the blocks for the last phase. The algorithm is the same during the first two phases, but during the last phase each representative (i, j) broadcasts directly to $(i - 1, j)$ and $(i, j + 1)$, and to $(i + 1, j + 1)$ via $(i + 1, j)$. It is clear that these "slanting" blocks give a valid tiling. The last phase using slanting blocks adds only $\alpha + 2\delta + L\tau$ to the cost for a total cost of $3\alpha + 10\delta + 3L\tau$. The 3α term is optimal, and so is the 10δ term since the diameter of a 10×10 torus is 10.

We can also form a 10×10 torus by combining four S_1 squares as shown in Fig. 6. In effect, we are replacing a single S_1 by a 2×2 block of S_1s. The bosses of the four S_1s are the nodes at the centers of the white crosses, and the originator is the boss of the lower left S_1. In the first phase, the originator broadcasts to the three other bosses. Each boss then starts a normal S_1 broadcast in its S_1. The first phase requires time $\alpha + 10\delta + L\tau$ and the last two phases require $2\alpha + 4\delta + 2L\tau$ for a total time of $3\alpha + 14\delta + 3L\tau$. Intuitively, the reason that this method is less efficient than the method in the preceding paragraph is because the inefficient phase in which informed nodes inform only three other nodes occurs between S_1s instead of S_0s. We can move the inefficient phase even further out in the recursion for larger tori, but this will always be worse than doing the block broadcast in the last phase.

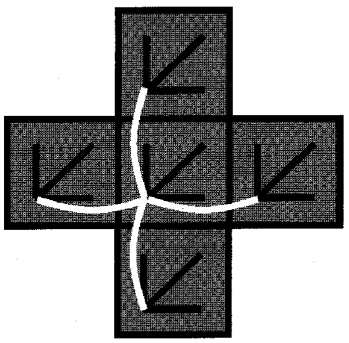


Fig. 5. An expanded c_0 cross with each node replaced by a 2×2 block.

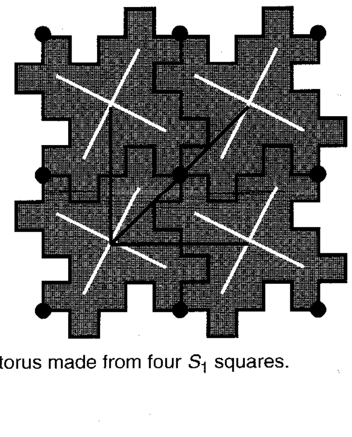


Fig. 6. A 10×10 torus made from four S_1 squares.

We can generalize our block boss technique by repeating the expansion of squares into 2×2 blocks of squares. For example, a second repetition on a 10×10 torus gives a 20×20 torus. The extreme case of this generalization is to start with an S_0 square (i.e., a single node) and expand the node into a 2×2 block, then expand each node of this block into a 2×2 block, and so on. This will give square tori with dimensions that are powers of 2. Since each phase of this broadcast algorithm is an "inefficient" phase in which each informed node informs only three other nodes, the number of phases is $\log_4 N$. This matches the performance of the divide-and-conquer algorithm that we will present in the next section, but the divide-and-conquer algorithm will be exponentially faster in terms of $L\tau$.

Our construction methods also work for tori that are not square. For example, a 5×10 torus can be formed by expanding each node of an S_1 square into a block of two horizontally adjacent nodes. In the first two phases, perform a broadcast to the left nodes of the blocks, and in the last phase each informed node broadcasts to its right neighbor. In the first two phases, the horizontal distances are twice as large as in an S_1 and the vertical distances are the same as in an S_1. The total cost is $3\alpha + 8\delta + 3L\tau$. The 3α term is optimal and the 8δ term is close to the lower bound of 7δ.

We can generalize our constructions to any size of torus by expanding nodes into blocks of the appropriate dimensions. We have not conducted a systematic analysis of this generalization, but based on the examples above and others that we have analyzed, it appears that the broadcast time will be closest to the lower bounds when the torus is close to square and when the dimensions are close to powers of 5. When the dimensions are close to powers of 2, it would be better to use the divide-and-conquer algorithm in the next section.

5 Efficient Algorithms for Long Messages

The tiling algorithm in the previous section is optimal with respect to α and δ, but it is far from the lower bound on the $L\tau$ term. For long messages, store-and-forward routing with pipelining and disjoint spanning trees can be done in $\frac{L\tau}{4} + o(L)$ time so the lower bound on $L\tau$ can be matched asymptotically with store-and-forward routing. However, the α term for any store-and-forward algorithm is exponentially larger than for the tiling algorithm. In this section we will combine the pipelining and disjoint spanning trees techniques with circuit-switched routing to obtain good performance simultaneously with respect to all three of α, δ, and $L\tau$.

The communication paths used by a broadcast algorithm form a directed spanning tree. For store-and-forward routing, the arcs of this *broadcast tree* are single communication links; for circuit-switched routing, they are paths. In both cases, the arcs of the broadcast tree should be *arc-disjoint* at any given time in the sense that no link is used by more than one broadcast tree arc at any given time. (Recall that links can be used simultaneously in both directions, so two arcs can share a link if their directions are different. Also note that we are not considering virtual channels or multiplexing of messages.)

In store-and-forward routing, a complete message must be received by a node before the node can begin to transmit it to another node. If the message is long, there can be a long delay between the receipt of the beginning of the message and the receipt of the end. Pipelining reduces this delay by partitioning the message into packets and then sending them one after the other along the same path. A node can begin to forward a message as soon as it has received the first packet instead of waiting for the entire message to arrive. Thus, the delay is reduced by overlapping the phases of a broadcast algorithm. This same technique can be used with circuit-switched routing as long as the paths used by overlapping phases are arc-disjoint. Unfortunately, the phases of the tiling algorithm of the previous section cannot be overlapped. In particular, the originator broadcasts to a new set of four nodes in each phase, and each set of four paths uses all four of the originator's outgoing ports. A similar problem occurs at all other nodes that are not leaves or parents of leaves in the broadcast tree. To take advantage of pipelining, we will use broadcast trees that have a maximum out-degree of four or less. This will increase the number of phases by a constant factor over the minimum-phase tiling algorithm, but the reduction in the $L\tau$ term will be proportional to $\log_5 N$.

The first algorithm of this section is based on the obvious divide-and-conquer approach of dividing the torus into four equal-sized parts, broadcasting to the centers of the parts, and then recursively broadcasting within the parts. For simplicity of exposition, we will only consider "square" tori in this section. However, the algorithms in this section can be easily extended to tori of other sizes.

Assume for the moment that messages are not partitioned into packets. The broadcast algorithm for a $2^k \times 2^k$ torus will have k phases. During phase i, $i = 1 \cdots k - 1$, each node (x, y) that received the message during phase $i - 1$ broadcasts it to nodes $(x + l, y + l)$, $(x - l, y + l)$, $(x + l, y - l)$, and $(x - l, y - l)$, with $l = 2^{k-i-1}$. During step k, if all nodes broadcast to their four immediate neighbors, some nodes will not be informed at the end of phase k while others will receive the message from two of their neighbors. At the expense of a small increase in path-length we can reroute redundant transmissions to otherwise uninformed nodes as suggested in Fig. 7. The center node in Fig. 7 could be informed by any of the four centers of "crosses". However, some of the paths from these cross centers to the center node may overlap paths from previous phases. This situation would prevent pipelining and must be avoided. Fortunately, it is always possible to find paths that are arc-disjoint from the paths of all previous phases. Fig. 8 shows the situation for an 8×8 torus. The solid circles denote nodes that are informed by the end of phase $k - 1$ and the open circles are uninformed nodes. It should be clear from the figure that all paths from the first $k - 1$ phases are arc-disjoint. Nodes a, b, c, and d are all possible informers for node e during phase k and each has two choices of path. The path that goes left from b and then down uses an arc that was used during the first phase to broadcast from the originator, so this path must be avoided. None of the seven other paths to e uses arcs that are used in any other phase. In general, care must be taken when informing nodes like e that are at the corner of a path, but there is always at least one usable path from each of the four possible informers of such a node.

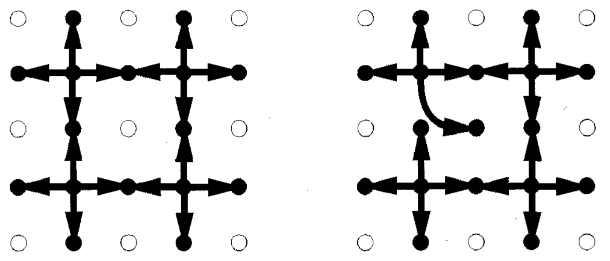


Fig. 7. Step *k* of broadcasting.

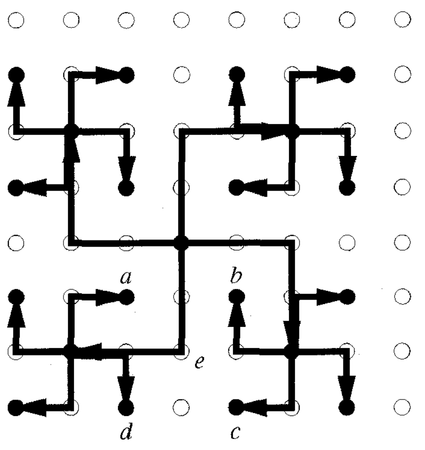


Fig. 8. Arc-disjoint broadcast paths for an 8 × 8 torus.

Since the broadcast tree described above is arc-disjoint, we can partition long messages into packets of some size *B* and use pipelining to overlap the phases and reduce the total cost. During the last phase, some of the paths have one arc and others have two. Otherwise, all paths in the same phase have the same length. The total number of arcs traversed by a packet from the originator to a node that is a leaf of the broadcast tree is therefore either $2^k - 1$ or 2^k. The total cost of a broadcast using packets of size *B* is:

$$T(B) = ka + 2^k\delta + kB\tau$$

(for the first packet)

$$+\left(\frac{L}{B} - 1\right)\left(\alpha + 2^{k-1}\delta + B\tau\right)$$

(for the remaining packets)

$$= \left(k + \tfrac{L}{B} - 1\right)(\alpha + B\tau) + 2^{k-1}\left(\tfrac{L}{B} + 1\right)\delta$$

The packet length that minimizes *T*(*B*) is

$$B_{\text{opt}} = \sqrt{\frac{\left(\alpha + 2^{k-1}\delta\right)L}{(k-1)\tau}}$$

and this gives a broadcast time of

$$T\left(B_{\text{opt}}\right) = \left(\sqrt{(k-1)\left(\alpha + 2^{k-1}\delta\right)} + \sqrt{L\tau}\right)^2 - (k-2)2^{k-1}\delta.$$

If messages are very short (or $L\tau$ is much smaller than α and δ) then pipelining does not improve the broadcast time. Thus, for short messages the broadcast time of this algorithm is $k\alpha + 2^k\delta = \left(\log_4 N\right)\alpha + \sqrt{N}\delta$. The α term is only a small constant factor ($\approx$ 1.16) larger than the $\log_5 N$ lower bound and the δ term essentially matches the lower bound of $\sqrt{N} - 1$. For long messages, the asymptotic broadcast time is $L\tau$. This is much better than the $(\log_5 N)L\tau$ term of the tiling algorithm but it does not match the lower bound of $\frac{L\tau}{4}$. To reduce the $L\tau$ term further, more than one arc-disjoint broadcast tree must be used. This is not possible with the divide-and-conquer algorithm just described because many of the nodes are using all four of their outgoing ports. Instead, we will use two arc-disjoint broadcast trees with maximum out-degree 2. The idea is to split the message into two parts and simultaneously broadcast the two parts using the two broadcast trees. This will halve the $L\tau$ term to $\frac{L\tau}{2}$ at the expense of doubling the α term. We will describe the algorithm for $(2^k - 1) \times (2^k - 1)$ tori. Extensions to tori of other sizes are not difficult.

Fig. 9 shows a broadcast in a 3 × 3 torus using two arc-disjoint broadcast trees. The first tree is shown with solid arcs. The second, shown with broken arcs is obtained from the first by a 90 degree rotation. These trees are called *H-trees* for obvious reasons. The recursive definition of H-trees is straightforward. To construct an H-tree for a $(2^k - 1) \times (2^k - 1)$ torus, arrange four copies of an H-tree for the $(2^{k-1} - 1) \times (2^{k-1} - 1)$ torus around a cross formed from a row and column of length $2^k - 1$ centered at the origin. Then connect the centers of the four H-trees with a new "H." Fig. 10 shows an H-tree for a 7 × 7 torus. H-trees are well known in VLSI design as a method to lay out binary trees with all

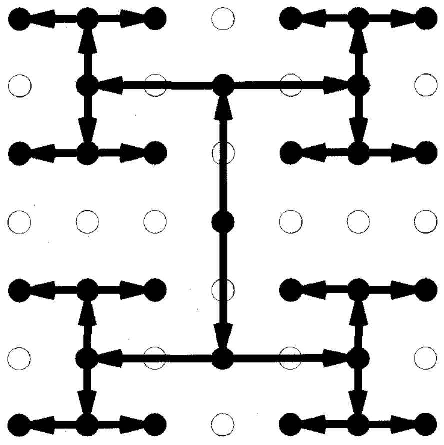


Fig. 10. H-tree for a 7 × 7 torus.

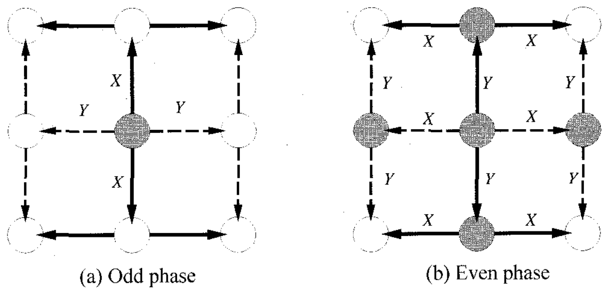


Fig. 9. H-trees for a 3 × 3 torus.

leaves equidistant from the root [1]. An H-tree tree with 2^{k-1} levels does not cover all of the nodes of the torus. However, every node that is not informed by the H-tree has two informed neighbors, so one more phase is sufficient to complete a broadcast.

Fig. 9 shows the two phases of a broadcast that uses two H-trees. The message is partitioned into two sub-messages X and Y. In the first phase (Fig. 9a), the originator sends X to its vertical neighbors and Y to its horizontal neighbors. In the second phase (Fig. 9b), the originator does the opposite and the four nodes informed in the first phase forward the sub-message they received to their two neighbors in the H-tree. This pattern of communication generalizes easily to a recursive definition of a broadcast algorithm using two H-trees. During each phase each node that received a submessage in the previous phase forwards it to its two neighbors in the H-tree. During odd phases, X sub-messages are traveling vertically and Y sub-messages are traveling horizontally. During even phases, the opposite happens. The total number of phases using H-trees for a $(2^k - 1) \times (2^k - 1)$ torus is $2k - 2$. After $2k - 2$ phases, every uninformed node has at least two informed neighbors, each of which has received both X and Y. During the last phase each uninformed node receives a submessage from two of its informed neighbors. One of the informed neighbors sends X and the other sends Y. Pipelining can be used with this algorithm by repeating the pattern for each packet. The originator first simultaneously sends the first packet of X vertically and the first packet of Y horizontally. The second step is to send the first packet of X horizontally and the first packet of Y vertically. In the third step, the originator sends the second packet of X vertically and the second packet of Y horizontally, and so on. The total time for this pipelining algorithm with packets of size B is:

$$T(B) = (2k-1)\alpha + \left(2^k - 1\right)\delta + (2k-1)B\tau$$

(for the first packets of X and Y)

$$+\left(\frac{L}{2B} - 1\right)\left(\alpha + 2^{k-2}\delta + B\tau\right)$$

(for the remaining packets)

$$= \left(2k + \tfrac{L}{2B} - 2\right)(\alpha + B\tau) + \left(2^{k-2}\left(\tfrac{L}{2B} + 3\right) - 1\right)\delta.$$

The packet length that minimizes $T(B)$ is

$$B_{\text{opt}} = \sqrt{\frac{\left(\alpha + 2^{k-2}\delta\right)L}{4(k-1)\tau}}$$

and this gives a broadcast time of

$$T\left(B_{\text{opt}}\right) = \left(\sqrt{2(k-1)\left(\alpha + 2^{k-2}\delta\right)} + \sqrt{L\tau / 2}\right)^2 - \left[(2k-5)2^{k-2} + 1\right]\delta.$$

For very short messages (when pipelining is not useful) the broadcast time is

$$(2k-1)\alpha + \left(2^k - 1\right)\delta = \left(2\log_4 N + o(1)\right)\alpha + \sqrt{N}\ \delta.$$

For long messages, the broadcast time is $\frac{L\tau}{2}$. So, by using two arc-disjoint broadcast trees and pipelining, we have reduced the $L\tau$ term to within a factor of 2 of the lower bound asymptotically while using a number of phases that is within a constant factor of the optimum.

6 Conclusion

Store-and-forward routing can be simulated by circuit-switched routing by restricting d to be 1 for all transmissions. In this case, the β for the simulated store-and-forward algorithm is $\beta = \alpha + \delta$. We will use this value for β in the following comparison of our new algorithms and store-and-forward algorithms. The table below summarizes the best upper bounds on the α, δ, and $L\tau$ terms for our three new algorithms and for store-and-forward broadcasting with pipelining and arc-disjoint broadcast trees. The table also contains the lower bounds on these three quantities. We have stated the α, δ, and $L\tau$ terms separately in the table below to facilitate comparisons even though some of the bounds derived in this paper involve tradeoffs among the three terms. Strictly speaking, we should be distinguishing between the asymptotic bounds on $L\tau$ for the pipelining algorithms and the exact bound for the tiling algorithm. Also, all logarithms have been converted to base 5 to facilitate comparisons, and ceilings have been removed. Finally, the bounds for some of the algorithms were only derived for tori of certain sizes, and we have not shown this in the table. Nevertheless, these minor inaccuracies do not affect the general conclusions that can be drawn from the table, and the results are stated precisely earlier in the paper.

TABLE 1
Bounds for Broadcasting

	Start-up Time	Switching Time	Propagation Time
Lower Bounds	$(\log_5 N)\alpha$	$\sqrt{N}\delta$	$L\tau/4$
Store-and-Forward	$\sqrt{N}\alpha$	$\sqrt{N}\delta$	$L\tau/4$
Circuit-Switched Tiling	$(\log_5 N)\alpha$	$\sqrt{N}\delta$	$(\log_5 N)L\tau$
Circuit-Switched D&C	$1.16\ (\log_5 N)\alpha = (\log_4 N)\alpha$	$\sqrt{N}\delta$	$L\tau$
Circuit-Switched H-trees	$2.32\ (\log_5 N)\alpha = (\log_2 N)\alpha$	$\sqrt{N}\delta$	$L\tau/2$

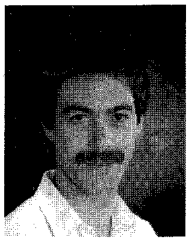
Since VLSI chips are wire-limited, 2-dimensional networks with low degree, such as the torus are good candidates for VLSI implementations [8]. With store-and-forward routing, the communication algorithms for these topologies require times that are at best proportional to the diameter of the network. The circuit-switched algorithms in this paper require exponentially less time when the messages are short and are more efficient than the best store-and-forward algorithm for all but extremely long messages. Our tiling algorithm is the first broadcast algorithm for tori to achieve the lower bound on the number phases. Our divide-and-conquer and H-trees algorithms are hybrids which combine circuit-switched routing with the pipelining and arc-disjoint spanning trees techniques from store-and-forward routing to achieve close to optimal performance in terms of phases, intermediate switch settings, and total transmission time. They are the first algorithms to achieve this performance in terms of all three parameters simultaneously.

Acknowledgments

We would like to thank Tom Shermer for suggesting the presentation of our tiling diagrams, and the referees for helpful comments.
The work of Joseph Peters was supported by CNRS—France and the Conseil Regional de Provence Alpes-Côte d'Azur. The work of Michel Syska was supported by INRIA—France.

References

[1] H.B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, pp. 367–370, 1990.
[2] M. Barnett, D.G. Payne, and R. van de Geijn, "Optimal Broadcasting in Mesh-connected Architectures," Tech. Report, Univ. of Texas at Austin, 1988.
[3] J.-C. Bermond and P. Fraigniaud, "Communications in Interconnection Networks," *Proc. Combinatorial Optimization in Science and Technology '91*, 1991.
[4] S. Bokhari, "Communication Overhead on the Intel *i*PSC-860 hypercube," Tech. Report, ICASE, NASA Langley Research Center, 1990.
[5] R. Boppana and C. Raghavendra, "All-to-all Personalized Communication on Circuit-switched Hypercubes," Tech. Rep., Dept EE-Systems, USC, Los Angeles, 1990.
[6] S. Borkar, et al., "iWarp: An Integrated Solution to High-speed Parallel Computing," *Proc. Supercomputing '88*, IEEE, pp. 330–339, 1988.
[7] S. Borkar et al., "Supporting Systolic and Memory Communication in iWarp," Tech. Report TR CMU-CS-90-197, School of Computer Science, Carnegie-Mellon Univ., 1990.
[8] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 775–785, 1990.
[9] W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *J. Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.
[10] W.J. Dally and C.L. Seitz, "Deadlock-free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, vol. 36, no. 5, pp. 547–553, 1987.
[11] J. Dongarra, R. van de Geijn, and R. Whaley, "Two Dimensional Basic Linear Algebra Communication Subprograms," *Proc. Sixth SIAM Conf. Parallel Processing*, pp. 347–352, 1993.
[12] P. Fraigniaud, "Communications Intensives dans les Architectures à Mémoire Distribuée et Algorithmes Parallèles pour la Recherche de Racines de Polynômes," PhD thesis, Ecole Normale Supérieure de Lyon, Université de Lyon I, 1990.
[13] P. Fraigniaud and E. Lazard, "Methods and Problems of Communication in Usual Networks," *Discrete Applied Math.*, vol. 53, pp. 79–133, 1994.
[14] B. Grünbaum and G.C. Shephard, *Tilings and Patterns.* W.H. Freeman, 1987.
[15] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, "A survey of Gossiping and Broadcasting in Communication Networks," *Networks*, vol. 18, pp. 319–349, 1986.
[16] S.L. Johnsson and C.T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Computers*, vol. 38,no. 9, pp. 1,249–1,268, 1989.
[17] P. Kermani and L. Kleinrock, "Virtual Cut-through: a New Computer Communication Switching Technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.
[18] X. Lin, P.K. McKinley, and L.M. Ni, "Performance Evaluation of Multicast Wormhole Routing in 2D-Mesh Multicomputers," *Proc. 1991 Int'l Conf. Parallel Processing*, pp. I-435–I-442, 1991.
[19] X. Lin and L.M. Ni, "Deadlock-free Multicast Wormhole Routing in Multicomputer Networks," *Proc. 18th Int'l Symp. Computer Architecture*, pp. 116–125, 1991.
[20] D.H. Linder and J.C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Trans. Computers*, vol. 40, no. 1, pp. 2–12, 1991.
[21] B.B. Mandelbrot, *The Fractal Geometry of Nature.* San Francisco: W.H. Freeman, 1982.
[22] P. Michallon, D. Trystram, and G. Villard, "Optimal Broadcast Algorithms on the Torus," Tech. Report 872-I-01-92 LMC-IMAG, Grenoble, France, 1992.
[23] S.F. Nugent, "The iPSC/2 Direct-connect Technology," *Proc. Third ACM Conf. Hypercube Concurrent Computers and Applications*, pp. 51–60, 1988.
[24] W. Oed, "The CRAY Research Massively Parallel Processor System, CRAY T3D," Cray Research, Munich, 1993.
[25] J-Y.L. Park, S-K. Lee, and H-A. Choi, "New Algorithms for Broadcasting in Meshes," Tech. Report GWU-IIST-93-03, George Washington Univ., 1993.
[26] Y. Saad and M.H. Schultz, "Data Communication in Parallel Architectures," *Parallel Computing*, vol. 11, pp. 131–150, 1989.
[27] S.R. Seidel, "Broadcasting on Linear Arrays and Meshes," Tech. Report ORNL/TM-12356, Oak Ridge National Laboratory, 1993.
[28] C.L. Seitz, "Concurrent Architectures," *VLSI and Parallel Computation*, R. Suaya and G. Birtwist, eds., Morgan Kaufmann, pp. 1–84, 1990.
[29] M. Senechal, "Tiling the torus," *Discr. and Comp. Geometry*, vol. 3, pp. 55–72, 1988.
[30] T. Shimizu, T. Horie, and H. Ishihata, "Low-Latency Message Comm. Support for the AP1000," *Proc. 19th Int'l. Symp. Computer Architecture, ACM*, pp. 288–297, 1988.
[31] Q.F. Stout and B. Wagar, "Intensive Hypercube Communication, Prearranged Communication in Link-bound Machines," *J. Parallel and Distributed Computing*, vol. 10, pp. 167–181, 1990.
[32] *nCUBE 6400 Processor Manual*, nCUBE Company, 1990.
[33] *Paragon XP/S Product Overview*, Intel Corporation, 1991.
[34] *The T9000 Transputer Products Overview Manual, first edition, INMOS*, 1991.



Joseph G. Peters received the BS degree in mathematics from the University of Waterloo, and the MSc and PhD degrees in computer science from the University of Toronto. He is an associate professor of computing science at Simon Fraser University. Dr. Peters' current research interests include network communications, algorithmic graph theory, and parallel computing.

Michel Syska received the PhD degree in computer science from the University of Nice—Sophia Antipolis, France, in 1992. He is a Maître de Conférences of Computing Science at the University of Nice—Sophia Antipolis, and is a member of the I3S laboratory. His current research interests include network communications, optical interconnections, and mapping problems in distributed systems. Dr. Syska is a member of RUMEUR.