Efficient Collective Data Distribution in All-Port Wormhole-Routed Hypercubes*

D. F. Robinson, D. Judd, P. K. McKinley, and B. H. C. Cheng

Department of Computer Science
Michigan State University
East Lansing, Michigan 48824-1027
{robinsod,danjudd,mckinley,chengb}@cps.msu.edu

Abstract

This paper addresses the problem of collective data distribution, specifically multicast, in wormhole-routed hypercubes. The system model allows a processor to send and receive data in all dimensions simultaneously. New theoretical results that characterize contention among messages in wormhole-routed hypercubes are developed and used to design new multicast routing algorithms. The algorithms are compared in terms of the number of steps required in each, their measured execution times when implemented on a relatively small-scale nCUBE-2, and their simulated execution times on larger hypercubes. The results indicate that significant performance improvement is possible when the multicast algorithm actively identifies and uses multiple ports in parallel.

1 Introduction

The recent trend in supercomputer design has been towards *scalable* parallel computers, which are designed to offer corresponding gains in performance as the number of processors is increased. Many such systems, known as *massively parallel computers* (MPCs), are characterized by the distribution of memory among an ensemble of processing nodes. Each node has its own processor, local memory, and other supporting devices.

In parallel scientific computing, data must be redistributed periodically in such a way that all processors can be kept busy performing useful tasks. Because they do not physically share memory, nodes in MPCs must communicate by passing messages through a communications network. Some communication operations are *point-to-point*, that is, they involve only a single source and a single destination. Other operations are *collective*, in that they involve more than two nodes. Examples of collective communication include *multicast*, *reduction*, and *barrier synchronization*. The growing interest in the use of such routines is evidenced by their inclusion in many commercial communication libraries and in the *Message Passing Interface* (MPI) [1], an emerging standard for communication routines used by message-passing programs. Besides message-passing, collective communication operations are also important in implementing data-parallel languages, such as High Performance Fortran [2].

Communication performance depends on several characteristics of the network architecture, including the *network topology*, *routing algorithm*, and *switching strategy*. Network topologies of commercial and research MPCs vary widely and include the 2D mesh (Intel Paragon), 3D mesh (MIT J-machine), hypercube (nCUBE-2), 3D torus (Cray MPP), and the Fat Tree (Thinking Machines CM-5). Most routing algorithms used in commercial machines are deterministic, that is, the path between a particular source and destination is fixed. The predominant switching technique is *wormhole routing* [3], in which a message is divided into a number of *flits* that are pipelined through the network. The two salient features of wormhole routing are: (1) only a small amount of buffer space is required in each router, and (2) the network latency is almost distance insensitive if there is no channel contention among messages [4].

In wormhole-routed MPCs, communication among nodes is handled by a separate *router*. As shown in Figure 1, several pairs of *external channels* connect the router to neighboring routers; the pattern in which the external channels are connected defines the network topology. Usually, the router can relay multiple messages simultaneously, provided that each incoming message requires a unique outgoing channel. In addition, two messages may be transmitted simultaneously in opposite directions between neighboring routers.

A router is connected to the local proces-

*This work was supported in part by the NSF grants CDA-9121641, MIP-9204066, CDA-9222901, and CCR-9209873, by the Department of Energy under grant DE-FG02-93ER25167, and by an Ameritech Faculty Fellowship.

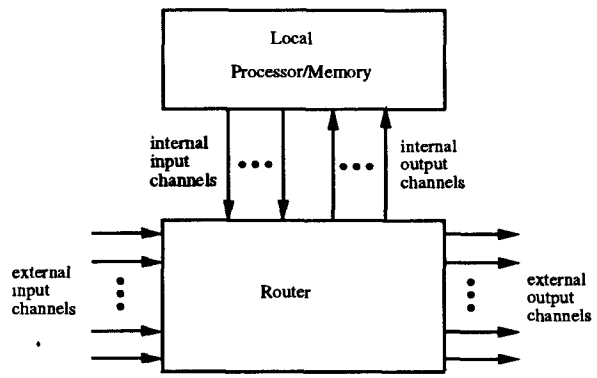

Figure 1. Generic MPC node architecture

sor/memory by one or more pairs of *internal* channels. One channel of each pair is for input, the other for output. The *port model* of a system refers to the number of internal channels at each node. If each node possesses exactly one pair of internal channels, then the result is a so-called "one-port communication architecture" [5]. A major consequence of a one-port architecture is that the local processor must transmit (receive) messages sequentially. Architectures with multiple ports reduce this bottleneck. In the case of an *all*-port system, every external channel has a corresponding internal channel, allowing the node to send to and receive on all external channels simultaneously.

In this paper, the specific problem of efficient multicast communication for all-port wormhole-routed hypercubes is addressed. Formally, a *hypercube* (or n-cube) consists of 2^n nodes, each of which has a unique n-bit binary address. For each node v, let v also denote its n-bit binary address, and let $\| v \|$ represent the number of 1's in v. A channel $c = (u, v)$ is present in an n-cube if and only if $\| u \oplus v \| = 1$, where $\oplus$ is the bitwise exclusive-or operation on binary numbers. The hypercube topology has been used in multicomputer design for many years [6]. The nCUBE-2 [7] is the first hypercube to support wormhole-routing, as does the recently announced nCUBE-3 [8].

The remainder of the paper is organized as follows. Section 2 illustrates the issues and problems involved in supporting efficient multicast communication in all-port hypercube systems. Section 3 gives new theoretical results that provide the foundation for this work. Section 4 presents the new algorithms that have been produced to support multicast in all-port wormhole-routed hypercubes. Section 5 compares the algorithms using analysis, simulation, and implementations on a 64-node nCUBE-2. Finally, conclusions are presented in Section 6.

2 Background and motivations

Collective communication operations may be implemented in either hardware or software. However, most existing wormhole-routed multiprocessors support only point-to-point, or *unicast* communication in hardware. In these environments, all communication operations must be implemented in software by sending one or more unicast messages; such implementations are called *unicast-based*. A multicast operation may be implemented by sending a separate copy of the message from the source to every destination. An alternative is to use a *multicast tree* of unicast messages. In a multicast tree, the source node actually sends the message to only a subset of the destinations. Each recipient of the message forwards it to some subset of the destinations that have not yet received it. The process continues until all destinations have received the message. Using this approach, the time required for the operation can be greatly reduced.

Although implemented in software, unicast-based collective communication algorithms must exploit the underlying architecture in order to minimize their execution time. In a wormhole-routed system, the implementation should not only exploit distance-insensitive unicast latency [4], but must also avoid channel contention, that is, no two messages involved in the operation should simultaneously require the same channel. Avoiding channel contention depends on the underlying unicast routing algorithm of the MPC; hypercubes often adopt *E-cube routing*, in which messages are routed through dimensions in either ascending or descending order. Also, the implementation should involve no local processors other than those affected by the operation. For example, in a multicast, only source and destination processors should be required to handle the message. Finally, the implementation should account for the port model, which affects how fast nodes can send and receive messages.

The following (small-scale) example illustrates the issues and difficulties involved in implementing efficient multicast communication in hypercubes.

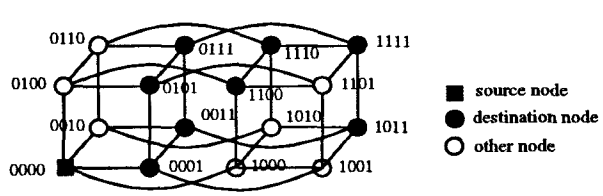

Figure 2. An example of multicast in a 4-cube

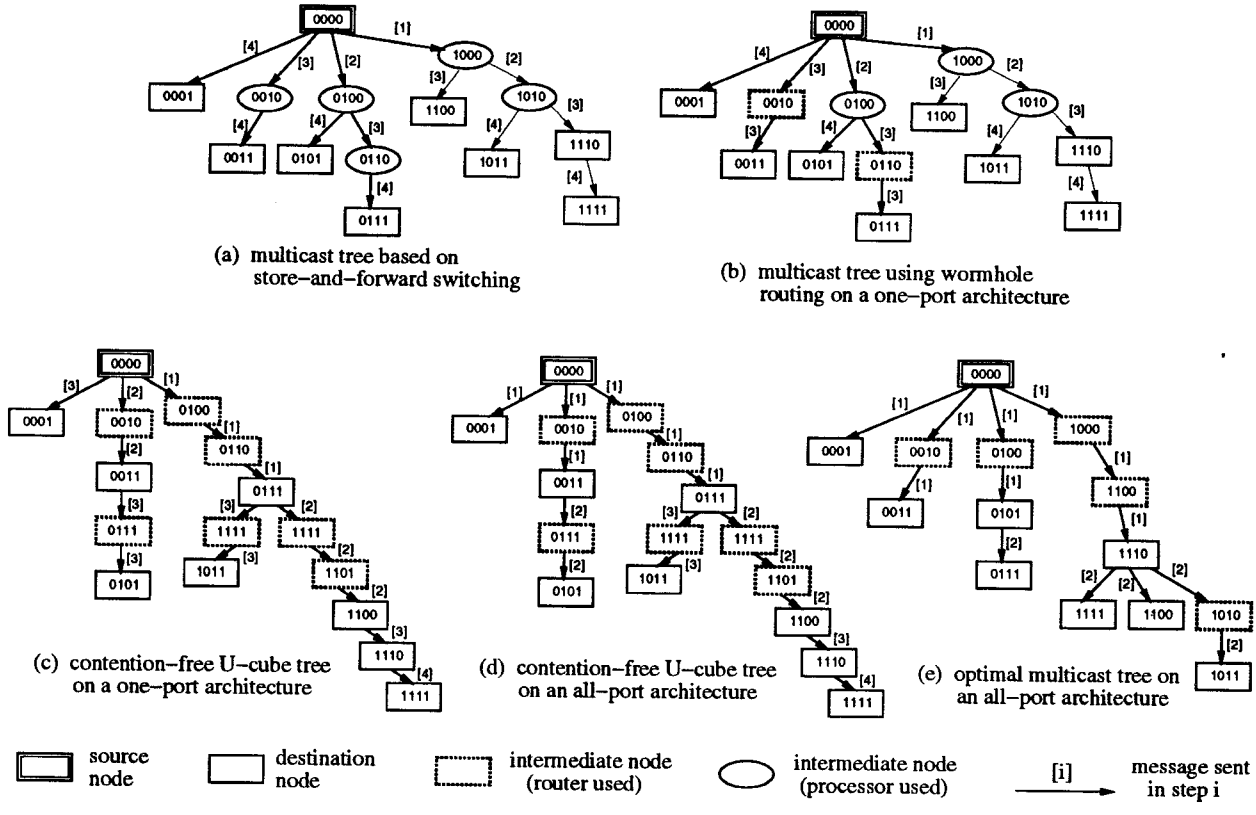

Figure 3. Unicast-based software multicast trees

Consider the 4-cube in Figure 2, and suppose that a multicast message is to be sent from node 0000 to eight destinations {0001, 0011, 0101, 0111, 1011, 1100, 1110, 1111}. (In this example and all subsequent examples, we assume that the E-cube routing algorithm resolves addresses from high order bits to low order bits. In the nCUBE-2, the opposite resolution strategy is used, but this difference does not affect any of the results presented.)

In early hypercube systems that used store-and-forward switching, the procedure shown in Figure 3(a) could be used. At step 1, the source sends the message to node 1000. At step 2, nodes 0000 and 1000 inform nodes 0100 and 1010, respectively. Continuing in this fashion, this implementation requires 4 steps to reach all destinations. In this example, five nodes (0010, 0100, 0110, 1000, and 1010) are required to relay the message, even though they are not destinations. Using the same routing algorithm in a one-port wormhole-routed network also requires 4 steps, as shown in Figure 3(b). In this case, however, only the *routers* at two of the non-destination nodes (0010 and 0110) are involved in forwarding the message. For example, the message may be passed from 0000 to 0011 in one step; the message is relayed at node 0010 by the router, not the local processor. However, because the message must be replicated at nodes 0100, 1000, and 1010, the local processors at those nodes must still handle the message.

Figure 3(c) illustrates the result of using the *U-cube algorithm* [9] to solve the problem on a one-port wormhole-routed system. (U-cube, which was designed specifically for one-port wormhole-routed architectures, will be discussed further in Section 4.) The only local processors required to handle the message are those at destination nodes. Furthermore, on a one-port architecture, all messages are guaranteed to be contention-free [9]. Although common channels are used between the 0111-to-1011 path and the 0111-to-1100 path, these messages are sent sequentially, so contention does not arise. Details of avoiding contention in one-port architectures can be found in [9].

Although the algorithm used to construct the tree in Figure 3(c) still requires four steps, this approach is optimal for multicasting to 8 destinations on a one-port architecture. Specifically, $\lceil \log_2(m+1) \rceil$ is a tight lower bound on the number of steps required

to reach m destinations on a one-port hypercube. Since the U-cube algorithm was designed specifically for one-port systems, however, it makes no attempt to take advantage of multiple ports. For example, if the algorithm were implemented on an all-port hypercube, it would still require four steps to complete the multicast in the above example, as illustrated in Figure 3(d), although some destinations are reached earlier than in Figure 3(c). Notice that three steps are required to reach destination node 1011, since that unicast message must traverse a channel in the path required to reach node 1100, thereby delaying its transmission.

Figure 3(e) shows a multicast tree that accounts for both wormhole routing and an all-port architecture. The algorithm requires only two steps, no local processors other than the source and destinations are involved, and contention among constituent messages is avoided. This particular tree, which is based on the methods presented in this paper, is optimal for multicast to the given set of nodes on an all-port architecture. In the next section, we develop the theoretical results necessary to guarantee that our new algorithms, presented in Section 4, are contention-free.

3 Theoretical foundations

In this section, we present new theoretical results that will serve as a basis for subsequent algorithms. First, we formally define terms related to dimension-ordered routing and subcubes. Then we present several theorems that are useful in determining that certain pairs of paths are guaranteed to be arc-disjoint (and hence, contention-free). Finally, we formally define contention in an all-port hypercube architecture, and present a related theorem.

3.1 Notation and definitions

Bitwise *and*, *or*, and *exclusive-or* are represented by symbols $\otimes$, $|$, and $\oplus$, respectively. Logical *and* and *or* are represented by $\wedge$ and $\vee$, respectively. We use N to represent the number of processors. Since n represents the dimensionality of the hypercube, $N = 2^n$. For each node, say x, the outgoing (and incoming) channels of node x are labeled 0 through $n-1$, where channel d connects node x to node $x \oplus 2^d$. We say that channel d is used to *travel* in dimension d.

The path from a source node u to a destination node v resulting from dimension-ordered routing is denoted $P(u,v)$. $P(u,v) = (u; w_1; w_2; \ldots; w_p; v)$ is the sequence of nodes visited on the path. Note that $p+1 = \|u \oplus v\|$. Under E-cube routing, a unique shortest path is always taken. In particular, a message will always travel over the required dimensions in *descending* (alternately, *ascending*) order. For example, the path from source node 0101 to destination node 1110 is $P(0101, 1110) = (0101; 1101; 1111; 1110)$. A unicast from node u to node v occurring at time step t is denoted $(u, v, P(u,v), t)$.

The following definition simplifies references to the initial channel in a dimension-ordered route, that is, the first dimension in which a message will travel.

Definition 1 *$\delta(u,v)$ represents the highest-ordered bit position in which u and v differ. Formally, $\delta(u,v) = \lfloor log_2(u \oplus v) \rfloor$. If $u = v$, then $\delta(u,v)$ is undefined.*

In order to specify a subset of the nodes of a hypercube, we may explicitly state some of the n address bits, and allow the other address bits to range over all possible values. We define a *subcube* to be such a subset of nodes, where the explicitly-stated address bits are the high order bits, and the free-ranging address bits are the low order bits.

Definition 2 *A* subcube $S = (n_S, M_S)$ *is defined by a* dimensionality $n_S \in \{0 \,..\, n\}$, *and a* mask $M_S \in \{0 \,..\, 2^{(n-n_S)} - 1\}$. *Informally, subcube S consists of those nodes whose highest-ordered $(n - n_S)$ bits have a value equal to M_S. Formally, for any node u, $u \in S$* iff $(u \gg n_S) = M_S$, *where "$u \gg i$" indicates a right-shift of u by i bits.*

3.2 Basic results

This section contains lemmas that will be used to facilitate subsequent theorems. These lemmas are also useful in understanding later sections of the paper. Due to space limitations, proofs of all lemmas and theorems are omitted here, but can be found in [10].

Lemma 1 *Let $P(x,y) = (x; w_1; w_2; \ldots; w_p; y)$ be any E-cube path (For clarity, let $w_0 = x$ and $w_{p+1} = y$.), and let $(w_i \rightarrow w_{i+1}) \in P$ be any arc in $P(x,y)$. Let d be the dimension over which $(w_i \rightarrow w_{i+1})$ travels, $2^d = w_i \oplus w_{i+1}$. Then the following conditions hold:*

1. $\forall\, j \in \{1..i\},\ \forall\, k \in \{0..d\},\ w_j \otimes 2^k = x \otimes 2^k$
2. $\forall\, j \in \{i+1..p\},\ \forall\, k \in \{d+1..n-1\},$ $w_j \otimes 2^k = y \otimes 2^k$
3. $x \otimes 2^d \neq y \otimes 2^d$.

Lemma 1 characterizes the behavior of dimension-ordered routing in a hypercube. In particular, the lemma formalizes the notion that any unicast travels in a particular dimension at most once, and that the unicast travels in a strictly decreasing order of dimension.

Lemma 2 *For any three nodes x, y, z, and for any subcube S, if $x, z \in S$ and $x \leq y \leq z$, then $y \in S$.*

Lemma 2 states that the node addresses within any subcube are contiguous.

3.3 Arc-disjoint paths

In implementing a multicast algorithm, whenever the paths of two constituent unicast messages share an arc (channel), care must be taken to ensure that the paths do not attempt to use the shared arc simultaneously. Alternatively, when two paths have no arc in common, contention (between these two particular paths) is always avoided. Paths with no common arc are said to be *arc-disjoint.*

Each of the following theorems state sufficient conditions on two paths such that any two paths meeting these conditions are arc-disjoint. Each theorem is stated formally. Where needed for clarity, theorems are stated informally within a parenthetical block of text.

Theorem 1 *Consider any two paths $P(x, y)$ and $P(x, v)$ originating from a common source node. If $\delta(x, y) \neq \delta(x, v)$, then $P(x, y)$ and $P(x, v)$ are arc-disjoint. (Paths leaving a common source on different channels are arc-disjoint.)*

Theorem 2 *Consider any two paths $P(u, v)$ and $P(x, y)$. If there exists a subcube S such that $u, v \in S \wedge x, y \notin S$, then $P(u, v)$ and $P(x, y)$ are arc-disjoint. (A path with source and destination within subcube S is arc-disjoint from any path with source and destination outside S.)*

3.4 Contention

As previously stated, any two unicasts having arc-disjoint paths are contention-free. Unicasts whose paths share an arc may or may not contend for that arc, depending on the relative timing and communication latencies of the unicasts. Due to the nondeterministic characteristics of communication latency, we must establish necessary conditions for avoiding contention, such that when these conditions are met, it is *guaranteed* that contention will not occur.

In order to study contention between messages, the definition of the *reachable set* of nodes in a multicast implementation is needed [9].

Definition 3 *A node v is in the reachable set of node u, denoted R_u, if and only if one of the following holds:*

1. $v = u$; or
2. There exists a unicast $(w, v, P(w, v), t)$ such that $w \in R_u$.

The reachable set of a node u contains those nodes in the multicast that receive the message, either directly or indirectly, through node u. If the multicast is viewed as a tree of unicast messages, then R_u is the set of nodes in the subtree rooted at u. Using this definition, the characteristics of a multicast necessary to avoid contention between messages sent in different time steps, called *depth contention*, can be formally defined.

Definition 4 *A multicast implementation is contention-free if and only if its constituent unicasts are pairwise contention-free. For any two unicasts $(u, v, P(u, v), t)$ and $(x, y, P(x, y), \tau)$ where $t \leq \tau$, the two unicasts are contention-free if and only if either:*

1. $P(u, v)$ and $P(x, y)$ are arc-disjoint; or
2. $(t < \tau) \wedge (x \in R_u)$.

When viewing a multicast as a unicast tree, the second item in the above definition allows two unicasts to share an arc under the condition that either (1) one unicast is an ancestor of the other (this case occurs when $x \in R_v$), or (2) one unicast is an ancestor of a later sibling of the other. The latter case occurs when $(t < \tau) \wedge (x \in R_u)$, but $x \notin R_v$. From the above definition of contention-free unicast pairs, we can conclude that no two unicasts from a common source node will experience contention, as shown in the following theorem.

Theorem 3 *Any two unicasts $(u, v, P(u, v), t)$ and $(u, y, P(u, y), \tau)$ with a common source node are contention-free.*

4 Algorithms

In this section, we define several algorithms whose performance was compared on all-port hypercubes. The performance evaluation data are given in Section 5.

4.1 Algorithms based on dimension-ordered chains

We begin with a brief review of the U-cube algorithm [9]. This algorithm, designed for one-port architectures, produces multicast trees on such systems that are of minimum height and are guaranteed to be contention-free. The U-cube multicast algorithm relies on the binary relation "dimension order," denoted $<_d$, which is defined between two nodes a and b as follows: $a <_d b$ if and only if either $a = b$ or there exists a j such that $a \otimes 2^j < b \otimes 2^j$ and $a \otimes 2^i = b \otimes 2^i$ for all i, $j + 1 \leq i \leq n - 1$. A sequence $\{d_0, d_1, d_2, \ldots d_p\}$ of source and destination

addresses in which all the elements are distinct and $d_i <_d d_j$ for all $0 \leq i < j \leq p$ is called a *dimension-ordered chain* [9]. A sequence $\{d_1, d_2, \ldots, d_m\}$ is called a *d_0-relative dimension-ordered chain* if and only if $\{d_0 \oplus d_1, d_0 \oplus d_2, \ldots, d_0 \oplus d_m\}$ is a dimension-ordered chain.

If address resolution is performed from highest (left) to lowest (right), then dimension order is the same as the usual increasing order. For example, dimension ordering of 10100, 00110, and 10010 results in the chain: 00110, 10010, 10100.

On the other hand, if addresses are resolved from lowest to highest, then a dimension-ordered chain is: 10100, 10010, 00110.

Figure 4 gives the U-cube algorithm. The source d_0 and the destination addresses are sorted into a d_0-relative dimension-ordered chain, denoted Φ, at the time when multicast is initiated. The source node successively divides Φ in half and sends a message to the first node in the upper half of the chain. That node is responsible for delivering the message to the other nodes in the upper half, using the same U-cube algorithm. At each step, the source deletes from Φ the receiving node and those nodes in the upper half. The source continues this procedure until Φ contains only its own address.

Algorithm 1: The U-Cube Multicast Algorithm [9]
Input: Dimension ordered address sequence $\{d_{left}, d_{left+1}, \ldots, d_{right}\}$, where d_{left} is the local address, and a message M.
Output: Send out one or more copies of message M
Procedure:
repeat
1. $x = \delta(d_{left}, d_{right})$, the position of the first bit difference
2. let $d_{highdim}$ be the leftmost destination in the chain such that $\delta(d_{left}, d_{highdim}) = x$
3. $center = left + \lceil \frac{right - left}{2} \rceil$
4. $next = center$
5. $D = \{d_{next+1}, d_{next+2}, \ldots, d_{right}\}$;
6. Send a copy of message M to node d_{next} with the address field D
7. $right = next - 1$

until ($left = right$)

Figure 4. The *U-cube* multicast algorithm

Figure 5 gives an example of this method in a one-port 4-cube. The source node 0100 is sending to a set of eight destinations {0001, 0011, 0101, 0111, 1000, 1010, 1011, 1111}. Taking the exclusive-or of each destination address with 0100 and sorting the results produces the d_0-relative dimension-ordered chain Φ = {0000, 0001, 0011, 0101, 0111, 1011, 1100, 1110, 1111}. (The reader will notice that this d_0-relative chain represents the same multicast operation examined in Figure 3.) The corresponding U-cube tree is shown in Figure 5. It takes 4 steps for all destination processors to receive the message.

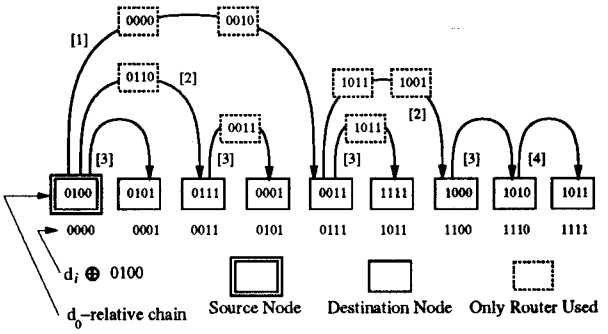

Figure 5. Multicast chain in a one-port 4-cube

It has been previously shown that message transmission in U-cube tree is contention-free regardless of startup latency and message length [9]. Furthermore, the U-cube algorithm achieves minimum time for a one-port architecture by requiring only $p = \lceil log_2(m+1) \rceil$ time steps for m destinations. For details of the theory behind the U-cube and the accompanying U-mesh algorithm, please refer to [9].

The *U-cube* algorithm, however, makes no attempt to parallelize message transmissions from a given sender. When executed on an all-port hypercube, the algorithm will often fail to take advantage of that architectural property. In the tree shown in Figure 3(d), for example, step 1 of the algorithm "mistakenly" selects node 0111 as the first destination to which the message is transmitted. This decision leaves node 0111 responsible for delivering the message to four nodes, all of which differ from 0111 in the highest dimension. Better message-forwarding decisions, shown in Figure 3(e), result in a tree of height two.

This observation leads to two simple variations on the U-cube algorithm. In fact, both algorithms differ from U-cube in a single statement, which determines the degree to which they exploit the all-port capability of the system. In the *Maxport* algorithm, a sender transmits (in parallel) to the maximum number of destinations permitted by the architecture and the specific destination set. Step 4 in the body of the main loop is $next = highdim$, rather than $next = center$, as in U-cube (Figure 4). This choice can sometimes lead to performance worse than U-cube, however. For example, if node 0000 is the source of a multicast to nodes 1001, 1010, and 1011, the resulting Maxport

"tree" will require three steps, as shown in Figure 6(a). The U-cube solution shown in Figure 6(b) requires only two steps.

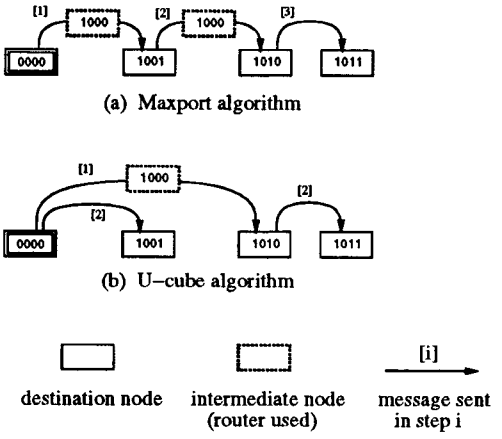

Figure 6. Simple Maxport and U-cube comparison

Just as U-cube does not account for dimension, neither does Maxport account for the number of destinations for which each node is responsible. A simple modification to the algorithm solves the problem. As the name implies, the *Combine* algorithm exhibits characteristics of both the *U-Cube* and *Maxport* algorithms. This algorithm attempts to use multiple ports, but not at the expense of leaving a single node responsible for a large subset of the destinations. In order to obtain the *Combine* algorithm, step 4 in the body of the main loop is changed to $next = \max(highdim, center)$. The performance of all three algorithms is compared in Section 5.

4.2 An algorithm based on cube-ordered chains

In this section, we present an alternative approach to multicasting, in which the source node and destination nodes are considered as elements of subcubes.

Definition 5 *A chain* $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ *is a* cube-ordered chain of dimension n *iff*

1. for all $d \in D$, $0 \leq d < 2^n$; and
2. for all subcubes $S = (n_S, M_S)$, where $n_S \leq n$ for all i, j, k where $first \leq i \leq j \leq k \leq last$, if $d_i, d_k \in S$, then $d_j \in S$.

(A chain D is cube-ordered iff the nodes of D within any subcube are contiguous.)

Theorem 4 *If* $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ *is a dimension ordered chain, then D is also a cube-ordered chain.*

In the Maxport algorithm, for each participating node, say x, the unicasts originating at node x are transmitted on different outgoing channels. In this approach, the message is always forwarded to different subcubes; when node x receives the message over channel d, it also receives a list of destination nodes, D, which are in the same d-dimension subcube as x, say subcube S. In turn, x issues one unicast into each subcube *within* S which (1) does not contain x, (2) is maximal, and (3) contains at least one destination node. As will be shown later, it is possible to input any cube-ordered chain, not just a dimension-ordered chain, to Maxport and still be able to avoid contention among messages. An ordinary dimension-ordered chain may not be the most appropriate cube-ordered chain to use. In fact, performance increase may be gained by exchanging subcubes of the chain, where possible, so that source nodes (including intermediate source nodes in the multicast tree) always choose the most "crowded" destination node among available destination nodes.

Figure 7 shows the *weighted_sort* algorithm, which permutes a cube-ordered chain so that the most "crowded" node appears as the first node of each subcube. This is accomplished by exchanging subcube halves (these halves are themselves subcubes) so that the most populated half occurs first in the chain. Notice that the *cube_center* function is applied to a cube-ordered chain of addresses that are contained within a subcube of dimension n_S. This function returns the starting position of the second $(n_S - 1)$ dimension subcube "half" of the input subcube. (If one of the $(n_S - 1)$ dimension subcubes contains no destination nodes, then cube_center returns a value of $last + 1$.)

The weighted_sort algorithm, as shown, is a centralized algorithm with computational complexity $O(m^2)$, where m is the number of destination nodes. We have also developed a distributed version that has complexity $O(m\ log_2 m)$. For details, please refer to [10]. To use the weighted_sort algorithm with Maxport, the list of destinations is first sorted according to dimension-order, then sorted using the weighted sort algorithm, and finally input to Maxport. Let us call the combination of these techniques the *W-sort* routing algorithm.

Figure 8 illustrates the behavior of the W-sort algorithm in a 4-cube. As shown in Figure 8(a), the set of destination nodes is $D = \{0, 1, 3, 5, 7, 11, 12, 14, 15\}$. (Their binary equivalents are given for reference.) Since the nodes of D are in ascending order, D is a cube-ordered address sequence, by Theorem 4.

Procedure: **weighted_sort** $(D, first, last, n_S)$
Input: Cube-ordered chain $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ and a subcube size n_S.
Output: Upon exit, D is a *weighted* cube-ordered chain.
Procedure:

```
if last − first ≥ 2 then
    center = cube_center(D, first, last, n_S)
    weighted_sort(D, first, center − 1, n_S − 1)
    weighted_sort(D, center, last, n_S − 1)
    if (first ≠ 0) ∧
       ((center − first) < (last − center + 1)) then
       /* swap subcubes */
       D = {d_center, d_center+1, ..., d_last,
            d_first, d_first+1, ..., d_center−1}
    endif
endif
```

Figure 7. The *weighted_sort* procedure

Figure 8(a) shows the U-cube algorithm executed on an all-port architecture, which requires 4 time steps to perform the multicast. Each arc represents a unicast, and is labeled with the time step in which it occurs. In this example, intermediate routers are not represented. Notice that node 7 cannot send to nodes 11 and 12 during the same time step, since both unicasts require the same outgoing channel.

Figure 8(b) shows the Maxport algorithm applied directly to address sequence D. In this example, the Maxport algorithm also requires 4 steps to reach all destination nodes. Notice that all unicasts with a common source node are transmitted on different outgoing channels, and thus can be sent during the same time step in an all-port architecture.

Now, we consider rearranging the nodes in the destination address sequence before beginning the multicast. Applying the *weighted_sort* algorithm to address sequence D produces a new address sequence $D = \{0, 1, 3, 5, 7, 14, 15, 12, 11\}$. Subcube $S = (3, 1)$ contains destination nodes {11, 12, 14, 15}. The two halves of subcube S, $S_0 = (2, 10)$ and $S_1 = (2, 11)$, contain destination nodes {11} and {12, 14, 15}, respectively. Thus, the *weighted_sort* algorithm interchanges S_0 and S_1, since S_0 contains fewer destination nodes than S_1. This interchange results in the more populated subcube (S_1) receiving the message first. Continuing recursively, the two halves of subcube S_1 are also interchanged. Figure 8(c) shows the resulting W-sort multicast, which requires only 2 steps.

Theorem 5 *The* weighted_sort *algorithm applied to a cube-ordered chain* $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ *results in* $\hat{D} = \{\hat{d}_{first}, \hat{d}_{first+1}, \ldots, \hat{d}_{last}\}$, *where:*

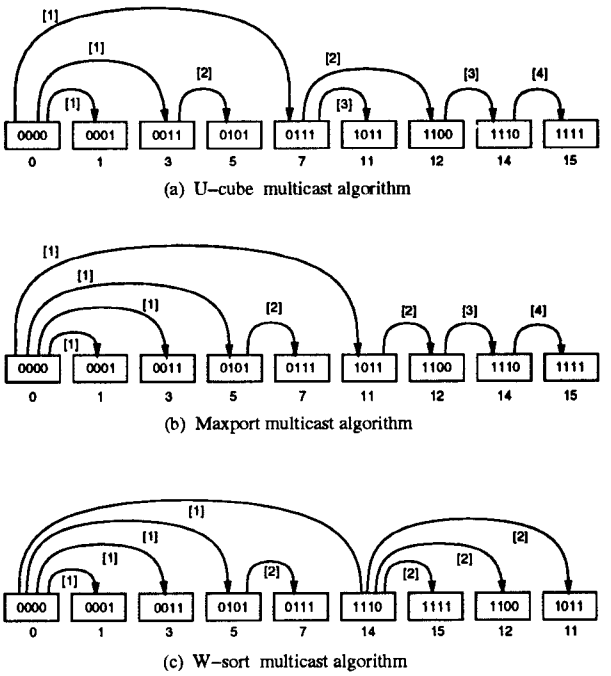

Figure 8. Examples of multicast communication

1. $\hat{D}$ is a cube-ordered chain;
2. $\hat{D}$ is a permutation of D; and
3. $\hat{d}_{first} = d_{first}$ (the source node remains in the first position).

Theorem 6 *The* W-sort *algorithm applied to a cube-ordered chain* $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ *results in a contention-free multicast from source node* d_{first} *to destination nodes* $\{d_{first+1}, d_{first+2}, , \ldots, d_{last}\}$.

5 Performance evaluation

In order to understand the relative performance of the algorithms presented in Section 4, they were compared in three ways on destination sets in which the nodes are randomly distributed throughout the hypercube. First, we compared their performance in terms of the average and maximum number of steps required to reach the destinations. Second, we compared the algorithms by implementing them on an nCUBE-2 and measuring the average and maximum delay, across destinations, for messages of various sizes. Third, we simulated the performance of the algorithms using a simulation tool that has been validated against the nCUBE-2. Since we had access to a real system with only 64 nodes, simulation allowed us to compare the algorithms on larger systems.

5.1 Stepwise comparisons

Figures 9 and 10 plot the averages, among random sets of destinations, of the maximum number of steps needed to multicast data in a 6-cube and a 10-cube, respectively. For each point in a curve, 100 destination sets were chosen randomly. In addition to reducing the number of steps, the new algorithms "smooth out" the staircase behavior of the U-cube algorithm.

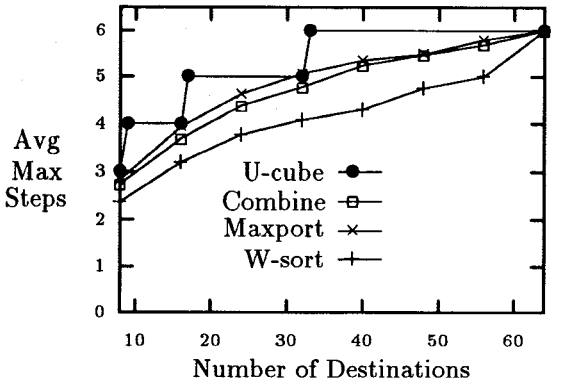

Figure 9. Stepwise comparisons on a 6-cube

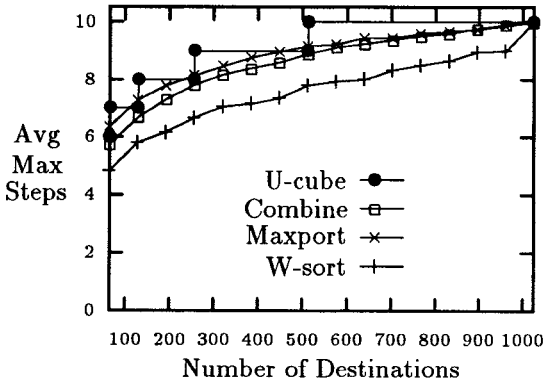

Figure 10. Stepwise comparisons on a 10-cube

5.2 Implementations on an nCUBE-2

Figures 11 and 12 plot the average and maximum, among destinations, of the measured delay between the sending of a 4096-byte multicast message and its receipt at the destination. For each point in a curve, 20 destination sets were chosen randomly in a 5-cube. These plots show that all the algorithms designed to take advantage of the multiport architecture offer some benefit over the U-cube algorithm. However, any advantage of among Maxport, Combine, and W-sort, is unclear. Interestingly, Figure 11 shows that the average delay for U-cube is actually worse for multicast than for broadcast. This anomaly occurs because the algorithm sometimes forces multiple messages out the same channel instead of taking advantage of multiple channels. In Figure 12, we see clearly the staircase behavior of U-cube. As predicted by the stepwise comparisons, the new algorithms tend to smooth the relative delays among various sized destination sets.

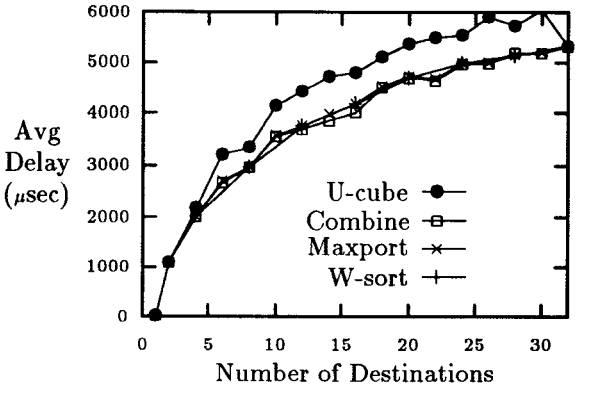

Figure 11. Average delay comparisons on a 5-cube

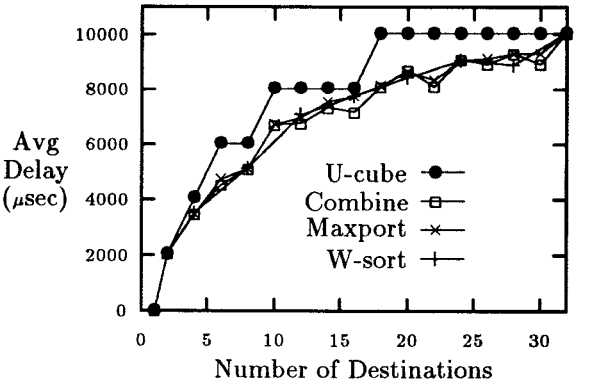

Figure 12. Maximum delay comparisons on a 5-cube

5.3 Simulations of larger systems

In order to compare the algorithm for larger hypercubes, we relied on simulation. As part of an earlier project [11], we have developed a CSIM-based simulation tool, called *MultiSim*, which can be used to simulate large-scale multiprocessors. In particular, MultiSim uses novel methods to efficiently simulate wormhole-routed systems. In addition, the simulator has been validated against an nCUBE-2 hypercube multicomputer [11].

Figures 13 and 14 plot the average and maximum, among destinations, of the delay between the sending of a 4096-byte multicast message and its receipt at the destination. For each point in a curve, 100 destination sets were chosen randomly in a 10 cube. These plots show that all the algorithms designed to take advantage of the multiport architecture offer advantages over the U-cube algorithm. For the larger systems, the advantage of W-sort becomes more obvious in both the average and maximum cases.

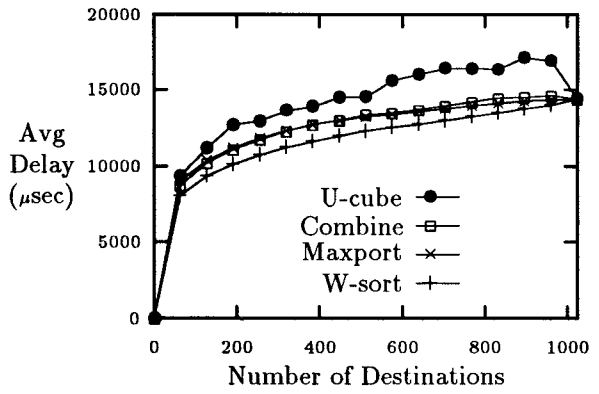

Figure 13. Average delay comparisons on a 10-cube

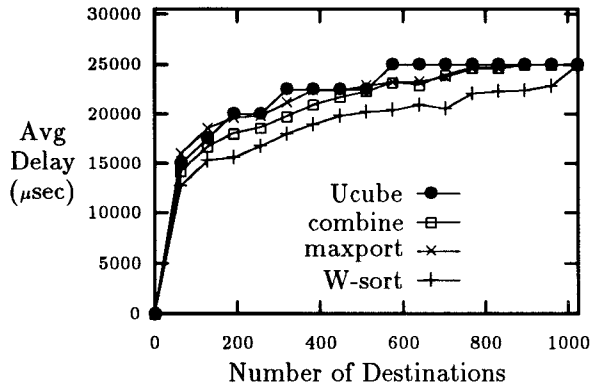

Figure 14. Maximum delay comparisons on a 10-cube

6 Conclusions

Efficient data distribution is critical to the performance of new generation supercomputers that use massively parallel architectures. In this paper, the problem of collective data distribution, specifically multicast in all-port wormhole-routed hypercubes, has been addressed. It has been demonstrated why the U-cube multicast algorithm [9], which is optimal for one-port architectures, fails to take advantage of multiple ports when they are present in the system. New theoretical results regarding contention among messages in wormhole-routed hypercubes have been developed and used to design new multicast routing algorithms and to prove that these algorithms are contention-free. The algorithms were compared in terms of the number of steps required in each, their measured execution times when implemented on a relatively small-scale nCUBE-2, and their simulated execution times on larger hypercubes. The results indicate that significant performance improvement is possible when the multicast algorithm actively identifies and uses multiple ports in parallel.

References

1. S. Berryman, J. Cownie, J. Dongarra, and others, "Document for standard message-passing interface." in preparation, July 1993.
2. High Performance Fortran Forum, "Draft High Performance Fortran language specification." (version 1.0), Jan. 1993.
3. W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.
4. L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, pp. 62–76, Feb. 1993.
5. S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, vol. C-38, pp. 1249–1268, Sept. 1989.
6. C. L. Seitz, "The cosmic cube," *Communications of the ACM*, vol. 28, pp. 22–33, January 1985.
7. NCUBE Company, *NCUBE 6400 Processor Manual*, 1990.
8. R. Duzett and R. Buck, "An overview of the nCUBE-3 supercomputer," in *Proc. Frontiers'92: The 5th Symposium on the Frontiers of Massively Parallel Computation*, pp. 458–464, Oct. 1992.
9. P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni, "Unicast-based multicast communication in wormhole-routed networks," in *Proc. of the 1992 International Conference on Parallel Processing*, vol. II, pp. 10–19, Aug. 1992.
10. D. F. Robinson, D. Judd, P. K. McKinley, and B. H. C. Cheng, "Efficient collective data distribution in all-port wormhole-routed hypercubes," Tech. Rep. MSU-CPS-93-8, Department of Computer Science, Michigan State University, East Lansing, Michigan, Apr. 1993.
11. P. K. McKinley and C. Trefftz, "MultiSim: A tool for the study of large-scale multiprocessors," in *Proc. 1993 International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Networks (MASCOTS 93)*, pp. 57–62, Jan. 1993.