Information Processing Letters 46 (1993) 219–224
Elsevier

9 July 1993

An efficient algorithm for multiple simultaneous broadcasts in the hypercube *

George D. Stamoulis ** and John N. Tsitsiklis

Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Communicated by D. Gries
Received 8 December 1992
Revised 30 March 1993

Abstract

Stamoulis, G.D. and J.N. Tsitsiklis, An efficient algorithm for multiple simultaneous broadcasts in the hypercube, Information Processing Letters 46 (1993) 219–224.

We analyze the following problem: Each of K nodes of the d-cube wishes (at the same time) to broadcast a packet to all hypercube nodes. We present a simple distributed algorithm for performing this task efficiently for any value of K and for any K-tuple of broadcasting nodes, and some variations of this algorithm that apply to special cases. In particular, we obtain an easily implementable algorithm for the multinode broadcast task ($K = 2^d$), which comes within a factor of 2 from the optimal.

Keywords: Broadcast; distributed systems; hypercube communications; routing algorithms

1. Introduction

During execution of parallel algorithms in a network of processors, subsets of processors sometimes wish to broadcast simultaneously pieces of information to all others. We present an efficient, yet simple to implement, algorithm for performing such simultaneous broadcasts in the hypercube network.

We consider the d-dimensional *hypercube* (or d-cube); see e.g. [2]. This network consists of 2^d nodes, numbered from 0 to $2^d - 1$. Associated with each node z is a binary identity $(z_d, \ldots, z_1)$, which coincides with the binary representation of the number z. There exist arcs only between nodes whose binary identities differ in a single bit. That is, arc (z, y) exists if and only if $z_i = y_i$ for $i \neq m$ and $z_m \neq y_m$ for some $m \in \{1, \ldots, d\}$. Note that (z, y) stands for a unidirectional arc pointing from z to y; of course, if arc (z, y) exists, so does arc (y, z). The d-cube has $d2^d$ arcs and its diameter is d. Other properties of the hypercube that are used in our analysis are presented in Section 2.1.

The underlying assumptions for communications are as follows: The time axis is divided into slots of unit length; all nodes are following the same clock. Each piece of information is transmitted as a packet of unit length. Only one packet can traverse an arc per slot; all transmissions are error-free. Each node may transmit packets

Correspondence to: J.N. Tsitsiklis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. Email: jnt@athena.mit.edu.

* Research supported by the NSF under Grant ECS-8552419 and by the ARO under Grant DAAL03-86-K-0171.

** Email: stamouli@theseas.ntu.gr.

through all of its output ports and at the same time receive packets through all of its input ports. Moreover, each node has infinite buffer capacity.

In the problem analyzed in this paper, it is assumed that each of a subset of K nodes of the d-cube wishes to broadcast a packet. We observe in Section 3.1 that, for any routing algorithm, the time required to perform these simultaneous broadcasts in the absence of other transmissions is $\Omega(\max\{d, K/d\})$, for any K-tuple of broadcasting nodes. We then devise a simple distributed algorithm which for any $K \leqslant 2^d$ and for any K-tuple of broadcasting nodes, comes within a factor of 6 of the lower bound. For $K \gg d^2$, the algorithm is within a factor of 2 of the lower bound. [1] The algorithm works even if no node of the hypercube knows K or the identities of any other broadcasting nodes. It uses a first phase, during which the broadcasting nodes coordinate in a decentralized fashion; this phase involves a parallel prefix task (see Section 2.2). We also present a randomized variation of this algorithm, which does not involve the prefix task; when randomization is employed, the completion time is $\Theta(\max\{d, K/d\})$ and the task is accomplished correctly with high probability. Finally, in Section 3.3, we present some other efficient algorithms for the special cases $K = O(d)$ and $K = d$.

The simplest communication task involving broadcasting is the *single node broadcast*, where exactly one of the nodes wishes to broadcast a packet. This can be accomplished in d time units, by using a spanning tree with shortest paths. The single node broadcast is an extreme case of the problem analyzed in this paper, corresponding to $K = 1$. The other extreme case, namely $K = 2^d$, corresponds to the *multinode broadcast*, where all nodes wish to perform a broadcast at the same time; see [2]. For hypercubes, the minimum possible time for this task, $\lceil (2^d - 1)/d \rceil$, is attained by an algorithm by Bertsekas et al. [1]. Previously, Saad and Schultz [6], as well as Johnsson and Ho [3], had constructed optimal or nearly optimal multinode broadcast algorithms for hypercubes, under somewhat different assumptions on packet transmissions. Our algorithm specialized to the multinode broadcast problem completes in time $2\lceil 2^d/d \rceil + 2d - 1$; this is a factor of 2 from the optimal, but the algorithm is much simpler and easier to implement than previously available algorithms. The multinode broadcast task arises in the distributed execution of iterative algorithms of the form $x := f(x)$, where $f: \mathbb{R}^n \to \mathbb{R}^n$ and n is the number of nodes; typically, node i knows the function f_i and updates x_i. Assume that the problem is dense, i.e. each entry of the function $f(x)$ depends explicitly on almost all entries of x; then, once x_i is updated, its new value must be broadcast to all other nodes, in order to be used in their subsequent calculations. If all nodes are perfectly synchronized, then all entries of the vector x are broadcast at the same time, which gives rise to a multinode broadcast. However, there are cases where not all of the x_i's are updated at the same time; e.g., in multigrid or Gauss–Seidel algorithms. It is in such cases that a simultaneous broadcast by a subset of $K \neq n$ nodes arises.

[1] In subsequent work, Varvarigos and Bertsekas [7], devised an algorithm whose completion time is faster by about a factor of 2 and which is within an additive constant of the optimal for $K \gg d^2$.

2. Background material

2.1. Definitions

Let z and y be two nodes of the d-cube. We denote by $z \oplus y$ the vector $(z_d \oplus y_d, \ldots, z_1 \oplus y_1)$, where $\oplus$ is the symbol for the XOR operation. The ith (from the right) entry of $z \oplus y$ equals 1, if and only if $z_i \neq y_i$. For $j \in \{1, \ldots, d\}$, we denote by e_j the node for which all entries of its binary identity equal 0 except for the jth one (from the right), which equals 1. Nodes $e_1, \ldots, e_d$ are the only neighbors of node $(0, \ldots, 0)$. In general, each node z has exactly d neighbors, namely nodes $z \oplus e_1, \ldots, z \oplus e_d$. Clearly, arc (z, y) exists if and only if $z \oplus y = e_m$ for some $m \in \{1, \ldots, d\}$. Such an arc is said to be of *type* m; the set of arcs of type m is called the mth *dimension*.

2.2. The completely unbalanced spanning tree

For two nodes z and y, let $i_1 < \cdots < i_k$ be the only entries of $z \oplus y$ that equal 1; k is called the *Hamming distance* between z and y. Any shortest path from z to y consists of k arcs, with one of them being of type i_1, one of them being of type i_2, etc. A packet originating at z will reach node y if it traverses exactly one arc of each of these types, regardless of the order in which it crosses the various hypercube dimensions.

A completely unbalanced spanning tree rooted at some node z is defined as the spanning out-tree [2] with the following property: Every node y is reached from the root z through the unique shortest path in which the hypercube dimensions are crossed in increasing index-order. That is, if $i_1 < \cdots < i_k$ are the dimensions to be crossed in any shortest path from z to y, then the tree under consideration contains that shortest path where the first arc belongs to dimension i_1, the second arc to dimension i_2, etc. One can easily see that this collection of paths constitutes a tree.

A completely unbalanced spanning tree T rooted at node z had d subtrees $T_1, \ldots, T_d$. Each of them is rooted at one of the neighbors of z. Subtree T_i consists of all nodes y with the following property: $y_1 = z_1, \ldots, y_{i-1} = z_{i-1}$ and $y_i \neq z_i$. Therefore, T_i contains 2^{d-i} nodes, hence the terminology "completely unbalanced". By considering different index-orders for crossing the hypercube dimensions, we can obtain other trees, isomorphic to the tree T defined earlier. Henceforth, we call all of these trees completely unbalanced, as well. Completely unbalanced trees have been used extensively in algorithms for hypercube communications (see [6], [3] and [1]). Johnsson and Ho [3] use the terminology "spanning binomial tree".

2.3. The d disjoint spanning trees

Johnsson and Ho [3] have constructed an imbedding of d disjoint (directed) spanning out-trees in the d-cube; they call them "d Edge-Disjoint Spanning Binomial Trees" (dESBT). This imbedding consists of d completely unbalanced trees $T^{(1)}, \ldots, T^{(d)}$. Tree $T^{(j)}$ is rooted at node e_j. The index-order of crossing the hypercube dimensions in the paths of tree $T^{(j)}$ is as follows:

$$(j \bmod d) + 1, \quad [(j+1) \bmod d] + 1, \ldots, \quad [(j+d-1) \bmod d] + 1.$$

2.4. Parallel prefix

Let $a_0, \ldots, a_{2^d-1}$ be given scalars. A special case of the *prefix* problem [4] is defined as follows: Compute all partial sums of the form $\sum_{y=x}^{2^d-1} a_y$. This prefix problem can be solved efficiently in parallel in time $2d$, by using $2^{d+1} - 1$ processors connected in a complete binary tree with bidirectional arcs [5]. It can also be solved in the d-cube in time $2d$, by embedding such a tree in the d-cube [5]. At the end, node x knows the value of $\sum_{y=x}^{2d-1} a_y$.

3. The results

3.1. Lower bounds

We observe that under any routing algorithm, K broadcasts involve a total of at least $(2^d - 1)K$ packet transmissions while at most $d2^d$ transmissions may be performed in each slot. Taking also into account that the diameter of the d-cube is d, we see that the task of interest requires at least

$$\max\left\{d, \frac{(2^d-1)}{d2^d}K\right\} = \Omega\left(\max\left\{d, \frac{K}{d}\right\}\right)$$

slots. In the analysis to follow, the K broadcasting nodes will be assumed distinct, unless otherwise specified.

As already mentioned in Section 1, we are interested in an algorithm that attains the optimal order of magnitude $\Theta(\max\{d, K/d\})$ of the completion time, for any K and for any K-tuple of broadcasting nodes. The simplest possible distributed algorithm for our task would be as fol-

[2] All spanning trees considered throughout the paper are directed, unless otherwise specified. Also, an out-tree is a tree emanating from its root.

lows: Each of the 2^d nodes of the hypercube is confined to broadcast its packet (if it has one) along a prespecified spanning tree. Unfortunately, such an algorithm would not always attain the optimal order of magnitude for the completion time. Indeed, for any fixed node x and for any of the 2^d prespecified trees except for the one rooted at x, there exists exactly one arc of the form $(x \oplus e_j, x)$ that belongs to the tree. Thus, there exists some arc $(x \oplus e_{j^*}, x)$ that belongs to at least $(2^d - 1)/d$ of the trees. Therefore, as long as $K \leqslant (2^d - 1)/d$, an adversary can choose the K broadcasting nodes in such a way that all of the packets will be received by node x through arc $(x \oplus e_{j^*}, x)$; in such a case the broadcasts last for at least K time units. The above argument shows that, in the worst case, the completion time of the task will not be of the optimal order of magnitude, unless there is some flexibility in choosing the paths to be followed by the packets.

3.2. The algorithm

In this subsection, we present a distributed algorithm for performing K simultaneous broadcasts in time $\Theta(\max\{d, K/d\})$ for any choice of K and of the broadcasting nodes. The main idea of the algorithm is as follows: The K packets to the broadcast are split evenly among the d disjoint spanning trees; each of the packets is sent to the root of one of these d trees, which will eventually broadcast the packet along that tree. In more detail, the algorithm consists of three phases:

Phase 1: A prefix task is implemented (see Section 2.2), with input $a_0, \ldots, a_{2^d-1}$, where $a_x = 1$ if node x wishes to broadcast a packet, and $a_x = 0$ otherwise. This task lasts for $2d$ time units. After completion of this prefix computation, node x knows the value of $\sum_{y=x}^{2^d-1} a_y = {}^{\text{def}} r_x$; notice that if node x is to broadcast a packet, then r_x equals its rank under the decreasing order within the subset of broadcasting nodes. Clearly, we have $r_0 = K$; node $(0, \ldots, 0)$ also has to transmit this value to its neighbors $e_1, \ldots, e_d$. The total duration of this phase is $2d + 1$ slots.

Phase 2: For each broadcasting node x, its respective packet is sent to the root $e_{j(x)}$ of tree $T^{(j(x))}$, where the index $j(x)$ is determined by the following rule: $j(x) = {}^{\text{def}} (r_x - 1) \bmod d + 1$. Let N_j be the number of packets to be received by root e_j; since the r_x's of the broadcasting nodes are distinct and consecutive, taking all the values $K, \ldots, 1$, it follows easily that N_j equals either $\lfloor K/d \rfloor$ or $\lceil K/d \rceil$, for all $j \in \{1, \ldots, d\}$. Therefore, the packets to be broadcast are split among the d disjoint trees as evenly as possible. The path to be followed by the packet of node x is the reverse of the path from $e_{j(x)}$ to x that is contained in $T^{(j(x))}$. Since the d disjoint trees remain disjoint after reversing all their constituent arcs, packets sent to different roots do not interfere. Due to pipelining, all N_j packets destined for root e_j will have been received after at most $N_j + d - 1$ slots from the beginning of the present phase. Therefore, all the transmissions involved in this phase will have been completed after $\max_{j=1,\ldots,d}\{N_j + d - 1\} = \lceil K/d \rceil + d - 1$ slots. Termination of the phase can be detected individually by each root e_j at time $\lceil K/d \rceil + d - 1$, because nodes $e_1, \ldots, e_d$ received the value of K at the last slot of the first phase. (Notice that the rest of the nodes do not have to detect termination of this phase, because they are not supposed to trigger the next phase.)

Phase 3: Each of the roots $e_1, \ldots, e_d$ broadcasts the packets received during the second phase. Root e_j broadcasts the corresponding N_j packets along $T^{(j)}$; just after forwarding the N_jth packet, root e_j starts broadcasting [along $T^{(j)}$] a termination packet. Again, packets broadcast along different trees do not interfere. By pipelining successive broadcasts over the same tree and taking the termination packets into account, it follows that this phase lasts for $\max_{j=1,\ldots,d}\{N_j + d\} = \lceil K/d \rceil + d$ slots.

It follows from the description of the algorithm that its total duration is $2\lceil K/d \rceil + 4d$, which is $\Theta(\max\{d, K/d\})$. For $K \gg d^2$, the completion time of the algorithm exceeds the lower bound $\max\{d, ((2^d - 1)/d2^d)K\}$ by a factor that is very close to 2. In fact, for the case $K = 2^d$, which corresponds to a multinode broadcast, the first

phase of the algorithm is not necessary, because it is known that $r_x = 2^d - x$ for every node x. We thus obtain a multinode broadcast algorithm with duration $2\lceil 2^d/d \rceil + 2d - 1$, which exceeds the optimal value $\lceil (2^d - 1)/d \rceil$ by a factor of 2. However, the suboptimal algorithm just derived is much simpler to implement than the multinode broadcast algorithms of [6], [3] and [1]. Indeed, our algorithm involves a total of $d + 1$ spanning trees, whereas the latter involve a total of at least 2^d trees; also the trees used by the algorithm discussed above can be described in a rather concise way, which reduces its memory requirements even further. For $K \ll d^2$, the completion time of the algorithm exceeds the lower bound $\max\{d, ((2^d - 1)/d2^d)K\}$ by a factor that is close to 4; finally, for $K = \Theta(d^2)$, the corresponding factor is between 2 and 6, with the worst case arising for $K = d^2$. (It should also be noted that the quantity $\max\{d, ((2^d - 1)/d2^d)K\}$ is not necessarily a tight lower bound for the completion time of the task.) It is worth noting that $K = \Theta(d^2)$ is the largest order of magnitude for K that can possibly lead to a completion time of $\Theta(d)$, i.e. of the same order of magnitude as the time for a single node broadcast.

Finally, it should be noted that the first phase can be avoided, by employing *randomization*. Indeed, assume that each of the broadcasting nodes x selects randomly the value of $j(x)$, with $\Pr[j(x) = i] = 1/d$ for all $i \in \{1, \ldots, d\}$. Using the Chernoff bound, we obtain

$$\Pr\left(\max\{N_1, \ldots, N_d\} \geqslant C\frac{K}{d}\right) \leqslant 2d\left(\frac{\mathrm{e}}{C}\right)^{CK/d}. \quad (1)$$

We distinguish two cases. If $K \geqslant 2d \log d$, then, for any fixed $C \geqslant 2\mathrm{e}$, some straightforward algebra yields

$$\Pr(\max\{N_1, \ldots, N_d\} \geqslant CK/d) \leqslant 2^{-C}.$$

It follows that the algorithm is guaranteed to terminate within $4CK/d + 2d$ time units, except for an event whose probability is bounded by 2^{-C}. In the second case that we consider, we assume that $K \leqslant d^2/2\mathrm{e}$. (For large enough d, these two cases are exhaustive.) Using (1) with $C = d^2/K$, we see that $\Pr(\max\{N_1, \ldots, N_d\} \geqslant d) \leqslant 2d \cdot 2^{-d}$ and the algorithm terminates in time $4d$, with high probability. We point out that because the duration of phase 2 is random, the algorithm has to be refined somewhat so that the root nodes can find out when this phase has ended. Alternatively, we can allow the root nodes to start broadcasting (phase 3) as soon as they receive the first packet to be broadcasted but let phase 3 packets have priority over phase 2 packets.

It has been assumed so far that the K broadcasting nodes were distinct. If this is not the case, the value of a_x (in the prefix computation) should be set to the number of packets to be broadcast by node x; K now stands for the total number of packets to be broadcast. If node x has $a_x \geqslant 2$ packets, then it should send the mth packet to the root indexed by $(r_x - a_x - 1 + m) \bmod d + 1$, for $m = 1, \ldots, a_x$.

3.3. Further results for some special cases

Next, we present some simple algorithms for cases where K is known to have a special value.

3.3.1. The case $K = O(d)$

Consider the following distributed algorithm: Each of the K nodes broadcasts its packet along a completely unbalanced spanning tree rooted at itself, with all these trees having the same index-order of crossing the hypercube dimensions; e.g. the increasing index-order. Suppose that a copy $\mathscr{P}_{z,j}(x)$ of the packet originating at a node x wishes to traverse some arc $(z, z \oplus e_j)$ at the same time with the copy $\mathscr{P}_{z,j}(y)$ of another packet originating at node y. Then, both $\mathscr{P}_{z,j}(x)$ and $\mathscr{P}_{z,j}(y)$ are destined for the same subset of nodes, namely all nodes of the form $z \oplus v$ with $v_1 = \cdots = v_{j-1} = 0$ and $v_j = 1$. Therefore, if $\mathscr{P}_{z,j}(x)$ traverses arc $(z, z, \oplus e_j)$ before $\mathscr{P}_{z,j}(y)$, then $\mathscr{P}_{z,j}(y)$ (or copies thereof to be generated later) will never be delayed again due to copies of the packet originating at node x. This argument implics that each copy of a packet suffers at most $K - 1$ units of delay caused by contention; thus, the algorithm terminates after at most $d + K - 1$ time units. Unfortunately, this upper bound for the completion time is of the optimal order of magnitude $\Theta(\max\{d, K/d\})$ only if K is $O(d)$;

moreover, since each node is confined in a prespecified spanning tree, there are cases where the algorithm does not complete in $\Theta(\max\{d, K/d\})$ time units (see Section 3.1). The algorithm above is faster than the one presented in Section 3.2 for all $K \leqslant 3d$.

3.3.2. The case $K = d$

For $K = d$, the algorithm of Section 3.3.1 lasts for at most $2d - 1$ slots. Below, we present an algorithm that completes in d time units; however, this algorithm assumed that each broadcasting node x knows its rank r_x within the d-tuple of broadcasting nodes. The algorithm is as follows: Node x will broadcast its packet along the completely unbalanced spanning tree (rooted at x) in which the hypercube dimensions are crossed in the following index-order: $r_x \bmod d + 1$, $(r_x + 1) \bmod d + 1, \ldots, (r_x + d - 1) \bmod d + 1$; moreover, at the mth slot, the packet of node x may only cross the permissible arcs of dimension $(r_x + m - 2) \bmod d + 1$. To see that copies of different packets never collide, it suffices to see that $(r_x + m - 2) \bmod d + 1 \neq (r_y + m - 2) \bmod d + 1$ for $x \neq y$; this follows from the fact $r_x \neq r_y$ while both r_x and r_y belong to $\{1, \ldots, d\}$.

As already established in Section 3.2, the ranks of the broadcasting nodes can be computed in $2d$ time slots, by running a parallel prefix phase. If this overhead is taken into account, then the total duration of the algorithm would be $3d$ slots; this is better than the time $4d + 2$ taken by the algorithm of Section 3.2, but it exceeds the completion time attained by the simple algorithm of Section 3.3.1. Of course, if the same d-tuple of nodes is to perform a simultaneous broadcast several times, then the computation of the ranks should be carried out only once; in such a case, the present algorithm might be preferable. In the extreme case where *one* node has d packets to broadcast, then the parallel prefix computation is redundant, and the algorithm takes d time units, which is the fastest possible.

Acknowledgment

The authors are grateful to Tom Leighton for helpful suggestions.

References

[1] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng and J.N. Tsitsiklis, Optimal communication algorithms for hypercubes, *J. Parallel Distributed Comput.* **11** (1991) 263–275.

[2] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[3] S.L. Johnsson and C.-T. Ho, Optimum broadcasting and personalized communication in hypercubes, *IEEE Trans. Comput.* **38** (1989) 1249–1267.

[4] R.E. Ladner and M.I. Fischer, Parallel prefix computation, *J. ACM 27* (1980) 832–838.

[5] T. Leighton and C.E. Leiserson, Theory of parallel and VLSI computation, Laboratory for Computer Science, Rept. LCS/RSS 6, M.I.T., 1990.

[6] Y. Saad and M.H. Schultz, Data communication in hypercubes, Dept. of Computer Sciences, Research Rept. YALEU/DCS/RR-428, Yale University, 1985.

[7] E.A. Varvarigos and D.P. Bertsekas, Dynamic broadcasting in parallel computing, Tech. Rept. LIDS-P-2111, Laboratory for Information and Decision Systems, M.I.T., Cambridge, MA, 1992.