December 1990 LIDS–P–2011

AN EFFICIENT ALGORITHM FOR MULTIPLE SIMULTANEOUS BROADCASTS IN THE HYPERCUBE †

by

George D. Stamoulis ‡ and **John N. Tsitsiklis** ‡

Abstract

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 a very easily implementable algorithm for the multinode broadcast task ($K = 2^d$), which comes within a factor of 2 from the optimal.

Key Words: Broadcast, distributed systems, hypercube communications, routing algorithms.

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

‡ Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Mass. 02139, USA; e-mail: stamouli@lids.mit.edu, jnt@lids.mit.edu.

1. INTRODUCTION

During the execution of parallel algorithms in a network of processors, it is often necessary that a subset of processors broadcast simultaneously pieces of information to all others. In this note, we present an efficient (yet simple to implement) algorithm for perfoming such simultaneous broadcasts in the hypercube network.

We consider the d-dimensional hypercube (or d-cube); e.g. see [BeT89]. 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 $z_m \neq y_m$ (or equivalently $|z - y| = 2^{m-1}$) 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). It is easily seen that the d-cube has $d2^d$ arcs; also the diameter of the d-cube equals d. Other properties of the hypercube that are used in our analysis are presented in §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 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 note, it is assumed that each of a subset of K nodes of the d-cube wishes to broadcast a packet. We prove in §3.1 that, for any routing algorithm, the time required to perform these simultaneous broadcasts in the absence of other transmissions is $\Omega(\max\{d, \frac{K}{d}\})$, for any K-tuple of broadcasting nodes. We then devise a simple distributed algorithm whose completion time is $\Theta(\max\{d, \frac{K}{d}\})$, for any $K \leq 2^d$ and for any K-tuple of broadcasting nodes. The algorithm works even if no node of the hypercube knows the value of K or the identities of any other broadcasting nodes. This algorithm uses a first phase, during which the broadcasting nodes coordinate in a decentralized fashion; this phase involves a parallel prefix task (see §2.2). It will be argued that such a coordination phase is necessary for an algorithm to attain a worst case completion time of $\Theta(\max\{d, \frac{K}{d}\})$, unless K is either very large or very small. We also present a randomized variation of this algorithm, which does not include the prefix task; when randomization is employed, the completion time is $\Theta(\max\{d, \frac{K}{d}\})$ and the task is accomplished correctly with high probability. Finally, we present some other efficient algorithms pertaining to the special cases $K = O(d)$, $K = 2$ and $K = d$; see §3.3.

The simplest communication task involving broadcasting is the single node broadcast, where exactly one of the nodes wishes to broadcast a packet; see [BeT89]. This task 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 note, corresponding to $K = 1$. The other extreme case, namely $K = 2^d$, corresponds to the multinode broadcast task, where all nodes wish to perform a broadcast at the same time; see [BeT89]. The minimum possible time for this task (in the d-cube) is $\lceil \frac{2^d-1}{d} \rceil$ and it is attained by an algorithm by Bertsekas et al. [BOSTT89]. Previously, Saad and Schultz [SaS85], as well as Johnsson and Ho [JoH89], 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 $2\lceil \frac{2^d}{d} \rceil + 2d - 1$; this is a factor of 2 far from the optimal, but the algorithm is much easier to implement than previously available algorithms. The multinode broadcast task arises in the distributed execution of any iterative algorithm of the form $x := f(x)$, where $f : \Re^n \to \Re^n$ and n is the number of nodes; typically, the ith node 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 explictly 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.

To the best of our knowledge, the results derived in this paper are new. The problem was considered later by Varvarigos and Bertsekas [VaB90], who used a completely different approach and derived an algorithm running in time $\Theta(\max\{d, \frac{K}{\ln K}\})$; this time is optimal only if K is very large [namely, if $\ln K = \Theta(d)$].

2. BACKGROUND MATERIAL

2.1 Background on the Hypercube Network

The fundamental properties of the hypercube network were briefly mentioned in §1. In this subsection, we present some additional background material.

2.1.1 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 numbered 2^{j-1}; that is, all entries of the binary identity of e_j 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.

2.1.2 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 (*) 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 of the 3-cube is presented in Fig. 1; the root of that tree is node $(0,0,0)$.

A completely unbalanced spanning tree T rooted at node z has 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 [SaS85], [BOSTT89] and [JoH89]). Johnsson and Ho [JoH89] use the terminology "spanning binomial tree".

2.1.3 The d Disjoint Spanning Trees

Johnsson and Ho [JoH89] 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:

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

In Fig. 2, we present this construction for $d = 3$; this figure is taken from [JoH89].

2.2 Parallel Prefix

Let $a_0,\ldots,a_{2^d-1}$ be given scalars. A special case of the prefix problem [LaF80] 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

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

with bidirectional arcs [LeL90]. The problem can also be solved in the d-cube in time $2d$, by embedding such a tree in the d-cube [LeL90]. At the end, node x knows the value of $\sum_{y=x}^{2^d-1} a_y$.

3. THE RESULTS

3.1 Lower Bounds

First, we establish a lower bound on the time required to perform K simultaneous broadcasts in the d-cube. Clearly, under any routing algorithm, K broadcasts involve a total of at least $(2^d-1)K$ packet transmissions. Since at most $d2^d$ transmissions may be performed in each slot, this task requires at least $\frac{(2^d-1)}{d2^d}K = \Theta(\frac{K}{d})$ time units. Furthermore, the completion time of a broadcast is no less than the diameter d of the d-cube. Hence, it follows that under any routing algorithm and for any K-tuple of broadcasting nodes, the task of interest takes time $\Omega(\max\{d, \frac{K}{d}\})$. In the analysis to follow, the K broadcasting nodes will be taken distinct, unless otherwise specified.

As already mentioned in §1, we are interested in devising an algorithm that attains the optimal order of magnitude $\Theta(\max\{d, \frac{K}{d}\})$ 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 follows: 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 $\frac{2^d-1}{d}$ of the trees. Therefore, as long as $K \leq \frac{2^d-1}{d}$, 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. [If K is $\Theta(d^{-\epsilon}2^d)$ with $0 < \epsilon < 1$, then the adversary can force $\Omega(d^{1-\epsilon}K)$ of the packets to be transmitted over the same arc.] The above argument shows that, in the worst case, the completion time of the task will not of the optimal order of magnitude, unless there is some flexibility in choosing the paths to be followed by the packets. This conclusion (and the argument we used) is reminiscent of an important result by Borodin and Hopcroft [BoH82] on the inefficiency of oblivious routing when performing a permutation task.

3.2 An Efficient Algorithm

In this subsection, we present a distributed algorithm for performing K simultaneous broadcasts in time $\Theta(\max\{d, \frac{K}{d}\})$ for any choice of K and of the broadcasting nodes. The main idea of the algorithm is as follows: The K packets to be 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 §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 $\sum_{y=x}^{2^d-1} a_y \overset{\text{def}}{=} r_x$; 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 and its termination can be detected individually by each node.

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: $j(x) \overset{\text{def}}{=} (r_x - 1) \bmod d + 1$. 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 $\lfloor \frac{K}{d} \rfloor$ or $\lceil \frac{K}{d} \rceil$, 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 intefere. 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 $\max_{j=1,\ldots,d}\{N_j + d - 1\} = \lceil \frac{K}{d} \rceil + d - 1$ slots. It is possible that these transmissions are completed earlier. However, unless K is a multiple of d, not all of the root nodes $e_1, \ldots, e_d$ can detect termination earlier, because a root e_j that has already received $\lfloor \frac{K}{d} \rfloor$ packets cannot tell whether or not there is still one more packet that it has yet to receive. On the other hand, termination of the phase can be detected individually by each root e_j at time $\lceil \frac{K}{d} \rceil + d - 1$, 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 triger the next phase.)

Phase 3: Each of the roots $e_1, \ldots, e_d$ broadcasts the packets received during the first 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 easily that this phase lasts for $\max_{j=1,\ldots,d}\{N_j + d\} = \lceil \frac{K}{d} \rceil + d$ slots; termination is detected individually by each node.

It follows from the description of the algorithm that its total duration is $2\lceil \frac{K}{d} \rceil + 4d$, which is $\Theta(\max\{d, \frac{K}{d}\})$. For $K \gg d^2$, the completion time of the algorithm exceeds the lower bound $\max\{d, \frac{2^d-1}{d2^d}K\}$ 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 completion time $2\lceil \frac{2^d}{d} \rceil + 2d - 1$, which exceeds the optimal value $\lceil \frac{2^d-1}{d} \rceil$ by a factor of 2. However, the suboptimal algorithm just derived is much simpler to implement than the multinode broadcast algorithms of [SaS85], [BOSTT89], and [JoH89]. Indeed, the former 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 $\max\{d, \frac{2^d-1}{d2^d}K\}$ 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 $\max\{d, \frac{2^d-1}{d2^d}K\}$ 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.

The algorithm presented above is distributed, that is, it does not require any centralized coordination. Moreover, the algorithm is non-oblivious, meaning that the paths followed by different packets are not selected independently.

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 $\Pr[j(x) = i] = \frac{1}{d}$ for all $i \in \{1, \ldots, d\}$. Then, by applying the Chernoff bound, it can be seen that for any $C > e$ there holds $\max\{N_1, \ldots, N_d\} \leq C\frac{K}{d}$ with high probability. Thus, we can fix a $C^* > e$ and run phase 2 (and resp. phase 3) for $C^*\frac{K}{d} + d - 1$ (and resp. $C^*\frac{K}{d} + d$) slots, without running phase 1 at all; then, the algorithm lasts for $2C^*\frac{K}{d} + 2d - 1$ slots and it accomplishes the task correctly with high probability (which depends on C^* and K). Alternatively, we can guarantee that the algorithm accomplishes the task correctly, and attain a completion time of $\Theta(\max\{d, \frac{K}{d}\})$ with high probability. This can be accomplished by running phase 2 until nodes $e_1, \ldots, e_d$ receive a total of K packets; termination of phase 2 can be detected in a distributed fashion (with a small overhead), because nodes 0 and $e_1, \ldots, e_d$ know the value of K.

Throughout the derivation of the algorithm, it was assumed 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 $a_x \geq 2$ 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 $\mathcal{P}_{z,j}(x)$ 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 $\mathcal{P}_{z,j}(y)$ of another packet originating at node y. Then, both $\mathcal{P}_{z,j}(x)$ and $\mathcal{P}_{z,j}(y)$ 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 $\mathcal{P}_{z,j}(x)$ traverses arc $(z, z, \oplus e_j)$ before $\mathcal{P}_{z,j}(y)$, then $\mathcal{P}_{z,j}(y)$ (or copies thereof to be generated later) will never be delayed again due to copies of the packet originating at node x. This argument implies 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 $\Theta(\max\{d, \frac{K}{d}\})$ 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 $\Theta(\max\{d, \frac{K}{d}\})$ time units (see §3.1). The algorithm above is faster than the one presented in §3.2 for all $K \leq 3d$.

3.3.2 The Case $K = 2$

Next, we present a distributed algorithm for performing 2 broadcasts simultaneously in d time units, which is clearly the fastest possible.

We denote by x and y the two dictinct broadcasting nodes; moreover, let $\bar{x}$ and $\bar{y}$ be the nodes at Hamming distance d from x and y, respectively. Using the distributed algorithm presented in §3.3.1, we can perform the 2 simultaneous broadcasts in at most $d+1$ time units. Since each copy of a packet suffers at most one unit of delay caused by contention (see §3.3.1), the algorithm will last for exactly d slots if the following property holds: Node $\bar{x}$ receives the packet of node x after d slots and at the same time node $\bar{y}$ receives the packet of node y after d slots. This is guaranteed by introducing the following simple priority discipline: Assume that two copies $\mathcal{P}_{z,j}(x)$ and $\mathcal{P}_{z,j}(y)$ of the packets under broadcast collide at arc $(z, z \oplus e_j)$; if node $\bar{x}$ (and resp. node $\bar{y}$) will eventually receive a copy of $\mathcal{P}_{z,j}(x)$ [and resp. of $\mathcal{P}_{z,j}(y)$], then $\mathcal{P}_{z,j}(x)$ [and resp. $\mathcal{P}_{z,j}(y)$] should be transmitted first. To see that this priority discipline works, it suffices to show the following property: If $\mathcal{P}_{z,j}(x)$ is destined for $\bar{x}$, then $\mathcal{P}_{z,j}(y)$ cannot be destined for node $\bar{y}$, and vice versa. To see this notice that when the hypercube dimensions are crossed in increasing index-order (or in any other prespecified index-order), then the paths from x to $\bar{x}$ and from y to $\bar{y}$ are disjoint. Indeed, if an arc $(w, w \oplus e_j)$ were shared by these two paths, then we would have $w_1 \neq x_1, \ldots, w_{j-1} \neq x_{j-1}, w_j = x_j, \ldots, w_d = x_d$ and at the same time $w_1 \neq y_1, \ldots, w_{j-1} \neq y_{j-1}, w_j = y_j, \ldots, w_d = y_d$; these imply that $x = y$, which is a contradiction.

3.3.3 The Case $K = d$

For $K = d$, the algorithm of §3.3.1 lasts for at most $2d - 1$ slots. This upper bound can be tightened by introducing priority disciplines such as the one presented in §3.3.2. Nevertheless, there still exist cases where the algorithm would take more than d time units. Below, we present an algorithm that completes in d time units; however, this algorithm assumes 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 §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 §3.2, but it exceeds the completion time attained by the simple algorithm of §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.

4. CONCLUSION

In this note, we have considered the communication task where, at he same time, each of K nodes of the d-cube wishes to broadcast a packet. The parameter K was allowed to take any value from 1 to 2^d. We have analyzed a distributed algorithm that attains the optimal order of magnitude for the completion time of this task for any value of K and for K-tuple of broadcasting nodes; this algorithm is very simple to implement. The communication task discussed in this note is a generalization of the multinode broadcast task, where all nodes of a network wish to perform a broadcast simultaneously.

Acknowledgement: The authors are grateful to Tom Leighton for his helpful suggestions.

REFERENCES

[BeT89] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall.

[BOSTT89] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis, "Optimal Communication Algorithms for Hypercubes". Laboratory for Information and Decision Systems, Report LIDS-P-1847, M.I.T.

[BoH82] A. Borodin and J.E. Hopcroft, "Routing, Merging and Sorting on Parallel Models of Computation", *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pp. 338-344.

[JoH89] S.L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Trans. Comput.*, vol. 38, pp. 1249-1267.

[LaF80] R.E. Ladner and M.I. Fischer, "Parallel Prefix Computation", *J. ACM*, vol. 27, pp. 832-838.

[LeL90] T. Leighton and C.E. Leiserson, "Theory of Parallel and VLSI Computation", Laboratory for Computer Science, Report LCS/RSS 6, M.I.T.

[SaS85] Y. Saad and M.H. Schultz, "Data Communication in Hypercubes", Dept. of Computer Sciences, Research Report YALEU/DCS/RR-428, Yale University.

[VaB90] E.A. Varvarigos and D.P. Bertsekas, personal communication.

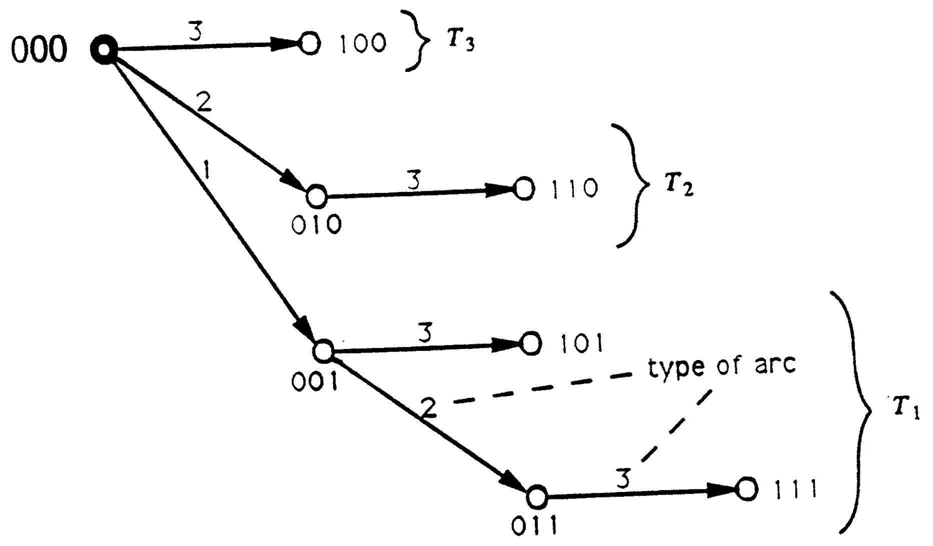

Figure 1: A completely unbalanced spanning tree, for $d = 3$.

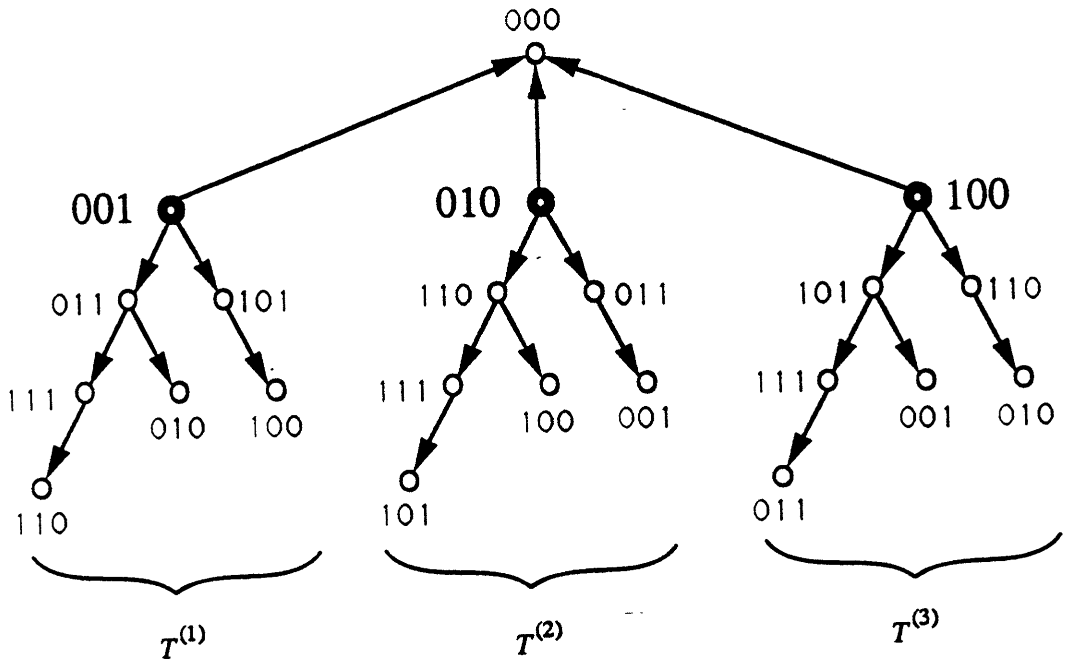

Figure 2: The *d* Disjoint Spanning Trees.