Multiple Message Broadcasting with Generalized Fibonacci Trees

Jehoshua Bruck Robert Cypher Ching-Tien Ho
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
{bruck,cypher,ho}@almaden.ibm.com

Abstract

We present efficient algorithms for broadcasting multiple messages. We assume n processors, one of which contains m packets that it must broadcast to each of the remaining n − 1 processors. The processors communicate in rounds. *In one round each processor is able to send one packet to any other processor* and *receive one packet from any other processor. We give a broadcasting algorithm which requires* $m + \log n + 3\log\log n + 15$ *rounds. In addition, we show a simple lower bound of* $m + \lceil \log n \rceil - 1$ *rounds for broadcasting in this model.*

1 Introduction

Broadcasting is one of the most important communication operations in both scientific and commercial parallel applications. In scientific applications, broadcasting is used for loading copies of a single program into each of the processors in a parallel machine and in a variety of linear algebra algorithms [10, 14] including matrix multiplication [15], Gaussian elimination [17], [7], LU-factorization [11], and Householder transformations. In commercial applications, broadcasting is frequently used in the database join operation [3].

Because of the importance of broadcasting, a great deal of research has been devoted to obtaining efficient broadcasting algorithms [12]. The majority of this research has assumed that the processors communicate through some specific interconnection topology and has attempted to optimize the broadcast algorithm to match the interconnection topology. However, several recent parallel architectures have supported a "fully-connected" model of communication in which it is equally efficient to send a packet between any given pair of processors. For example, GF-11 [4] is a synchronous, SIMD machine in which the processors are connected by a multistage Beneš network. The time required to send a packet between any pair of processors is independent of which pair of processors are communicating, and the Beneš network can support the simultaneous transmission of any n packets from n processors to n processors, provided that each processor sends at most one packet and receives at most one packet (and the pattern of communication is known at compile-time). Another example is the Vulcan parallel computer [5], in which each processor has a local memory and processors communicate by sending messages over a multistage network. Thus in Vulcan, each processor is the same distance from every other processor and there is no locality in the interconnection network. Other examples of parallel computers which support a "fully-connected" model of communication (despite having some locality in their interconnection networks) include Intel's iPSC/2, Thinking Machines' CM-2 and CM-5, and NCUBE Inc.'s NCUBE-10.

In this paper we study efficient broadcasting algorithms for a "fully-connected" model of parallel communication in which the communication is organized in *rounds*. In this model there is a collection of n processors, each of which has a local memory. During any one round, every processor can *both* send a single packet to any one other processor *and* receive a single packet from any one other processor.

In addition to accurately reflecting certain parallel architectures, there are many other advantages to this model of parallel communication. First, the fact that the rounds model does not assume any single topology makes it very general and flexible. For example, it allows the creation of algorithms that are portable between different machines, which can operate within arbitrary and dynamic subsets of processors, and which can operate in the presence of faults (assuming connectivity is maintained). Second, algorithms developed for the rounds model can be helpful for creating algorithms for specific topologies, either by finding an efficient mapping of the communication pattern of the algorithm developed for the rounds model to the desired topology, or by providing insight by focusing on the problem of scheduling send and receive calls rather than the problem of routing these calls. Third, even though the rounds model is synchronous, the assumption that each packet transmission takes unit time is not necessary for the correctness of the algorithm. Because sends and receives are paired, an asynchronous architecture which supports blocking sends and receives will implement algorithms designed for the rounds model correctly.

The broadcasting problem consists of sending m packets from one processor, called the *broadcaster*, to each of the $n - 1$ remaining processors. Broadcasting with this model of communication has been studied by several other researchers [1, 2]. Alon *et al.* showed that when n is a power of 2, or when n is a prime such that 2 is a generator of the multiplicative group modulo n, broadcasting $m = 1$ packet requires exactly $\lceil \log n \rceil$ rounds[1] [1]. Alon *et al.* also showed that for arbitrary n, broadcasting $m = 1$ packet can be performed in $2 \lceil \log n \rceil$ rounds, and they considered fault-tolerant broadcasting algorithms [1]. Bar-Noy and Kipnis gave an algorithm for arbitrary n which broadcasts m packets in $m + 2 \log n - O(1)$ rounds [2].

In this paper we will present a broadcasting algorithm that requires only $m + \log n + 3 \log \log n + 15$ rounds and works for arbitrary values of n. Therefore, when n is large compared to m, the algorithm presented here is almost twice as fast as the fastest previously known algorithm. In addition, we show a simple lower bound of $m + \lceil \log n \rceil - 1$ rounds, so the algorithm presented here is within an additive $O(\log \log n)$ term of optimal. Finally, it should be noted that a number of broadcasting algorithms have been designed for different models of communication. For example, the hypercube algorithm developed by Ho [13] yields an $m + \log n - 1$ rounds algorithm when n is a power of 2, exactly matching the lower bound mentioned above. However, no efficient technique is known for extending this algorithm to the case of arbitrary values of n. Also, optimal broadcasting algorithms have been developed for a model of communication in which each processor is only allowed to send *or* receive a single packet in a round [8, 9]. Although this more restricted model of communication does match some existing parallel architectures, other existing architectures allow simultaneous sends and receives [4, 5], and it is likely that many future architectures will support simultaneous sends and receives.

The remainder of this paper is organized as follows. In Section 2 we give the lower bound and briefly review techniques for efficient broadcasting. In Section 3 we define a class of trees, which we call "Generalized Fibonacci Trees", and prove certain properties of these trees. Section 4 then shows how these Generalized Fibonacci Trees can be used to obtain an efficient broadcasting algorithm.

2 Preliminaries

We begin by proving a simple lower bound. Many similar lower bounds have been shown for related problems [1, 12].

2.1 Lower Bound

Lemma 2.1 *Broadcasting m packets among n processors requires at least* $m + \lceil \log n \rceil - 1$ *rounds.*

Proof: The broadcaster needs $m - 1$ rounds to send out all but one of the packets. Let l denote the last packet which is sent from the broadcaster. Because in any one round each processor that contains a copy of l can send l to at most one other processor, the number of processors containing l can at most double per round. Therefore, it takes at least $\lceil \log n \rceil$ rounds to distribute l to all of the processors, and the broadcast requires at least $m + \lceil \log n \rceil - 1$ rounds. □

[1]Throughout this paper, $\log n$ will denote the base-2 logarithm of n.

2.2 Broadcast Based on One Spanning Tree

One very simple technique for broadcasting is to create a single tree which spans the n processors and which is rooted at the broadcaster. The broadcaster begins by sending a copy of its first packet to each of its children. The broadcaster then repeats this process for each of the remaining $m - 1$ packets. Whenever a node receives a packet, it sends copies of the packet to each of its children.

Different spanning trees can be used, yielding broadcasting algorithms with different running times. One frequently used spanning tree is the ***binomial tree*** [18], which is naturally derived from a recursive doubling algorithm. When a single spanning binomial tree is used, the broadcast requires $m \lceil \log n \rceil$ rounds. Therefore, except for when $m = 1$ (in which case the binomial tree algorithm is optimal), this algorithm is far from optimal. When a Hamiltonian path is used (which is a special case of a spanning tree), the broadcast requires $m + n - 2$ rounds.

A more efficient algorithm is obtained by using a single almost complete binary tree. An almost complete binary tree of n nodes, where $2^{h-1} - 1 < n \leq 2^h - 1$, is a binary tree derived from a $2^h - 1$ node complete binary tree by removing $2^h - 1 - n$ nodes at the bottom level from right to left. This version of single spanning tree broadcasting algorithm requires $2m + 2 \log n - O(1)$ rounds, which is approximately a factor of 2 greater than the lower bound given above.

2.3 Broadcast Based on Multiple Spanning Trees

A more efficient technique for broadcasting is to create d different d-ary trees, each of which spans the $n - 1$ processors other than the broadcaster. The broadcaster then sends its packets to the roots of these d trees in a round-robin fashion, with each packet being sent to only a single such root. Each processor which is an internal node in one of the spanning trees is responsible for sending copies of the packets which it receives from its parent in that spanning tree to all of its children in that spanning tree. The advantage of this approach is that the root only has to send one copy of each packet, so the time required is proportional to m rather than $2m$.

A challenge with this approach is arranging the spanning trees so that each node has a total of at most d children in all of the spanning trees. This is required so that each node has time to distribute copies of the packets that it has received to each of its children before receiving another round of packets from its parents. Another challenge with this approach is scheduling the communications so that each processor sends at most one packet per round and receives at most one packet per round. Despite these difficulties, this approach has been used with d almost complete d-ary trees to obtain an algorithm which requires $m + 2\log n - O(1)$ rounds [2].

Also, in the special case where n is a power of 2, an algorithm [13] based on the combination of the edge-disjoint spanning trees [16] and the binomial tree of the hypercube can be used. This algorithm requires only $m + \log n - 1$ rounds, thus exactly matching the lower bound proven above. However, no efficient technique is known for extending this algorithm to the case of arbitrary values of n.

3 Generalized Fibonacci Trees

Definition 3.1 A *full* d-ary tree is a d-ary tree in which each node has either 0 or d children.

The following lemma can be proven with a straightforward induction.

Lemma 3.2 *A full d-ary tree T with x internal nodes contains exactly $x(d-1)+1$ leaf nodes.*

Definition 3.3 A tree T is *a treetop of tree T'* if T is a connected subgraph of T' and T contains the root of T'.

Definition 3.4 Given any integers $d \geq 2$ and $t \geq 0$, an order-t d-ary *generalized Fibonacci tree*, denoted $FT_d(t)$, is defined as follows. $FT_d(t)$, where $0 \leq t < d$, has only one node. To obtain $FT_d(t)$, where $t \geq d$, first create a forest consisting of the trees $FT_d(t-1), FT_d(t-2), \cdots, FT_d(t-d)$, then add one additional node (which will be the root of $FT_d(t)$), and finally connect this additional node to the root of each of the trees $FT_d(t-1), FT_d(t-2), \cdots, FT_d(t-d)$.

Figure 1 shows an example of $FT_2(6)$ and Figure 2 shows an example of $FT_3(6)$. It can be easily seen by induction that an $FT_d(t)$ is a *full* d-ary tree. We now define a labeling of the nodes in an $FT_d(t)$.

Definition 3.5 The *base-x labeling* of a generalized Fibonacci tree $FT_d(t)$ with root node r assigns the label x to r and for $1 \leq i \leq d$, performs a base-$(x+i)$ labeling of the subtree $FT_d(t-i)$ rooted at the i-th child of r. The *base-x labeling* of a tree T which is a treetop of a generalized Fibonacci tree $FT_d(t)$ assigns the same label to each node in T as its label in $FT_d(t)$.

Figures 1 and 2 show examples of base-0 labelings. Note that a node in a d-ary tree which receives a packet at round t can send copies of the packet to its children at rounds $t+1$ through $t+d$. Therefore, a base-x labeling of a d-ary tree defines a valid scheduling for broadcasting one packet to all nodes on the tree. In particular, the label of a node in a base-x labeled tree represents the rounds at which it first receives a packet, assuming that the root obtains the packet at round x. The largest label in the tree therefore gives the number of rounds required to broadcast one packet in the tree according to this scheduling. Note that the largest label in a *base-x* labeling of a Fibonacci tree $FT_d(t)$ is $t + x$.

Notice that the base-x labeling has an additional property which facilitates the scheduling for pipelining a new packet every d rounds. Specifically, the labels of the d children of each internal node are all distinct modulo d. This means that the root can receive new packets from some external nodes at rounds $x, x+d, x+2d, \cdots$, in which case each node labeled i will receive new packets at rounds $i, i+d, i+2d, \cdots$.

Definition 3.6 Let $F_d(t)$ be the number of nodes in the order-t d-ary Fibonacci tree $FT_d(t)$.

From Definition 3.4, one can derive the following recursion

$$F_d(t) = \begin{cases} 1, & \text{if } 0 \leq t < d, \\ 1 + \sum_{i=1}^{d} F_d(t-i), & \text{if } t \geq d. \end{cases}$$

Note that this recursion is similar to the recursion which defines the Fibonacci numbers, thus explaining our use of the term "Fibonacci tree". The following theorem gives a lower bound for $F_d(t)$. The proof is given in Appendix A.

Theorem 3.7 *For any integer $d \geq 2$ and for any integer $t \geq d+2$, $F_d(t) \geq (2 - 2^{-d+1})^{t-d-4}$.*

Given n and d, we are actually interested in finding the smallest integer t such that $FT_d(t)$ has at least n nodes. Formally, we wish to find $f_d(n)$ defined as follows.

Definition 3.8 Let $f_d(n) \overset{\text{def}}{=} \min\{t \mid F_d(t) \geq n\}$.

The following theorem gives an upper bound for $f_d(t)$. The proof is given in Appendix B.

Theorem 3.9 *For any integer $d \geq 2$ and for any $n \geq 1$, $f_d(n) \leq \log n + (3\log n)/(2^d - 3) + d + 5$.*

4 The Broadcast Algorithm

4.1 Overview

The idea is to use d d-ary spanning trees, where each tree is a treetop of a d-ary Fibonacci tree. (We will later choose $d \approx \log\log n$ for sufficiently large n to yield a good upper bound.) There are different cases based on the value of $n \bmod d^2$. The simplest case, in which $n \bmod d^2 = d+1$, will be presented first. Throughout this section we will assume that d is odd.

4.2 Broadcast among n processors when $n \bmod d^2 = d + 1$

Definition 4.1 Two nodes in a d-ary tree are in the same ***sibling set*** if they have the same parent. A sibling set consisting entirely of leaf nodes is called a ***leaf sibling set***.

Let $s = (n-1)/d$ and note that $s \bmod d = 1$. Remove the broadcaster and partition the $n-1$ remaining processors into d groups of size s, and number the groups 0 through $d-1$. Next, build a tree T_0 which spans the processors in group 0 and has the following properties:

1. T_0 is a treetop of $FT_d(t)$, where $t = f_d(s)$.
2. T_0 is a full d-ary tree.

Note that T_0 can be obtained from $FT_d(t)$ by successively removing leaf sibling sets until s nodes remain. Next, for each i, $1 \le i \le d-1$, create a tree T_i which is isomorphic to T_0 and which spans the processors in group i, and let r_i denote the root of T_i. Figure 3 shows an example of T_0, T_1 and T_2 when $n = 22$ and $d = 3$.

We are now ready to extend the T_i's so that each of them spans all of the $n-1$ processors other than the broadcaster. More specifically, we will extend each tree T_i to obtain a new tree $\hat{T}_i$ by adding d children to each leaf in T_i. Let $x = (s-1)/d$ and note that it follows from Lemma 3.2 that each tree T_i has exactly x internal nodes and $x(d-1)+1 = (s-1)(d-1)/d+1$ leaves. Therefore, adding d children to each leaf in T_i will add exactly $(s-1)(d-1)+d = s(d-1)+1$ nodes, and each tree $\hat{T}_i$ will have exactly $s + s(d-1)+1 = n$ nodes. However, our goal is to have each $\hat{T}_i$ span the $n-1$ processors other than the broadcaster, so each $\hat{T}_i$ has 1 extra node. In order to accommodate this extra node, we will pretend that we have 1 additional processor, called the ***virtual processor*** and denoted v, which is not in any of the T_i's but which will be a leaf in all of the $\hat{T}_i$'s.

The $\hat{T}_i$'s are created as follows. For each i where $0 \le i \le d-1$, pair all but one of the leaves in T_i with the sibling sets in the $d-1$ trees T_j where $j \ne i$. Each leaf in T_i and each sibling set in T_j where $j \ne i$ must appear in exactly one of these pairs, but otherwise the pairing is arbitrary. Next, take the one remaining leaf in T_i and pair it with the set $R_i = \{r_j | j \ne i\} \cup \{v\}$. The tree $\hat{T}_i$ is obtained from the tree T_i by taking the d nodes that are paired with each leaf node in T_i and making them children of that leaf node. Figure 4 shows an example of $\hat{T}_0$ when $n = 22$ and $d = 3$.

At this point we have a single broadcaster and a forest of d trees, T_0 through T_{d-1}, which collectively span the $n-1$ processors other than the broadcaster. Also, we have a virtual processor v which is not in any of the T_i's, and extensions of the T_i's, $\hat{T}_0$ through $\hat{T}_{d-1}$, each of which spans the n processors (including v) other than the broadcaster. All that remains is to define a schedule for broadcasting the packets along the $\hat{T}_i$'s. We will define this schedule by labeling each of the $\hat{T}_i$'s. Note that each processor appears in all d of the $\hat{T}_i$'s, so it will receive d different labels. We will let $l_i(p)$ denote processor p's label in tree $\hat{T}_i$. These labels are defined as follows.

Definition 4.2 Let p_i denote an arbitrary processor in T_i and for $0 \le j \le d-1$, let q_j denote p_i's parent in $\hat{T}_j$ (if such a parent exists). Let $l_i(p_i)$ be the label assigned to p_i by a base-i labeling of T_i. For each j where $0 \le j \le d-1$ and $i \ne j$, let $l_j(p_i)$ be the smallest integer greater than $l_j(q_j)$ such that $l_j(p_i) + j \equiv l_i(p_i) + i \pmod d$. Finally, if q_j is v's parent in $\hat{T}_j$, let $l_j(v)$ be the smallest integer greater than $l_j(q_j)$ such that $l_j(v) \equiv j \pmod d$.

The following lemmas show that these labels can be used to schedule the broadcasting of the packets along the $\hat{T}_i$'s.

Lemma 4.3 *If processor q is the parent of processor p in tree $\hat{T}_i$, where $0 \le i \le d-1$, then $l_i(q) < l_i(p)$.*

Proof: If p is an internal node of $\hat{T}_i$, then both p and q are in T_i and the lemma follows from the fact that a base-i labeling of T_i assigns smaller labels to parents than to children. If p is a leaf node in $\hat{T}_i$, then it is clear from the construction given above that $l_i(q) < l_i(p)$. □

Lemma 4.4 *If processor q is an internal node in tree $\hat{T}_k$, $0 \le k \le d-1$, and processors p_i and p_j are distinct children of q in $\hat{T}_k$, then $l_k(p_i) \not\equiv l_k(p_j) \pmod d$.*

Proof: If p_i and p_j are internal nodes of $\hat{T}_k$, then both p_i and p_j are in T_k and the lemma follows from the fact that a base-k labeling of T_k assigns distinct labels, modulo d, to siblings. Otherwise, if p_i and p_j are siblings in some tree T_h, then $l_k(p_i)+k \equiv l_h(p_i)+h \pmod d$ and $l_k(p_j)+k \equiv l_h(p_j)+h \pmod d$, so $l_k(p_i) + l_h(p_j) \equiv l_h(p_i) + l_k(p_j) \pmod d$. However, $l_h(p_j) \not\equiv l_h(p_i) \pmod d$ so it follows that $l_k(p_j) \not\equiv l_k(p_i) \pmod d$.

Otherwise, assume without loss of generality that $p_i = r_i$ and $i \ne k$. If $p_j = r_j$ where $j \ne k$ and $j \ne i$, then $l_k(p_i) + k \equiv 2i \pmod d$ and $l_k(p_j) + k \equiv 2j \pmod d$, which implies that $l_k(p_i) \not\equiv l_k(p_j) \pmod d$ (because d is odd). Finally, if $p_j = v$, then $l_k(p_i)+k \equiv 2i \pmod d$ and $l_k(p_j) \equiv k \pmod d$, which implies that $l_k(p_j) + k \equiv 2k \pmod d$ and $l_k(p_i) \not\equiv l_k(p_j) \pmod d$ (because d is odd). □

Lemma 4.5 *Let p be any processor (including v) other than the broadcaster, and let i and j be distinct integers where $0 \le i \le d-1$ and $0 \le j \le d-1$. Then $l_i(p) \not\equiv l_j(p) \pmod d$.*

Proof: If $p = v$ then $l_i(p) \equiv i \pmod{d}$ and $l_j(p) \equiv j \pmod{d}$, so $l_i(p) \not\equiv l_j(p) \pmod{d}$. Otherwise, p is in some tree T_h where $0 \leq h \leq d-1$. In this case, $l_i(p)+i \equiv l_h(p)+h \pmod{d}$ and $l_j(p)+j \equiv l_h(p)+h \pmod{d}$, which implies that $l_i(p) \not\equiv l_j(p) \pmod{d}$. □

Lemma 4.6 *It is possible to broadcast* m *packets among* n *processors in* $m + f_d(\frac{n-1}{d}) + d - 1$ *rounds, assuming* $n \bmod d^2 = d+1$ *and* $d \geq 3$ *is an odd integer.*

Proof: The broadcaster can send out the m packets in a round-robin fashion to each of the $\hat{T}_i$'s. Each processor p receives its first packet in tree $\hat{T}_i$ at round $l_i(p)$ and successive packets in tree $\hat{T}_i$ at rounds $l_i(p)+d, l_i(p)+2d, \cdots$. Whenever a processor p that is an internal node in $\hat{T}_i$ receives a packet it sends copies of the packet to its d children in $\hat{T}_i$, in increasing order according to the children's labels in $\hat{T}_i$. From the previous lemmas, and the fact that each processor is an internal node in at most one of the $\hat{T}_i$'s, it follows that this broadcasting schedule requires each processor to receive at most one packet per round and to send at most one packet per round. Finally, note that at most $f_d(\frac{n-1}{d}) + d$ rounds are required to pass any single packet from the broadcaster to all of the remaining processors. Therefore, the broadcast completes in $m + f_d(\frac{n-1}{d}) + d - 1$ rounds. □

4.3 Broadcast among n processors when $n \bmod d = 1$

We will now show how the construction given in the previous subsection can be extended to handle the case where $n \bmod d = 1$. Let $n = n' + \beta d$ where n' and β are integers, $0 \leq \beta \leq d-1$, and $n' \bmod d^2 = d+1$. Note that the construction given in the previous subsection can be used to perform the broadcast among n' of the processors. Therefore, all that is required is that we insert the βd additional processors into the broadcast without greatly increasing the time requirement of the algorithm. In fact, we will be able to insert these βd additional processors while only increasing the number of rounds required by one.

The idea is to insert the i-th additional processor into the middle of the edge entering the i-th leaf of $\hat{T}_j$, where $0 \leq j \leq d-1$ (note that n must be sufficiently large with respect to d, so that $\hat{T}_j$ will have at least βd leaves). Figure 5 shows an example of the modified $\hat{T}_0$ from Figure 4 when $n = 28$ and $d = 3$. Thus each of the additional processors will appear in all d of the $\hat{T}_i$'s and will have a single child in each of them. In each of these $\hat{T}_i$'s, if an additional processor receives a packet from its parent at round t it will simply pass this packet on to its (only) child at round $t+1$.

There are two complications regarding the scheduling of the broadcast after adding these additional nodes. First, the first βd leaf nodes in each $\hat{T}_i$ now have to wait one additional round to receive each packet (because the additional processor that was inserted above it introduces a delay of one round). However, note that the number of additional processors is a multiple of d, so either all of the processors in a leaf sibling set in a tree $\hat{T}_i$ have their labels increased by one or none of them have their labels increased. As a result, all of the processors in a leaf sibling set can be assigned labels that are distinct modulo d and we are able to use the rules given in Definition 4.2 to label these leaf nodes.

Second, we must show that the additional processors neither receive more than one packet in a round nor send more than one packet in a round. However, this is trivial since each additional processor is inserted in the same location in each of the $\hat{T}_i$'s and the labels associated with any given position in each of the $\hat{T}_i$'s are all distinct modulo d.

Finally, it is clear that the insertion of the additional processors only increases the number of rounds required by at most one. Combining these observations yields the following lemma.

Lemma 4.7 *It is possible to broadcast* m *packets among* n *processors in* $m + f_d(\frac{n-1}{d}) + d$ *rounds, assuming* $n \geq d^2 + d + 1$, $n \bmod d = 1$, *and* $d \geq 3$ *is an odd integer.*

4.4 Broadcast among n processors where n is arbitrary

The construction given in the previous subsection can be extended to handle the case of arbitrary n. Let $n = n' + \alpha$ where n' and α are integers, $0 \leq \alpha \leq d-1$ and $n' \bmod d = 1$. Note that the previous construction can be used to perform the broadcast among n' of the processors. Therefore, all that is required is that we insert the α additional processors into the broadcast in an efficient manner. We will do this by simply creating a linear array of these α additional processors. The first processor in this linear array will play the role of the virtual processor, v, described above. Whenever a processor in this linear array (other than the last one) receives a packet it passes a copy of the packet on to the next processor in the linear array one round later. Because the virtual processor v receives all of the broadcast packets in the above algorithm, the additional processors in this linear array will also receive all of the broadcast packets. Finally, note that the addition of this linear array increases the number of rounds required by at most $d-2$, as the linear array has at most $d-1$ processors and the time required to broadcast the packets to the virtual processor was included in the analysis of the previous algorithm. As a result, we have the following lemma.

Lemma 4.8 *It is possible to broadcast* m *packets among* n *processors in* $m + f_d(\frac{n-1}{d}) + 2d - 2$ *rounds, assuming* $n \geq d^2 + d + 1$ *and* $d \geq 3$ *is an odd integer.*

We are now prepared to prove the main result of this paper.

Theorem 4.9 *For all sufficiently large n, it is possible to broadcast m packets among n processors in $m + \log n + 3\log\log n + 15$ rounds.*

Proof: Let d be the smallest odd integer greater than or equal to $\log(3+\log n)$ and note that $d < 3+\log\log n$ for all sufficiently large n. It follows from Lemma 4.8 that the algorithm described above requires at most $m+f_d(n)+2d-2$ rounds. From Theorem 3.9, $f_d(n) \le \log n+(3\log n)/(2^d-3)+d+5 < \log n+\log\log n+11$. Therefore, the algorithm requires at most $m+\log n+3\log\log n+15$ rounds. □

Although setting $d \approx \log\log n$ gives the best asymptotic bounds, it should be noted that even small constant values of d give very good bounds. For example, when $d = 7$ Theorem 3.9 yields $f_7(n) \le 1.024\log n+12$ and Lemma 4.8 shows that the broadcasting algorithm requires at most $m + 1.024\log n + 24$ rounds.

A Proof of Theorem 3.7

The proof relies on the following definition and theorem, which were given by Capocelli *et al.* [6].

Definition A.1

$$G_d(t) = \begin{cases} 0, & \text{if } t = 0, \\ 1, & \text{if } t = 1, \\ 2^{t-2}, & \text{if } 2 \le t < d, \\ \sum_{i=1}^{d} G_d(t-i), & \text{if } t \ge d. \end{cases}$$

Theorem A.2 (Capocelli *et al.*) *For any integers d and t where $d \ge 2$ and $t \ge 0$,*

$$G_d(t) = \left[\frac{x-1}{(d+1)x^d - 2d} x^{t-1} + 0.5 \right]$$

where x is a constant in the range $2-2^{-d+1} < x < 2-2^{-d}$.

Lemma A.3 *For any integers d and t where $d \ge 2$ and $t \ge d-2$, $F_d(t) \ge G_d(t-d+3)$.*

Proof: It is straightforward to verify that $F_d(t) = 2^{t-d}d + 1 \ge 2^{t-d+1}$ if $d \le t < 2d$. Also, note that $F_d(d-2) = 1 = G_d(1)$, that $F_d(d-1) = 1 = G_d(2)$, and that $G_d(d) = 2^{d-2}$. Therefore, it follows that for any t where $d-2 \le t \le 2d-3$, $F_d(t) \ge G_d(t-d+3)$. It then follows by induction that for any t where $t \ge 2d-2$, $F_d(t) \ge G_d(t-d+3)$. □

Theorem A.2 and Lemma A.3 can then be used to prove Theorem 3.7.

Proof: From Theorem A.2,

$$\begin{aligned} G_d(t) &\ge \frac{1-2^{-d+1}}{(d+1)(2-2^{-d})^d - 2d}(2-2^{-d+1})^{t-1} + 0.5 \\ &\ge (\frac{1}{11})(2-2^{-d+1})^{t-1} \\ &\ge (2-2^{-d+1})^{t-7} \end{aligned}$$

when $d \ge 2$ and $t \ge 5$. Therefore, when $d \ge 2$ and $t \ge d-2$, it follows from Lemma A.3 that $F_d(t) \ge G_d(t-d+3) \ge (2-2^{-d+1})^{t-d-4}$. □

B Proof of Theorem 3.9

Proof: Let $y = \lfloor \log n + (3\log n)/(2^d-3) + d + 5 \rfloor$ and note that

$$\begin{aligned} y &\ge \log n + (3\log n)/(2^d-3) + d + 4 \\ &= 2^d(\log n)/(2^d-3)) + d + 4 \\ &\ge 2^d(\log n)/(2^d - \log e - 1)) + d + 4. \end{aligned}$$

Using a Taylor Series expansion, we have

$$\log(2-a) = 1 - (\log e)\sum_{i=1}^{\infty}(a/2)^i/i$$

so

$$\begin{aligned} \log(2-2^{-d+1}) &= 1 - (\log e)\sum_{i=1}^{\infty} 2^{-di}/i \\ &\ge 1 - (\log e)\sum_{i=1}^{\infty} 2^{-di} \\ &= 1 - (\log e)/(2^d-1) \end{aligned}$$

and

$$\begin{aligned} 1/\log(2-2^{-d+1}) &\le (2^d-1)/(2^d-\log e-1) \\ &\le 2^d/(2^d-\log e-1). \end{aligned}$$

Therefore

$$y \ge (\log n)/\log(2-2^{-d+1}) + d + 4$$

and

$$y - d - 4 \ge (\log n)/\log(2-2^{-d+1})$$

and

$$(y-d-4)\log(2-2^{-d+1}) \ge \log n$$

and

$$n \le (2-2^{-d+1})^{y-d-4} \le F_d(y)$$

from Theorem 3.7. Thus $F_d(y) \ge n$ which implies that $f_d(n) \le y \le \log n + (3\log n)/(2^d-3) + d + 5$. □

References

[1] Noga Alon, Amnon Barak, and Udi Manber. On disseminating information reliably without broadcasting. In *International Conference on Distributed Computing Systems*, pages 74–81, 1987.

[2] Amotz Bar-Noy and Shlomo Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. Technical report, IBM RC 17429, November 1991.

[3] C. Baru, S. Padmanbhan, Q. Stout, and B. Wagar. A comparison of join algorithms for hypercubes. In *The Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 469–474, 1989.

[4] John Beetem, Monty Denneau, and Don Weingarten. The GF-11 supercomputer. In *International Symposium on Computer Architecture*, pages 108–115, 1985.

[5] J. Bruck, R. Cypher, C.T. Ho, and E. Upfal. A proposal for the topology and wiring of the Green Vulcan. Technical Report RJ 8015, IBM Almaden Research Center, March 1991.

[6] R. Capocelli, G. Cerbone, P. Cull, and J. Holloway. Fibonacci facts and formulas. In *Workshop on Sequences*, pages 123–137, Positano, Italy, 1988.

[7] P.R. Cappello. Gaussian elimination on a hypercube automaton. *J. Parallel Distributed Comput.*, 4:288–308, 1987.

[8] E. Cockayne and A. Thomason. Optimal multi-message broadcasting in complete graphs. In *Eleventh SE Conference on Combinatorics, Graph Theory and Computing*, pages 181–199, 1980.

[9] Arthur M. Farley. Broadcast time in communication networks. *SIAM J. Appl. Math.*, 39(2):385–390, October 1980.

[10] Dennis Gannon and John Van Rosendale. On the impact of communication complexity in the design of parallel numerical algorithms. *IEEE Trans. Computers*, 33(12):1180–1194, December 1984.

[11] G.A. Geist and Michael T. Heath. Matrix factorization on a hypercube multiprocessor. In *Hypercube Multiprocessors 1986*, pages 161–180. SIAM, 1987. Edited by Michael T. Heath.

[12] S. M. Hedetniemi, S. T. Hedetniemi and A. L. Liestman, A survey of gossiping and broadcasting in communication networks, *Networks*, Vol. 18, pp. 319-349, 1988.

[13] Ching-Tien Ho. Optimal broadcasting on SIMD hypercubes without indirect addressing capability. *Journal on Parallel and Distributed Computing*, 13(2):246–255, October 1991.

[14] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987.

[15] S. Lennart Johnsson and Ching-Tien Ho. *Matrix Multiplication on Boolean Cubes Using Generic Communication Primitives*, pages 108–156. SIAM, 1989. In Parallel Processing and Medium-Scale Multiprocessors, edited by Arthur Wouk.

[16] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.

[17] Cleve Moler. Matrix computation on distributed memory multiprocessors. In *Hypercube Multiprocessors 1986*, pages 181–195. SIAM, 1987. Edited by Michael T. Heath.

[18] H. Sullivan and T. R. Bashkow. A large scale homogeneous, fully distributed parallel machine. In *Proceedings of the fourth Annual Symposium on Computer Architecture*, pages 105–124, March 1977.

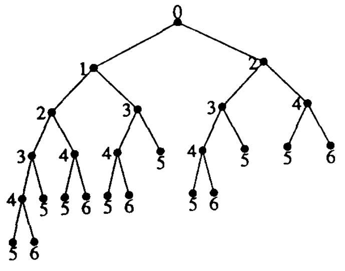

Figure 1: An example of the order-6 binary Fibonacci tree, $FT_2(6)$.

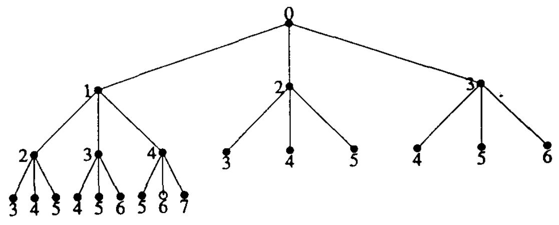

Figure 4: An example of $\hat{T}_0$ for $n = 22$ and $d = 3$. The empty circle denote the virtual processor.

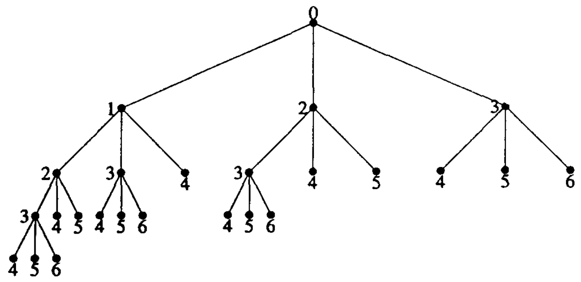

Figure 2: An example of the order-6 ternary Fibonacci tree, $FT_3(6)$.

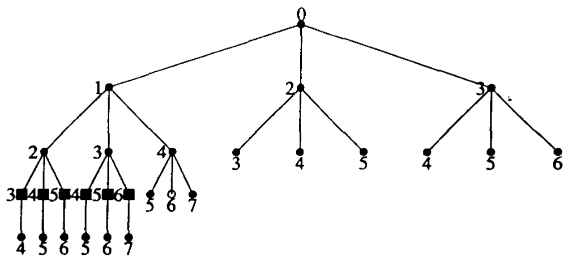

Figure 5: An example of modified $\hat{T}_0$ for $n = 28$. Nodes added to the previous figure are denoted by squares.

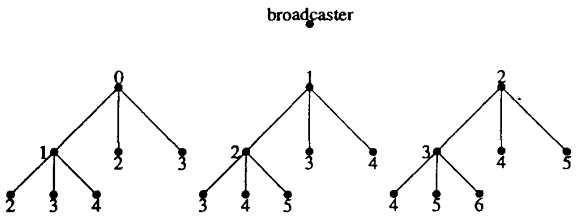

Figure 3: An example of T_0 through T_2 for $n = 22$ and $d = 3$.

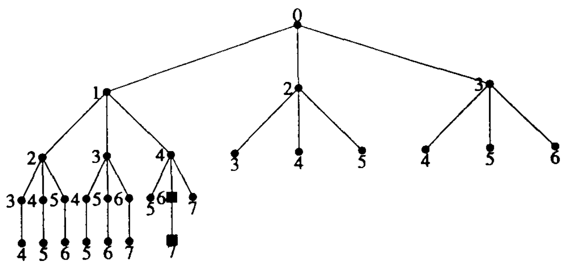

Figure 6: An example of modified $\hat{T}_0$ for $n = 30$. Nodes added to the previous figure are denoted by squares.