Della Copies to: N. Tinkelpaugh B. Andrews M. Jones (LU) C. Robinson (Link) W. Heimerdinger from Jim Masten NUGENT

Correspondence

Incomplete Hypercubes

HOWARD P. KATSEFF

Abstract—**A widely used topology for the interconnection of computing nodes in multiprocessor systems is the binary hypercube. This topology has the advantage of simple, deadlock-free routing and broadcast algorithms. Since a k-dimensional hypercube has 2^k vertices, these systems are restricted to having exactly 2^k computing nodes. Because system sizes must be a power of two, there are large gaps in the sizes of systems that can be built with hypercubes.**

This paper presents routing and broadcast algorithms for hypercubes that are missing certain of their nodes, called *incomplete hypercubes*. Unlike hypercubes, incomplete hypercubes can be used to interconnect systems with any number of processors. The routing and broadcast algorithms for incomplete hypercubes are also simple and deadlock-free.

Index Terms—**Broadcast, deadlock, hypercube, interconnection topology, multiprocessor, routing.**

I. Introduction

In 1963, Squire and Palais proposed a message-passing multiprocessor computer system with 2^k processing nodes, in which each node is placed at the vertex of a k-dimensional hypercube and the edges of the hypercube are links between the processors [1]. A later paper [2] described simple algorithms for routing and for broadcasting messages between arbitrary nodes of the hypercube. It has been shown that the routing algorithms pass messages without deadlock [3], and have a worst case path length of k. Several successful multiprocessor computer systems have been built using the hypercube topology, including the Cosmic Cube [4] and the iPSC system [5].

A problem with the hypercube topology is that the number of nodes in a system must be a power of 2. In practical terms, this is a severe restriction on the sizes of systems that can be built. This restriction can be overcome by using an *incomplete hypercube,* a hypercube missing certain of its nodes. Unlike hypercubes, incomplete hypercubes can be constructed with any number of nodes. The algorithms presented here for routing and broadcasting on an incomplete hypercube are nearly as simple as those for the hypercube. Furthermore, the routing algorithm for the incomplete hypercube has similar performance characteristics as the routing algorithm for the hypercube. For example, the algorithm passes messages without deadlock and for an incomplete hypercube with n nodes, has a worst case path length of $\lceil \log_2 n \rceil$.

II. Nomenclature

We refer to the processors of a multiprocessor computer system as *nodes,* and the communications links connecting processors as *links.* The processors communicate by passing *messages* on these links. A message sent from one processor to another may be routed through intermediate processors. The ordered list of processors visited by the message is a *path.* The *length* of a path is the number of links in the path.

We use *hypercube* to refer to a Boolean hypercube, also known as a Boolean n-cube or binary cube. A hypercube with n nodes is constructed by numbering the nodes from 0 to $n - 1$ and linking each pair of nodes whose binary representations differ by exactly one bit. If n is a power of two, then this scheme describes a hypercube. Otherwise, it describes an *incomplete* hypercube. We sometimes refer to a hypercube as a *complete* hypercube for emphasis. A link in an incomplete hypercube is said to *exist* if the hypercube contains that link. Note that in a complete hypercube with n nodes, each node has $\log_2 n$ links that exist.

We define the *relative address* of two nodes a and b as the bitwise Exclusive-OR of their node numbers, $a \oplus b$, and define the *distance* between two nodes as the number of ones in their relative address. A link has *link number i* if it connects two nodes whose numbers differ only in the ith bit position (starting with the least significant bit as bit 0). For example, the link between nodes 101 and 001 is referred to as link 2 because the two nodes differ in bit 2. Note that the same link number is obtained when computed from either end of the link.

For convenience, n is always used to refer to the number of nodes in an incomplete hypercube. Finally, log2(n) is defined as $\lceil \log_2 n \rceil$.

III. Routing Algorithm

We first review the routing algorithm for hypercubes due to Sullivan and Bashkow [2]. The algorithm is a procedure that is executed by the originating node and by every node in the path to the destination. It is described as Algorithm 1.

Algorithm 1 (send or forward a message from node *src* to node *dest* in a complete hypercube [2]):

```
if (src = = dest)
        Send message to local processor.
else
        Compute reladdr ← src ⊕ dest.
        Starting with the most significant bit of reladdr:
                let i be the bit number of the first 1 in reladdr.
        Send the message on link i.
```

Fig. 1 shows how a message is routed from node 011 to node 100 in a cube with eight nodes. It is easy to see that this algorithm correctly routes messages because each step in the routing reduces the distance to the destination by 1.

Surprisingly, a simple variant of this algorithm, described as Algorithm 2, works for incomplete hypercubes.

Algorithm 2 (to send or forward a message from node *src* to node *dest* in an incomplete hypercube):

```
if (src = = dest)
        Send message to local processor.
else
        Compute reladdr ← src ⊕ dest.
        Starting with the most significant bit of reladdr:
                let i be the bit number of the first 1 in reladdr.
                where link i exists from node src.
        Send the message on link i.
```

Fig. 2 shows how this algorithm routes a message from node 011 to node 100 in a cube with seven nodes.

If we could show that Algorithm 2 always succeeds in finding a link i on which to forward the message, then it should be clear that the algorithm correctly routes messages because each step in the routing reduces the distance to the destination by 1. We do this in Theorem 1.

Theorem 1: Given any two distinct nodes *src* and *dest* in an incomplete hypercube, the bit number of at least one bit in the relative

Manuscript received July 15, 1985; revised December 11, 1986.
The author is with AT&T Bell Laboratories, Holmdel, NJ 07733.
IEEE Log Number 8716139.

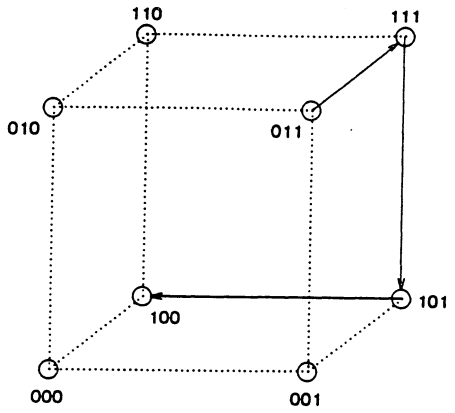

Fig. 1. Routing from 011 to 100 in a cube with eight nodes.

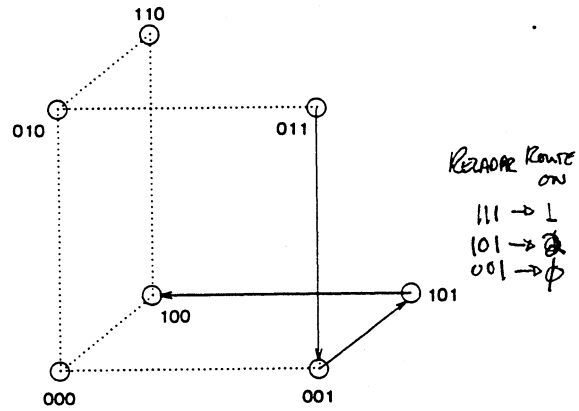

Fig. 2. Routing from 011 to 100 in an incomplete cube with seven nodes.

address of *src* and *dest*, *src* ⊕ *dest*, is the number of a link from node *src*.

Proof: Let *reladdr* be the relative address of *src* and *dest* and let n be the number of nodes in the incomplete hypercube. We say that *reladdr specifies* bit i if bit i of *reladdr* is set to one. We partition the possible values of *src* ∧ *reladdr* into four cases, and show that the theorem holds in each case.

Case 1: at least one of the bits specified by *reladdr* was a one in *src*, i.e.,

src ∧ *reladdr* is nonzero.

Let k be the number of the most significant bit in *reladdr* that was a one in *src*. Link k from *src* goes to a node with a smaller number than k, because bit k in *src* is 1. Since an incomplete hypercube contains every node number up to $n - 1$, link k must exist because it leads to a node which is in the incomplete hypercube.

Case 2: all of the bits specified by reladdr are zero in *src* AND *reladdr* has at least 2 bits equal to 1, i.e.,

src ∧ *reladdr* is zero AND *reladdr* has at least two bits equal to 1.

In this case, *dest* has all these bits 1. The link specified by the most significant bit of *reladdr* is to a node with number less than *dest* because the specified bits other than the most significant bit are 0 in the node at the end of this link and are 1 in *dest*. So, this link exists.

Case 3: all of the bits specified by *reladdr* are zero in *src* AND *reladdr* has exactly one bit equal to 1, i.e.,

src ∧ *reladdr* is zero AND *reladdr* has exactly one bit equal to 1

In this case the link from *src* to *dest* is traversed.

Case 4: all of the bits specified by *reladdr* are zero in *src* AND *reladdr* has no bits equal to 1, i.e.,

src ∧ *reladdr* is zero AND *reladdr* is zero.

This is the case of *src* and *dest* being the same node. ■

Since no two nodes differ in distance by more than log2(n), and Algorithm 2 reduces the distance to the source by one each time the message is routed, the maximum path length determined by this algorithm is log2(n).

IV. Deadlock

A useful property of a message routing algorithm is that it does not deadlock. DeBenedictis has shown that Sullivan and Bashkow's routing algorithm for complete hypercubes does not deadlock [3]. In this section we show that Algorithm 2, the algorithm for routing in incomplete hypercubes, does not deadlock.

We first examine the issue of buffering in the links between nodes. If the links are not buffered, then a node sending a message blocks until the receiving node reads the message. If there is buffering of size k on a link, then the sender does not block until k messages are sent on that link that have not been read. It is easy to see that some buffering is necessary to prevent deadlock. Suppose that there is no buffering and each node starts out by sending some message over each of its links. Then each node is blocked sending its messages until the node at the other end of the link reads it. But all nodes are busy sending the message from the local processor, so the messages are not read, causing deadlock.

It is therefore necessary to assume that there is buffering on every link. It is sufficient that each link is able to independently buffer a single message in each direction to show that deadlock does not occur. Algorithm 3, below, is the routing algorithm for incomplete hypercubes modified to explicitly indicate buffering on the links. An instance of this algorithm is connected to each input link of a node. When the algorithm blocks waiting for some condition, it stops running and waits for the desired condition to occur. The other instances of the algorithm at the other nodes continue to process incoming data.

Algorithm 3 (send or forward a message from node *src* to node *dest* in an incomplete hypercube, with explicit buffering):

```
if (src = = dest)
        Send message to local processor
else
        Compute reladdr ← src ⊕ dest.
        Starting with the most significant bit of reladdr:
                let i be the bit number of the first 1 in reladdr
                where link i exists from node src.
        If the previous message that was sent on link i has not been
                read,
                wait until it is read, then send the message on link i.
```

Lemma 1 shows that if there is deadlock, then there must be a cycle among the deadlocked nodes.

Lemma 1: Given an incomplete hypercube that is deadlocked, there is a cycle of nodes $a_0, a_1, \cdots, a_{m-1}$ (in which a node may be included more than once) where each node a_k is blocked sending a message that has arrived on the link from node $a_{k-1 \bmod m}$ and is to be sent on the link to node $a_{k+1 \bmod m}$. Note that the message may have originated from a node other than $a_{k-1 \bmod m}$ and have destination other than $a_{k+1 \bmod m}$.

Proof: Given the deadlocked incomplete hypercube, we construct the cycle of nodes. Pick some node that is deadlocked sending a message and call it b_0. The message is waiting to be sent to some other node, say b_1. It cannot be sent because the message previously sent on this link (that was sent from b_0 to b_1) is blocked. This message is blocked because it is waiting to be sent to some other node, say b_2. It cannot be sent because the message previously sent on this link (from b_1 to b_2) is blocked.

Continue in this manner until a cycle occurs: until a message is sent to some node that has already been visited, say node b_k, and the message is blocked because it is to be sent to the node following b_k in the cycle: node b_{k+1}. This termination condition must eventually occur because there only a finite number of nodes in the incomplete hypercube.

Suppose that this procedure terminates after j iterations, so that b_j is the same node as b_k. Then define the sequence a of length $j - k$ as

$$a_i = b_{i+k}, \qquad \text{for } 0 \le i < j-k.$$

This is the cycle desired to prove the lemma. ■

To prove that Algorithm 3 does not exhibit deadlock, we show that the condition described in Lemma 1 cannot occur when using this routing algorithm.

Theorem 2: Algorithm 3 does not exhibit deadlock.

Proof: For the sake of contradiction, assume that there is deadlock. Then, there is a cycle of nodes $a_0, a_1, \cdots, a_{m-1}$, as described in Lemma 1. Let l_i be the link number for the message leaving node a_i. Note that each link number appears an even number of times in the cycle (otherwise, a cycle would be impossible because each node in the cycle differs from the previous one in exactly 1 bit). Let $g = \max\{l_i : 0 \le i < m\}$. Then there is a j where $l_j = g$ and the link l_j indicates a transition of bit g in the address from 1 to 0 (because each appearance of link number g in the cycle indicates alternating transitions of bit g).

Consider the message that is blocked at transmission from node a_j. This message came from node $a_{j-1 \bmod m}$ and is destined for $a_{j+1 \bmod m}$ or beyond. These two nodes differ in two bits: bit g and some other bit h. Link h connects $a_{j-1 \bmod m}$ to a_j and link g connects a_j to $a_{j+1 \bmod m}$. Since g is the maximum link number, h must be less than g. We also know that node $a_{j-1 \bmod m} \oplus 2^g$ is included in this incomplete hypercube (because this node has bit g zero so it is less than a_j which has bit g set to one), so there is a link from $a_{j-1 \bmod m}$ to $a_{j-1 \bmod m} \oplus 2^g$. Algorithm 3 should send the message on this link from node $a_{j-1 \bmod m}$ in preference to sending it on link h. But, we assumed that the algorithm is deadlocked and the message was sent from node $a_{j-1 \bmod m}$ to node a_j on link h. This gives the desired contradiction. ■

V. Broadcast Algorithm

Sullivan and Bashkow have devised an algorithm for broadcasting a message to all other processors in a hypercube [2]. This algorithm sends the message to all other nodes in $\log2(n)$ steps and sends the message nonredundantly, meaning that each broadcast message is sent to every node exactly once. The algorithm works by sending a *weight* along with each message that indicates how the algorithm should continue broadcasting the message from the receiving node. Algorithm 4 is Sullivan and Bashkow's algorithm. It is somewhat different than the algorithm presented in their paper [2] because we number links differently than they do. Fig. 3 shows how a message is broadcast from node 011 in a cube with eight nodes. The figure shows a tree where the nodes of the tree are nodes of the hypercube and the arcs in the tree are the links over which messages are broadcast by the algorithm. The root of the tree is the node at which the broadcast originates.

Algorithm 4 (forward (or originate) a broadcast message in a complete hypercube [2]):

Start the algorithm at any node with *weight* set to $\log_2(n)$.
For each link l from this node with l less than *weight:*
    send the message on link l with a *weight* of l.

A modified version of this algorithm that ignores missing links in an incomplete hypercube successfully broadcasts messages when starting at node 0 because broadcast messages are always sent from a smaller node number to a larger one. However, it does not work for broadcasting from arbitrary nodes. For instance, in an incomplete hypercube with three nodes, a broadcast message sent from node 01 would never arrive at node 10.

To motivate the broadcasting algorithm for incomplete hypercubes, we first describe a version of Algorithm 4 in which the integer *weight* is replaced by a $\log2(n)$-bit Boolean array, *travel*. The elements of the array indicate how the receiver should continue broadcasting a message. If a message is received with *travel*[i] set to TRUE, this indicates that the message should be sent out on link i.

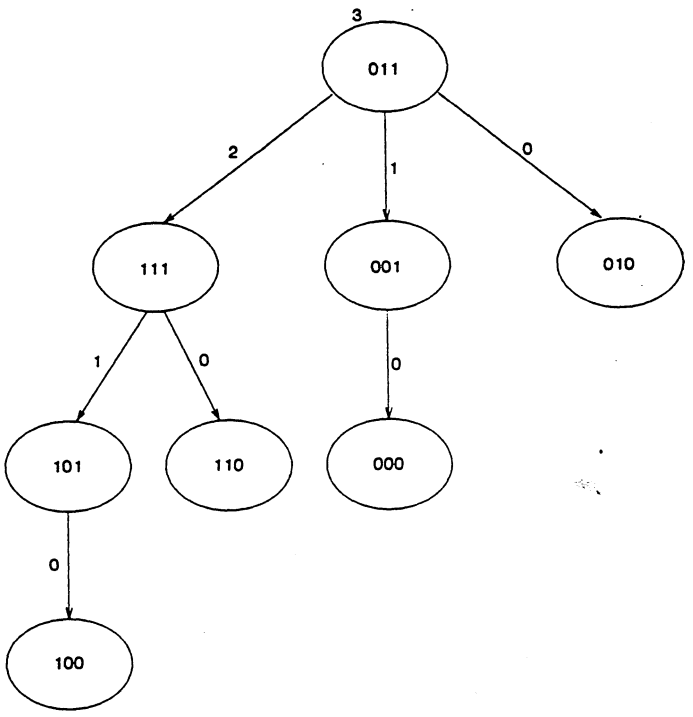

Fig. 3. Broadcast from 011 in a cube with eight nodes.

See Algorithm 5. Fig. 4 shows how this algorithm broadcasts a message from node 011 in a cube with eight nodes.

Algorithm 5 (forward (or originate) a broadcase message in a complete hypercube, using *travel* array):

Start at any node with all elements in the *travel* array set to TRUE.
for each link l from this node:
    if *travel*[l] is TRUE:
        send the message on link l with the array *newtravel* computed as:
        for $i \in 0 \cdots \log2(n)-1$:
            newtravel[i] ← TRUE **iff**
                travel[i] is TRUE AND $i < l$.

The broadcast algorithm for incomplete hypercubes is obtained by modifying Algorithm 5 to remember which links were not traversed because they did not exist. See Algorithm 6. Fig. 5 shows how this algorithm broadcasts a message from node 011 in a cube with seven nodes.

Algorithm 6 (forward (or originate) a broadcast message in an incomplete hypercube):

Start the algorithm with all elements in the *travel* array set to TRUE.
for each link l that exists from this node:
    if *travel*[l] is TRUE:
        send the message on link l with the array *newtravel* computed as:
        for $i \in 0 \cdots \log2(n)-1$:
            newtravel[i] ← TRUE **iff**
                travel[i] is TRUE AND
                ($i < l$ OR link i from this node does not exist).

We now show that the broadcast algorithm, Algorithm 6, routes broadcast messages in exactly the same way that the routing algorithm, Algorithm 2, does. More precisely, if a message is broadcast from some node s, we show that for any other node t, the broadcast algorithm routes the message from node s to node t over exactly one path: the path determined by the routing algorithm to route a message from node s to node t. Given this fact, it should be

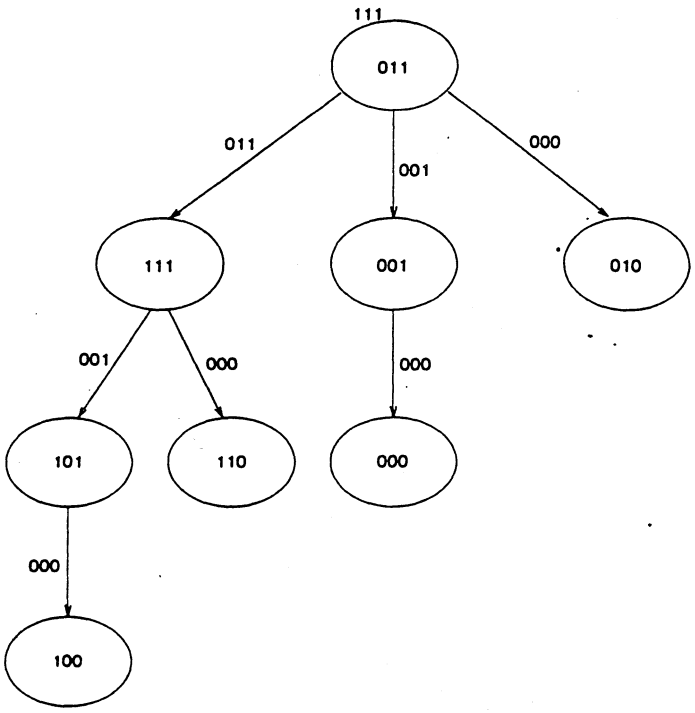

Fig. 4. Broadcast from 011 using travel array in a cube with eight nodes.

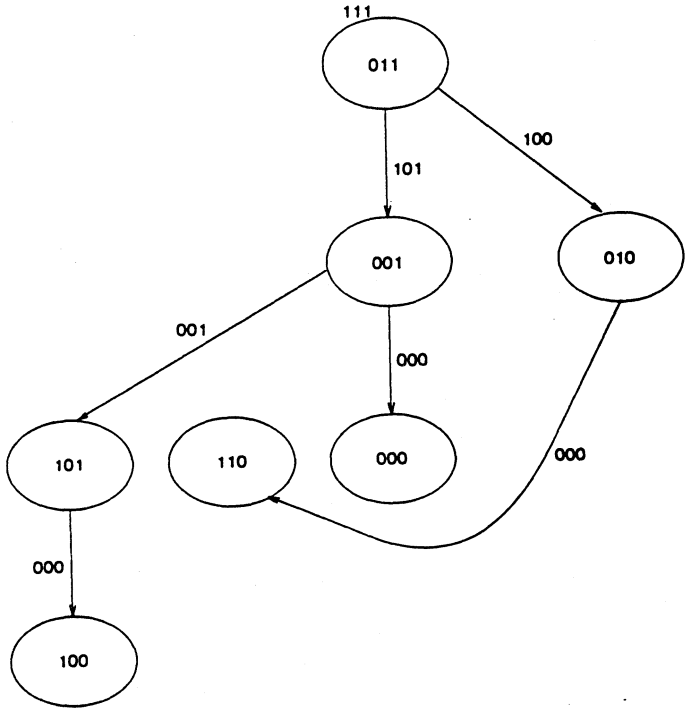

Fig. 5. Broadcast from 011 in an incomplete cube with seven nodes.

clear that a message broadcast by Algorithm 6 routes the message to each node exactly once, and does so with a path length no greater than $\log2(n)$.

We demonstrate the equivalence of the paths obtained by the routing and broadcast algorithms in two steps. We first show that the broadcast algorithm routes a message along this path and then show that messages are not routed along any other paths.

Theorem 3: Given two distinct nodes s and t, let p be the path determined by Algorithm 2 to route a message from node s to node t. If a message is broadcast from node s by Algorithm 6, then it is received at node t via path p.

Proof: Define the ith node of path p as p_i. We show by induction on i that Algorithm 6 reaches p_i along the path $p_0, p_1, \cdots, p_{i-1}$, and that for each bit j in which p_{i-1} and t differ, *travel*[j] is TRUE in the *travel* array at node p_{i-1}.

For $i = 0$, the basis step of the induction, observe that p_0 is node s and the Algorithm 6 starts with each element of the *travel* array set to TRUE.

For the induction step, we assume that the hypothesis is true for $i-1$ and show that it is true for i. Let k be the most significant bit of the relative address of p_{i-1} and t for which a link from node p_{i-1} exists. By its definition, we see that Algorithm 2 routes the message from node p_{i-1} to node p_i via link k. Algorithm 6 sends the message via link i because, by the induction hypothesis, *travel*[k] is true. To see that the *travel* array sent to node p_i has the appropriate bits set to TRUE, note that Algorithm 6 duplicates the values of the first k elements of the *travel* array at node p_{i-1} in the *travel* array sent to node p_i. Since the bitwise representation of p_{i-1} and p_i agree in the first k bits, these elements of *travel* are set to the appropriate values at node p_i. For the other bits in which p_i differs from t, note that p_{i-1} also differs from t in these bits. Since Algorithm 2 did not traverse any of the links from node p_{i-1} indicated by these bits, these links do not exists from node p_{i-1}. Thus, Algorithm 6 retains a TRUE value for these elements of the *travel* array sent to node p_i.

By induction, we see that a message broadcast from node s with Algorithm 6 reaches node t via path p. ■

Theorem 4: Given two distinct nodes s and t, let p be the path determined by Algorithm 2 for routing a message from node s to node t. If a message is broadcast from node s by Algorithm 6 and arrives at node t via path p', then $p' = p$.

Proof: For the sake of contradiction, suppose that a message broadcast from node s reaches t by some path $p' \neq p$. Let i be the smallest integer where $p_i' \neq p_i$. Let k be the bit number in which p_{i-1} and p_i differ and let k' be the bit number in which p_{i-1} and p_i' differ. Since a path determined by Algorithm 6 never traverses two links with identical link numbers, p_{i-1} differs from t both in bits k and k'. Furthermore, links k and k' exist from node p_{i-1}. Suppose $k < k'$. Then Algorithm 2 would route the message to t via link k', a contradiction. Suppose $k > k'$. Then the *travel* array sent to node p_i' would have *travel*[k] set to FALSE, because link k exists from node p_{i-1}. The broadcast message would never reach node t from node p_i' because these nodes differ in bit k, a contradiction. ■

Because the broadcast algorithm routes messages over the same paths as the routing algorithm, it is clear that the broadcast algorithm does not exhibit deadlock. Furthermore, if broadcasting and routing are done simultaneously in an incomplete hypercube, deadlock does not occur.

VI. Conclusions

We have shown that there are simple and effective algorithms for routing and broadcast in incomplete hypercubes. It is thus feasible to build multiprocessor computing systems based on the incomplete hypercube topology. This improves on existing systems based on complete hypercubes because complete hypercubes restrict the sizes of systems that can be constructed. Systems based on incomplete hypercubes can be built with any number of computing nodes. The routing and broadcast algorithms presented here assume that the network is fault-free. It would be of interest to extend these algorithms to allow deadlock-free routing in the face of faulty nodes or links.

Acknowledgment

We thank M. Yannakakis for suggesting this proof of correctness for Algorithm 6.

References

[1] J. Squire and S. M. Palais, "Programming and design considerations of a highly parallel computer," in *Proc. AFIP Spring Joint Comput. Conf.*, vol. 23, 1963, pp. 395–400.

[2] H. Sullivan and T. R. Bashkow, "A large scale homogeneous, fully distributed parallel machine, I," in *Proc. Fourth Symp. Comput. Architecture*, Mar. 1977, pp. 105–117.