define the generation potential of $l2$ and $l1$ to be the maximum delay of a carry generated anywhere in these two sections to the center of the adder. This delay is 6.2 ns. Each block of $r2$ will be made to have the maximum length so that the generation potential of $l2$ and $l1$ (6.2 ns) *plus* the delay from the output of the mux at the of $l1$ to the new block's msb is limited by 11 ns. The resulting $r2$ is shown in Fig. 6(f).

Just like $l1$, $r1$ may have to be modified so that a carry that starts in $r1$ and ends in another right-hand-side section always has a life shorter than 11 ns. Modifying a right-hand-side section such as $r1$ involves providing a direct path from the carry output of a section (which is a mux output) to the input of the mux at the end of the section.

Continuing in this fashion, the finished adder is shown in Fig. 6(g).

The adder has 60 b, more than enough for adding a pair of 52-b mantissas and guard bits of a double-precision floating-point number conforming to IEEE Standard 754.

V. Concerning Long Wire Delays

It is highly unlikely that metal delays will be significant in any reasonable layout of an adder of our type. According to Annaratone [1, p. 135], if the wire (metal) width is 2-μm, then the RC constant of aluminum per centimeter (10 000 μm) of length is quite a bit less than the RC constant of a minimum feature size MOS capacitor. And for wire (metal) widths well into the submicrometer range, it would take several hundred micrometers of wire to have the same RC constant as a MOS transistor. Thus, in 2-μm technology and even for much smaller technology, wire delays should be quite insignificant. The capacitive wire delays that the muxes at the end of a section have to drive is also still small, and can be compensated if desired (as discussed earlier) by enlarging those few muxes just slightly.

Acknowledgment

The author is grateful to the constructive criticisms offered by the editor, Prof. M. J. Irwin, and referees A and B and the new referee who read the first revised version. These comments helped greatly in transforming the theoretical results of the original version of this correspondence into truly practical adders.

References

[1] M. Annaratone, *Digital CMOS Circuit Design.* Boston. MA: Kluwer, 1986.
[2] C. Babbage, *On the Mathematical Powers of the Calculating Engine,* 1837. (Reprinted in *The Origins of Digital Computers,* B. Randell, Ed. Berlin: Springer-Verlag, 1973.
[3] Pak K. Chan and Martine D. F. Schlag, "Analysis and design of CMOS manchester adders with variable carry-skip," *IEEE Trans. Comput.,* Aug. 1990; an earlier version appeared in *Proc. 9th IEEE Symp. Comput. Arithmetic,* 1989.
[4] L. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits.* Reading MA: Addison-Wesley, 1985.
[5] A. Guyot, B. Hochet, and J.-M. Muller, "A way to build efficient carry-skip adders," *IEEE Trans. Comput.,* Oct. 1987.
[6] V. Kantabutra, "Designing optimum one-level carry-skip adders," *IEEE Trans. Comput.,* vol. 42, no. 6, pp. 759–764, June 1993.
[7] ——, "A recursive carry-lookahead/carry-select hybrid adder," *IEEE Trans. Comput.,* to appear.
[8] T. Lynch and E. Swartzlander, "A spanning tree carry lookahead adder," *IEEE Trans. Comput.,* vol. 41, Aug. 1992.
[9] V. G. Oklobdzija and E. R. Barnes, "Some optimal schemes for ALU implementation in VLSI technology," in *Proc. 7th Symp. Comput. Arithmetic,* 1985.
[10] S. Turrini, "Optimal group distribution in carry-skip adders," in *Proc. 9th IEEE Symp. Comput. Arithmetic,* 1989.

Broadcasting on Incomplete Hypercubes

Jenn-Yang Tien, Ching-Tien Ho, and Wei-Pang Yang

***Abstract*—Incomplete hypercubes make the hypercubes more flexible on task allocation in large cubes, cost of manufacturing hardware, and hypercubes with faulty nodes. In this correspondence, we devise and analyze a broadcasting algorithm based on edge-disjoint spanning trees in an incomplete hypercube of $2^n + 2^k$ nodes, where $0 \leq k < n$. The broadcasting algorithm is strictly optimal.**

***Index Terms*—Broadcasting, edge-disjoint spanning trees, incomplete hypercubes, routing.**

I. Introduction

Broadcasting is one of the most important communication primitives used in multiprocessors with distributed memory. It is frequently used in a variety of linear algebra algorithms, such as matrix multiplication and Gaussian elimination. The reverse operation of broadcasting is *reduction,* in which the data set is reduced by applying operators such as addition/subtraction and max/min.

The binary (Boolean) hypercube topology has been an attractive architecture [12] in connecting from tens to tens of thousands of processors. Among the attractive architectural features are its logarithmic diameter, rich bandwidth, regular structure, fault tolerance, and, more importantly, the fact that many regular (algorithm or machine) structures can be embedded into it with adjacency (or nearly) preserved. Current examples of commercially available hypercubes are the Intel iPSC/860, the NCUBE 2, and the Connection Machine model CM-2. Despite its attractive features, however, one restriction of the hypercube topology is that the size of a system has to be an integer power of two. It leaves a large gap between the two sizes of systems that can be constructed, or of subcubes that can be assigned to an application in a multitasking environment [4]. An incomplete hypercube [10]—a hypercube missing a certain number of nodes—improves this restriction. An incomplete hypercube may be caused by the hardware construction; by some nodes of a complete hypercube becoming faulty; or by allocating a partial number of nodes of a complete hypercube to an application in a multiple-host system. Extensive study on the incomplete hypercube has been conducted recently [2], [16], [17], since it was first introduced by Katseff [10].

The communication model is assumed to be *store-and-forward* or *packet-switched.* Messages may be broken down into packets of the

Manuscript received July 22, 1991; revised July 9, 1992 and January 21, 1993. This work was supported in part by the National Science Council, Republic of China, under Contract NSC79-0408-E009-01. This paper was presented at the 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, TX, 1991.

J.-Y. Tien is with the Chung Shan Institute of Science and Technology, Lungtan, Taoyuan, Taiwan, R.O.C.
C.-T. Ho is with IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.
W.-P. Yang is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

IEEE Log Number 9212775.

same size. A node receiving a packet must finish receiving it before any of its content can be utilized. All incident links of a node can be used simultaneously for transmission or reception. We use M to denote the number of elements to be broadcast; t_c is the transfer time for an element, and τ is the start-up for communication of a packet of maximum B elements. For clarity, we first assume that $t_c = 1$ and $\tau = 0$ in deriving the communication complexity. It is straightforward to derive the general complexities from the simplified ones, and we give the general complexities also.

Broadcasting on a graph can be realized based on one spanning tree, or a set of spanning trees (termed spanning graphs), with the root being the source node. Graphs of minimum height have minimum propagation time which is the overriding concern for small data volumes or for a high communication overhead. For large data volumes, it is important to use the bandwidth of the interconnection network effectively, in particular, if each processor is able to communicate on all its ports concurrently. *Binomial spanning tree* and *n edge-disjoint spanning trees* proposed by Johnsson and Ho [8] are used for broadcasting algorithms on hypercubes. In this correspondence, we generalize the work of Johnsson and Ho in [8] to the case of incomplete hypercubes. The *edge-disjoint spanning tree* (EDST) is composed of $k+1$ spanning trees in which any two spanning trees are edge-disjoint (in the directed-edge sense). The $k+1$ parallel paths in the $(k+1)$ edge-disjoint spanning trees offer a speedup of up to $k+1$ over the algorithm based on the hierarchical binomial spanning tree [15].

The remainder of this correspondence is organized as follows. Notations, properties of incomplete hypercubes, and spanning trees in complete hypercubes are given in Section II. In Section III, the edge-disjoint spanning tree (graph) is defined and characterized. Broadcasting based on edge-disjoint spanning tree and broadcasting complexity estimation is given in Section IV. Section V concludes the results.

II. Preliminaries

Let Q_n be the n-dimensional Boolean cube, or n-cube. Then, an incomplete hypercube of N' nodes, where $2^n < N' < 2^{n+1}$, is the graph induced by the node set $\{0, 1, \cdots, N'-1\}$ in Q_{n+1}. In this correspondence, we only consider incomplete hypercubes of sizes $2^n + 2^k$, where $0 \leq k < n$. Such an incomplete hypercube, denoted I_k^n, consists of an n-cube and a k-cube. Throughout the correspondence, we use $N = 2^n$ and $K = 2^k$, and refer to Q_n and Q_k as the *front cube* and *back cube*, respectively. Nodes are numbered from 0 to $N-1$ in the front cube, and from N to $N+K-1$ in the back cube. Node addresses are represented by $(n+1)$-bit binary numbers $(x_n x_{n-1} \cdots x_0)$. Thus, $(0*^n)$ and $(10^{n-k}*^k)$ denote the front cube and back cube, respectively, where $*$ is a *don't-care* symbol whose value can be 0 or 1, and y^i stands for i consecutive y's.

A. Subcubes in Incomplete Hypercubes

A set of nodes having the same lower k-bit value in their binary node addresses constitutes a (complete) subcube in the incomplete hypercube I_k^n. A *respective subcube* S_i of I_k^n is defined as $S_i = (x_n \cdots x_k *^k)$, such that the value of $(x_n \cdots x_k) = i$.

With distinct higher $(n-k+1)$-bit values, there are $2^{n-k}+1$ *respective* subcubes. For convenience, we will also refer to the back cube as S_b exchangeably with $S_{2^{n-k}}$. A set of nodes is *corresponding* if all nodes in the set have the same value in their lower k bits of binary node addresses. A set of *corresponding nodes* in the front cube, which form a (complete) subcube in the original incomplete hypercube, is called a *corresponding subcube*. Corresponding subcube

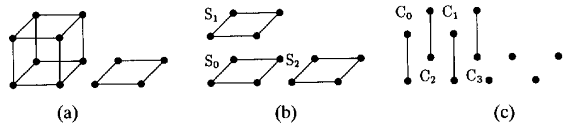

Fig. 1. Incomplete hypercube I_2^3.

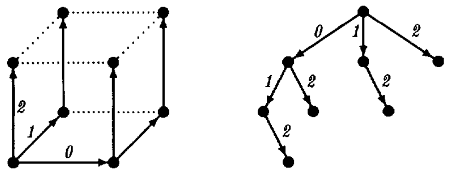

Fig. 2. Binomial spanning tree and binomial broadcasting in a 3-cube.

C_i, where $0 \leq i < 2^k$, is defined as $C_i = (0*^{n-k} x_{k-1} \cdots x_0)$, such that the value of $(x_{k-1} \cdots x_0) = i$.

Corresponding subcubes are $(n-k)$-cubes, and links in each corresponding subcube have dimensions in the range of k to $n-1$. There are 2^k corresponding subcubes in the front cube, each of which is associated with an node in the back cube.

Example: Fig. 1(a) shows an I_2^3 with 4 cross edges (of dimension 3) being omitted. The binary addresses for nodes in S_0, S_1, and S_2 have a form $(00*^2)$, $(01*^2)$, and $(10*^2)$, respectively. Corresponding subcubes C_0, C_1, C_2, and C_3, which are shown in Fig. 1(c), have binary node addresses $(0*00)$, $(0*01)$, $(0*10)$, and $(0*11)$, respectively.

B. Binomial Spanning Tree (BST)

A 0-level binomial tree has one node. An n-level binomial tree is constructed out of two $(n-1)$-level binomial trees by adding one edge between the roots of the two trees, and making either root be the new root. The well-known spanning tree on a hypercube generated by the e-cube routing algorithm [14] is indeed a binomial tree. We refer to it as the *binomial spanning tree* (BST) in the following. Broadcasting based on the BST is to double, per step, the number of processors holding the source data, referred to as the *binomial broadcasting* in the following. Fig. 2 shows the binomial spanning tree and the binomial broadcasting in a 3-cube. The spanning tree on the right of the figure is a binomial tree for which the number of nodes at level i is $\binom{n}{i}$.

Applying pipelining, the propagation time to the node farthest away from the source is at least n. When this node receives all elements, the broadcasting is terminated. The total time for broadcast is then $M + n - 1$.

C. Edge-Disjoint Spanning Trees (EDST's)

In [8], Johnsson and Ho defined n directed spanning trees in an n-cube and showed that they are all edge-disjoint (in the directed-edge sense). For the construction of the jth spanning tree, where $j \in \{0, 1, \cdots, n-1\}$, the root first extends an edge across dimension j to node $x = (0^{n-j-1}10^j)$. Then, construct a spanning tree rooted at node x in subtree $(*^{n-j-1}1*^j)$ according to the sequence of dimensions $(j+1) \bmod n, (j+2) \bmod n, \cdots, (j-1) \bmod n$. By connecting each node (but node x) in subcube $(*^{n-j-1}1*^j)$ to its corresponding node in subcube $(*^{n-j-1}0*^j)$ by a new edge, the jth spanning tree is formed. Fig. 3(a) shows an example of 3 EDST's in a 3-cube where labels on directed links are their associated link dimensions. The subtree rooted at the immediate successor of the root spans all (but the root) nodes in the hypercube.

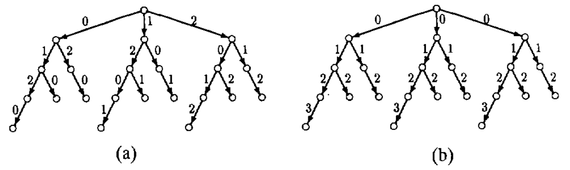

Fig. 3. EDST and broadcasting based on EDST in 3-cube.

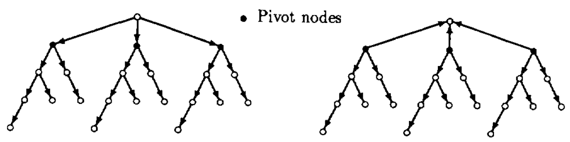

Fig. 4. Rooted nEDST and pivoted nEDST in 3-cube.

An example of one-port broadcasting of 3 elements based on the n EDST on a 3-cube is shown in Fig. 3(b). Labels on links are time steps of communication. After n steps, all but the links entering the root are filled with units of message. The number of communication steps of broadcasting for algorithms based on the nEDST is shown to be $\lceil M/n \rceil + n$, which is a factor of n faster than the BST broadcasting.

Note that it is possible to combine the nEDST algorithm and the BST algorithm [13] to save one communication step [6], and yield a truly optimal broadcasting algorithm (with respect to the number of elements transferred in sequence). Thus, broadcasting based on the nEDST attains a time of $\lceil M/n \rceil + n - 1$ in an n -cube.

III. Edge-Disjoint Spanning Trees in Incomplete Hypercubes

In this section, we describe edge-disjoint spanning trees in I_k^n, which will be used as the basis for our broadcasting algorithm introduced in the next section. Two types of edge-disjoint spanning trees in a (complete) hypercube are defined first to facilitate the description. Then, we construct $k+1$ EDST's, $(k+1)$EDST, on $\{S_0, S_1, S_b\}$, which is the incomplete hypercube I_k^{k+1}, before a general construction on I_k^n is described.

A. Rooted and Pivoted nEDST

The nEDST in Q_n (described in Section II-C), in which the set of edge-disjoint spanning trees have a common root (the root of nEDST), is referred to as the *rooted nEDST*. Another type of n edge-disjoint spanning trees, in which the direction of edges between the root and its immediate successors in the rooted nEDST is reversed, is referred to as the *pivoted n EDST*. The rooted nEDST and pivoted nEDST in a 3-cube are shown in Fig. 4. Since the n incoming edges of the root of nEDST are unused edges, the n spanning trees in the pivoted nEDST are still edge-disjoint. Both the n immediate successors of the common root of n spanning trees in rooted nEDST, and the distinct n roots of n spanning trees in the pivoted nEDST, are referred to as the *pivot nodes*.

B. $k+1$ EDST's in $\{S_0, S_1, S_b\}$

In the following, we will show the construction of $k+1$ edge-disjoint spanning trees in $I_k^{k+1} \equiv \{S_0, S_1, S_b\}$. The construction is separated into three cases according to the position of the root. The construction has two steps. In the first step, a ($k+1$)EDST in Q_{k+1} is constructed. In the second step, nodes (and links) of the other Q_k are added.

1) Root in S_0: We first construct $k+1$ EDST's in $\{S_0, S_1\}$, which is Q_{k+1}, with the root being in S_0. Then, for each spanning tree j, where $0 \le j \le k$, we attach nodes in S_b and preserve them edge-disjoint. Fig. 5 gives a global view of attaching nodes in S_b to existing spanning trees. For each node u_0 in S_0 in the original spanning tree j, if it has a child, say, u_1, connected through a dimension-k edge, then we duplicate the subtree rooted at u_1 and attach it to node u_0 via a dimension-n edge. The duplicated subtree is rooted at node u_b. (By duplicating a subtree, we mean that the structure of the subtree and the dimensions of the edges are preserved. The node addresses are derived accordingly once the root is assigned with address u_b.) Note that all nodes in the duplicating subtree are in S_1, and those in the duplicated subtree are in S_b. (The only exception is that u_0 is the root, in which case, all leaf nodes in the subtree are in S_0.) Then, all the leaf nodes along dimension k in the duplicated subtree are deleted. (This guarantees that all nodes in duplicated subtrees are in S_b.)

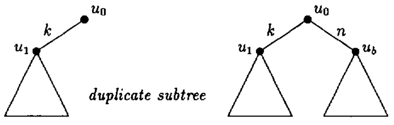

Fig. 5. Duplicating a subtree along link n appended to node u_b.

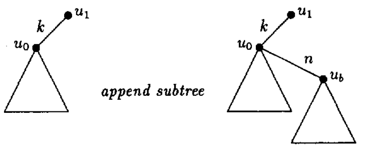

Fig. 6. Appending the subtree rooted at u_0 to node u_b along link n.

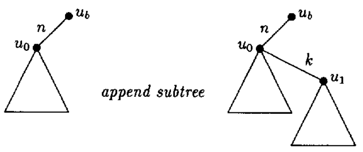

Fig. 7. Appending the subtree rooted at u_0 to node u_1 along link k.

2) Root in S_1: We first construct $k+1$ EDST's in $\{S_0, S_1\}$ with the root being in S_1. Then, for each spanning tree j, where $0 \le j \le k$, we append nodes in S_b. Fig. 6 gives a global view of appending a subtree. As before, leaf nodes along link k in the appended subtrees are delete to generate a spanning tree in I_k^{k+1}.

3) Root in S_b: Since S_b is similar (isomorphic) to S_1 from the viewpoint of S_0, this case is similar to case 2. We first construct $k+1$ EDST's in $\{S_0, S_b\}$ with the root being in S_b. Fig. 7 shows the global view.

Thus, we have the next two lemmas.

Lemma 1: We have constructed $k+1$ EDST's rooted at any node in an I_k^{k+1}.

Proof: We give the proof of case 1 (root in S_0). The other two cases can be proved in a similar way. When $k+1$ EDST's in $\{S_0, S_1\}$ are constructed, nodes in S_1 are connected to the root through exactly one link in dimension k. The duplication of a subtree makes corresponding nodes (in S_b) of those node (in S_1) in the subtree connected to the root. Since edges in subtrees being duplicated are disjoint, the resulting $k+1$ spanning trees are edge-disjoint. □

Lemma 2: For any given node i in $\{S_0, S_1\}$, we can construct $k+1$ pivoted EDST's such that they are all rooted at different nodes in $\{S_0, S_1\}$ and their common neighbor is node i.

Proof: We construct the $k+1$ pivoted EDST's by taking the $k+1$ EDST's rooted at node i and reversing the edge from the root

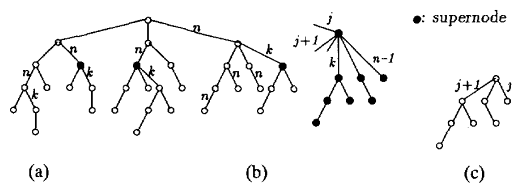

Fig. 8. Edge-disjoint spanning trees in I_2^6 (source is in back cube).

(i.e., node i) to each pivot node. It is easily seen that only one "new" directed edge, which was not used by the $k+1$ EDST's rooted at node i, is added to each of the $k+1$ pivoted EDST's. Furthermore, these $k+1$ new added edges are all different, which implies that the $k+1$ pivoted EDST's remain edge-disjoint. □

C. $k+1$ EDST's in I_k^n

To simplify the description, we first assume that the root is in $\{S_0, S_1, S_b\}$. Then, the case that the root is in the other subcube is considered.

When $k+1$ EDST's in $\{S_0, S_1, S_b\}$ are constructed, the $(k+1)$EDST in I_k^n (rooted at node in $\{S_0, S_1, S_b\}$) can be constructed as follows. Partition the front cube into subcubes of dimension $k+1$ with respect to dimensions $0, 1, \cdots, k$. For convenience, let Y_i, where $1 \leq i < 2^{n-k-1}$, be the $(k+1)$-dimensional subcube $\{S_{2i}, S_{2i+1}\}$. Let the root node be r_0 in $\{S_0, S_1, S_b\}$. Also, denote the corresponding root (i.e., having the same $k+1$ lower order bits) of r_0 in Y_i as r_i. We call the $k+1$ neighbors of r_i in Y_i the pivot nodes. The $k+1$ corresponding nodes in $\{S_0, S_1\}$ of the distinct pivot nodes (in Y_i) are also called pivot nodes. We are now ready to describe the construction rules of $k+1$ EDST's in I_k^n.

1) Construct the $k+1$ EDST's in $\{S_0, S_1, S_b\}$ rooted at node r_0.

2) Within each Y_i, construct the $k+1$ pivoted EDST's rooted at the $k+1$ pivot nodes within Y_i.

3) Construct $k+1$ BST's, where the jth BST is rooted at the jth pivot node in $\{S_0, S_1\}$ spanning all the jth pivot nodes in Y_i's for all i.

From the individual point of view, each of the $k+1$ spanning trees in I_k^n consists of three types of spanning trees: a spanning tree of $\{S_0, S_1, S_b\}$ rooted at the source node (step 1); a BST on corresponding pivot nodes (step 3); and a spanning tree of Y_i, for all i, rooted at the pivot node spanned by the BST (step 2).

Example: A part of edge-disjoint spanning trees rooted at a node in S_b is illustrated in Fig. 8. Subtrees along dimension n are duplicated from those in dimension k. There are $k+1$ supernodes (darked nodes) in Fig. 8(a) corresponding to $k+1$ pivot nodes in $\{S_0, S_1, S_b\}$. Each supernode stands for a BST [Fig. 8(b)] among corresponding pivot nodes (one in each Y_i). Adjacent edges of the supernode in Fig. 8(a) are adjacent to the root of BST in Fig. 8(b). Each node in Fig. 8(b) represents one of the $k+1$ pivoted EDST's on the respective subcube shown in Fig. 8(c). The largest subtree of BST in Fig. 8(c) is on the dimension $(j+1) \bmod k$ [Fig. 8(b)], provided that dimension j is the dimension of link to which the root of BST [Fig. 8(b)] is adjacent.

Now, we describe the condition that the root is not in $\{S_0, S_1, S_b\}$. Let the root node be in $Y_\alpha, 1 \leq \alpha < 2^{n-k-1}$. The $(k+1)$EDST in I_k^n is constructed as follows. The $k+1$ EDST's in $\{S_0, S_1, S_b\}$ are constructed in pivoted form (Lemma 2). $k+1$ EDST's in $Y_i, i \neq \alpha$, are also in pivoted form; and $k+1$ EDST's in Y_α are rooted. $k+1$ BST's which connect $k+1$ sets of corresponding pivot nodes are rooted at pivot nodes in Y_α.

Theorem 1: The data structure constructed above is composed of $k+1$ edge-disjoint spanning trees.

Proof: In the construction, $k+1$ edge-disjoint spanning trees in each respective subcube are constructed. The $k+1$ immediated successors of the root in the rooted $(k+1)$EDST and $k+1$ roots in each pivoted $(k+1)$EDST act as "pivots" among $k+1$ sets of spanning trees. Each set of spanning trees, composed of a spanning tree in each respective subcube and a BST in pivot nodes of those spanning trees, is a spanning tree in the incomplete hypercube. Since the $k+1$ BST's are in distinct 2^{n-k-1} sets, they are node-disjoint; and spanning trees in respective subcubes are edge-disjoint. The $k+1$ spanning trees of I_k^n are edge-disjoint. □

Lemma 3: The height of $k+1$ EDST's is $n+2$ if the root is in respective subcube $S_{2^{n-k}-1}$ or S_b, and the height is $n+1$ otherwise.

Proof: When the root is in $S_{2^{n-k}-1}$ (respectively, S_b), pivot nodes in Y_0 (respectively, $Y_{2^{n-k-1}-1}$ are in the lowest level of the BST's of pivot nodes. The height of EDST's is 1 + $(n-k-1)$ + 1 + $(k+1)$ [respectively, 2 + $(n-k-1)$ + $(k+1)$] = $n+2$. When the root is in any other respective subcube, the furthest pivot nodes are $n-k-1$ hops away from pivot nodes in Y_α. □

The diameter of the incomplete hypercube I_k^n is $n+1$. When the root is in a subcube other than S_b or $S_{2^{n-k}-1}$, the furthest node away from the root is n, i.e., the diameter of n-cube. The height of the EDST's is at most one higher than optimal. It is optimal if the height of $k+1$ EDST's in an n-cube is at least $n+1$. The optimal height of n EDST's in the n-cube is known to be $n+1$ [8]. In the following, we prove that the height $n+2$ of $k+1$ EDST's rooted at node in S_b or $S_{2^{n-k}-1}$ is optimal.

Theorem 2: The maximum height of $k+1$ EDST's in I_k^n is optimal when the root is in $S_{2^{n-k}-1}$ or S_b, and $k > 0$.

Proof: If the root is in S_b, k out of $k+1$ neighbors of the root are in S_b. The furthest nodes away from nodes in S_b have distance $n+1$. Since the root has only one child node in each spanning tree, the maximum height of $k+1$ EDST's is at least $n+2$.

If the root r is in $S_{2^{n-k}-1}$, $\bar{r}$ is in S_b; where $\bar{r}$ is the node with complementing address bits of r. Clearly, node $\bar{r}$ is $n+1$ hops away from node r. Since each node in S_b has $k+1$ parent nodes (w.r.t. $k+1$ EDST's), all incoming links of a node in S_b are used in the EDST's. For neighbors of $\bar{r}$ to them implies that the paths from r to those passing through $\bar{r}$ have hops at least $n+2$. □

IV. Broadcasting on Incomplete Hypercubes

There are two important factors which affect the communication complexity of broadcasting. One is the average number of packets which are sent per step from the root—*bandwidth utilization*, denoted by α. The higher α an algorithm attains, the better the algorithm performs. The other factor is the propagation delay for a packet—*latency*, denoted by β; that is, the number of steps for a packet to reach all nodes from the step it was sent out from the root. The communication complexity of a broadcasting algorithm, for which both bandwidth utilization and latency are stable (i.e., does not depend on the time step), is $\lceil M/\alpha \rceil + \beta - 1$.

Lemma 4: The lower bound on latency is $n+1$.

Proof: The longest distance between two nodes in the incomplete hypercube is $n+1$. □

Lemma 5: The maximal bandwidth utilization is $k+1$.

Proof: There are $k+1$ outgoing (incoming) edges, or parallel paths, for nodes in the back cube. It is useless even if the root can send out more than $k+1$ elements in a step (when the root is in the front cube), since the links which are adjacent to nodes in the back cube are fully loaded. □

Theorem 3: The lower bound of broadcasting is $[M/(k+1)] + n$.

Proof: It is easily obtained from Lemmas 4 and 5. □

A. Edge-Disjoint Hamiltonian Paths

In this subsection, we prove that there exist at least k EDHP's in

I_k^n if k is an even number; and there exist $k + 1$ EDHP's in I_k^n if k is an odd number. For ease of describing, we assume, at first, that the graph is undirected. Under the undirected graph, we prove that there exist $k/2$ (respectively, $[k + 1]/2$) edge-disjoint Hamiltonian cycles (EDHC's). Each undirected EDHC represents two directed EDHC's—one for each direction. Removing the incoming edge into the root, each directed EDHC is exactly a directed EDHP.

Lemma 6 [3]: If graphs C_1 and C_2 are Hamiltonian, then $C_1 \times C_2$ contains two edge-disjoint Hamiltonian cycles (EDHC's).

Lemma 7 [1]: If the graph G contains two EDHC's and the graph C is Hamiltonian, then the graph $G \times C$ contains three EDHC's.

Lemma 8: If the graph C is Hamiltonian and L_2 is an edge, then the graph $C \times L_2$ is Hamiltonian.

Proof: The product graph $C \times L_2$ can be considered as two planes, each containing a copy of C, and completing each edge perpendicular to both planes. There exists a cycle traversing all vertices in the same plane. Select adjacent vertices, u and v, in the cycle remove edge (u, v), duplicate the path on the other plane, and add perpendicular edges of both u and v, and a Hamiltonian cycle in $C \times L_2$ is formed. □

Lemma 9: The product graph $C^n = \underbrace{C \times C \times \cdots \times C}_{n}$ contains n EDHC's, where C is a cycle with arbitrary number of vertices.

Proof: We prove the lemma by induction on the number of C's. By Lemma 6, $C \times C$ contains 2 EDHC's. By Lemmas 6 and 7, $C \times C \times C$ contains 3 EDHC's. Assume that C^i contains i EDHC's for all $i < n$. We wish to show that C^n contains n EDHC's.

- If n is even, then $C^n = C^{n/2} \times C^{n/2}$. By the induction hypothesis, the product graph $C^{n/2}$ contains $n/2$ EDHC's. Consider the $n/2$ product graphs which are formed as the product of the ith HC in the two product graphs $C^{n/2}$'s for $1 \leq i \leq n/2$. Each product graph contains two EDHC's, by Lemma 6. Furthermore, all these product graphs are edge-disjoint. Therefore, there exist n EDHC's.
- If n is odd, then $C^n = C^{(n-1)/2} \times C^{(n+1)/2}$. Pair each but one HC of the $C^{(n-1)/2}$ with an HC in the $C^{(n+1)/2}$. This leaves one HC of the $C^{(n-1)/2}$ and two HC's of the $C^{(n+1)/2}$ unpaired. By Lemmas 6 and 7, there are n EDHC's in the product graph C^n. □

Theorem 4: I_k^n contains $k + 1$ directed EDHC's when $k + 1$ is even, and contains k directed EDHC's when $k + 1$ is odd.

Proof: Since $L_{2^{n-k}}$ is a subgraph of $Q_{2^{n-k}}$ with the same set of vertices, $H = Q_k \times L_{2^{n-k}+1}$ is a subgraph of I_k^n with the same set of vertices. Note that $Q_k = \underbrace{L_2 \times L_2 \times \cdots \times L_2}_{k}$. We are going to prove that there are directed $l + 1$ (respectively, k) EDHC's in H, when $k + 1$ is even (respectively, odd).

Case 1: $k + 1$ is even. Since $L_i \times L_j$ is Hamiltonian, H contains a subgraph of the form $C^{(k+1)/2}$, with the same set of vertices. By Lemma 9, there exist $(k + 1)/2$ EDHC's in H. Thus, there are $k + 1$ directed EDHC's in I_k^n.

Case 2: $k + 1$ is odd. $H = L_2^k \times L_{2^{n-k}+1} = L_2 \times L_2^{k-1} \times L_{2^{n-k}+1}$. By Lemma 8, H contains a subgraph of the form $C^{k/2}$, with the same set of vertices. By Lemma 9, there exist $k/2$ EDHC's in H. Thus, there are k directed EDHC's in I_k^n. □

B. Broadcasting Based on EDST's

We first examine the heights of the $(k + 1)$EDST. When the root is in the back cube, there is a pivot node (the one in S_1) which is two hops away from the source node, and the height of the $(k + 1$)EDST is 2 + $(n - k - 1)$ + $(k + 1) = n + 2$. When the source node is in $S_{2^{n-k}-1}$, the height of the $(k + 1)$EDST is 1 + $(n - k - 1)$ + $(k + 1)$ + 1 = n + 2 (the nodes in S_b have one hop deeper than their corresponding nodes in S_0). Precisely, the height is $x + 1$, where x is the longest distance from the source to other nodes in I_k^n. The number of communication steps of broadcasting based on the $(k + 1)$EDST is $\lceil M/(k + 1)\rceil + n + 1$, which has an optimal bandwidth utilization and one more step than the optimal latency.

TABLE I
COMMUNICATION COMPLEXITIES OF BROADCASTING ALGORITHMS ASSUMING $\tau = 0$ AND $t_c = 1$

Graphs	*HP*	*EDST*	*Lower Bound*
Complexity	$\lceil \frac{M}{k} \rceil + N + K - 2$	$\lceil \frac{M}{k+1} \rceil + n + 1$	$\lceil \frac{M}{k+1} \rceil + n$

TABLE II
COMMUNICATION COMPLEXITIES OF BROADCASTING ALGORITHMS

Graphs	T	B_{opt}	T_{min}
HP	$(\lceil \frac{M}{kB} \rceil + N + K - 2)(Bt_c + \tau)$	$\sqrt{\frac{M\tau}{k(N+K-2)t_c}}$	$(\sqrt{\frac{Mt_c}{k}} + \sqrt{(N + K - 2)\tau})^2$
EDST	$(\lceil \frac{M}{B(k+1)} \rceil + n + 1)(Bt_c + \tau)$	$\sqrt{\frac{M\tau}{(k+1)(n+1)t_c}}$	$(\sqrt{\frac{Mt_c}{k+1}} + \sqrt{(n + 1)\tau})^2$

C. Comparisons

Table I compares the communication complexities of the proposed broadcasting algorithm and the broadcasting algorithm based on a Hamiltonian path. Recall that this is a simplified model in which $\tau = 0$ and $t_c = 1$. The general complexities (T), optimal packet size (B_{opt}), and the complexity with optimal packet size ($T_{\min}$) are derived in Table II.

The number of edge-disjoint spanning trees embedded in the broadcasting graph dominates the bandwidth utilization of broadcasting. The longest path from the root to all other nodes in the graph dominates the latency of broadcasting. The EDST-based broadcasting algorithm always has good bandwidth utilization, as the number of edge-disjoint spanning trees increases. The broadcasting algorithm based on the edge-disjoint Hamiltonian paths has good bandwidth utilization. The latency, however, is extremely large. Broadcasting algorithms based on the proposed $k + 1$ EDST's have an optimal bandwidth utilization and near optimal latency for any values of (n, k) regardless of the position of the root.

V. CONCLUSION

The hypercube topology has the restriction that its size must be a power of two, leaving a gap between hypercubes of two adjacent sizes. The family of incomplete hypercubes alleviates such restriction. Incomplete hypercubes can also be derived from faulty hypercubes or from allocations of subsets of hypercube nodes, which do not necessarily form subcubes.

We have presented edge-disjoint spanning trees in incomplete hypercubes of size $2^n + 2^k$, and given the broadcasting algorithm based on the edge-disjoint spanning trees. We also derive the number of edge-disjoint Hamiltonian path, which is the extreme case of a spanning tree with height $2^n + 2^k - 1$. The bandwidth utilization of edge-disjoint Hamiltonian path is optimal (when k is even), or one less than optimal (when k is odd). However, the communication latency is extremely high. The number of communication steps of broadcasting based on the edge-disjoint spanning trees is $\lceil M/(k + 1)\rceil + n + 1$, which has an optimal bandwidth utilization and one more step than the optimal latency.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their helpful comments and suggestions.

References

[1] J. Aubert and B. Schneider, "Decomposition de la somme cartesienne d'un cycle et de l'union de deux cycles hamiltoniens en cycles hamiltoniens," *Discrete Math.*, vol. 38, pp. 7–16, 1982.

[2] H. L. Chen and N.-F. Tzeng, "Enhanced incomplete hypercube," in *Proc. 1989 Int. Conf. Parallel Process.*, vol. 1, pp. 270–277, 1989.

[3] M. Foregger, "Hamiltonian decompositions of products of cycles," *Descrete Math.*, vol. 24, pp. 251–260, 1978.

[4] W. D. Hillis, *The Connection Machine.* Cambridge, MA: M.I.T. Press, 1985.

[5] C.-T. Ho, "Optimal communication primitives and graph embeddings on hypercubes," Ph.D. dissertation, Yale Univ., May 1990.

[6] ——, "Optimal broadcasting on SIMD hypercubes without indirect addressing capability," *J. Parallel Distrib. Comput.*, 1991.

[7] C.-T. Ho and S. L. Johnsson, "Distributed routing algorithms for broadcasting and personalized communication in hypercubes," in *Proc. 1986 Int. Conf. Parallel Process.*, 1986, pp. 640–648.

[8] S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Comput.*, vol. 38, pp. 1249–1268, Sept. 1989.

[9] ——, "Embedding meshes into small Boolean cubes," in *Proc. 1990 Distrib. Memory Comput. Conf.*, 1990, pp. 1366–1374.

[10] H. P. Katseff, "Incomplete hypercubes," *IEEE Trans. Comput.*, vol. 37, pp. 604–608, May 1988.

[11] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. 37, pp. 867–872, July 1988.

[12] C. L. Seitz, "The cosmic cube," *Commun. ACM,* vol. 28, pp. 22–33, Jan. 1985.

[13] Q. F. Stout and B. Wagar, "Intensive hypercube communication prearranged communication in link-bound machines," *J. Parallel Distrib. Comput.*, vol. 10, pp. 167–181, 1990.

[14] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I," in *Proc. 4th Symp. Comput. Arch.*, 1977, pp. 105–117.

[15] J.-Y. Tien and W.-P. Yang, "Hierarchical spanning trees and distributing on incomplete hypercubes," *Parallel Comput.*, vol. 17, pp. 1343–1360, 1991.

[16] N.-F. Tzeng, "Structural properties of incomplete hypercubes," in *Proc. 10th Int. Conf. Distrib. Comput. Syst.*, May 1990, pp. 262–269.

[17] N.-F. Tzeng, H.-L. Chen, and P.-J. Chuang, "Embeddings in incomplete hypercubes," in *Proc. 1990 Int. Conf. Parallel Process.*, 1990, pp. 335–339.

A Routing and Broadcasting Scheme on Faulty Star Graphs

Nader Bagherzadeh, Nayla Nassif, and Shahram Latifi

Abstract—**In this correspondence, we present a routing algorithm that uses the depth first search approach combined with a backtracking technique to route messages on the star graph in the presence of faulty links. The algorithm is distributed and requires no global knowledge of faults. The only knowledge required at a node is the state of its incident links. The routed message carries information about the followed path and the visited nodes. The algorithm routes messages along the optimal, i.e., the shortest path if no faults are encountered or if the faults are such that an optimal path still exists.**

In the absence of an optimal path, the algorithm always finds a path between two nodes within a bounded number of hops if the two nodes are connected. Otherwise, it returns the message to the originating node. We provide a performance analysis for the case where an optimal path does not exist. We prove that for a maximum of $n-2$ faults on a graph with $N=n!$ nodes, at most $2i+2$ steps are added to the path, where i is $\mathcal{O}(\sqrt{n})$.

Finally, we use the routing algorithm to present an efficient broadcast algorithm on the star graph in the presence of faults.

Index Terms—**Fault tolerance, star graph, routing, broadcasting, distributed algorithm.**

I. Introduction

A new interconnection network topology called the star graph has been recently introduced in [1] and [2]. An extension to this network has also been introduced in [7]. The star graph is vertex symmetric. It provides an interconnection network for a large number of processors using a low number of communication channels while providing a high level of redundancy that makes it highly fault-tolerant [1]–[3]. An optimal algorithm for routing messages between any two nodes of the star graph, assuming that no faulty links or nodes exist in the graph, was presented in [1]. Fault-tolerance routing has been discussed for different interconnection networks [6], [8]. Fault tolerance of the star graph was discussed in [3] and [9]. Reference [2] compared properties of the hypercube and the star graphs. A depth first search approach to provide fault-tolerant routing in the hypercube was presented in [5]. Reference [9] presented a routing scheme using a depth first search approach on faulty star graphs. The scheme had shortcomings: by keeping the information about the traversed path in a stack that is popped every time a message backtracks, the algorithm does not guarantee liveness and deadlock-free transmission. In fact, an example can be found where the message gets stuck by being continuously sent to the same node. Due to faulty conditions, the node cannot forward the message. Once it returns the message, it receives it later because its node reference has already been popped off the stack.

In this correspondence, we use symmetry and fault-tolerance properties of the star graph to introduce a distributed algorithm for efficiently routing messages between any two nodes of the star graph in the presence of faulty links. The algorithm is based on the

Manuscript received June 8, 1992.

N. Baherzadeh and N. Nassif are with the Department of Electrical and Computer Engineering, University of California, Irvine, Irvine, CA 92717.

S. Latifi is with the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV 89154.

IEEE Log Number 9209050.