define the generation potential of I2 and I1 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 I2 and I1 (6.2 ns) plus the delay from the output of the mux at the of I1 to the new block's msb is limited by 11 ns. The resulting r2 is shown in Fig. 6(f).

Just like I1, 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 rl 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


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
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; $t_e$ is the transfer time for an element, and $\tau$ is the start-up for communication of a packet of maximum $B$ elements. For clarity, we first assume that $t_e = 1$ 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.

Notations and 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 $\{0, 1, \ldots, N' - 1\}$ in $Q_{n+1}$. In this correspondence, we only consider incomplete hypercubes of sizes $2^k + 2^k$, where $0 \leq k < n$. Such an incomplete hypercube, denoted $I_n^k$, consists of an $n$-cube and a $k$-cube. Throughout this correspondence, we use $N = 2^n$ and $K = 2^k$, and refer to $Q_n$ and $Q_n$ 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 $(x_n, x_{n+1}, \ldots, x_0)$. Thus, $(0^*^*)$ and $(0^*^{n-1}1^*)$ denote the front cube and back cube, respectively, where $*$ is a don't-care symbol whose value can be 0 or 1, and $y$'s 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 $I_n^k$. A respective subcube $S_i$ of $I_n^k$ is defined as $S_i = (x_n, \ldots, x_{n-k})$, such that the value of $(x_{n-k}, \ldots, x_0) = i$.

With distinct higher $(n-k)$-bit values, there are $2^{n-k} + 1$ respective subcubes. For convenience, we will also refer to the back cube as $S_{\tau}$ and interchangeably as $S_{n-1}$. 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 $I_3^1$.](image)

Fig. 1. Binomial spanning tree and binomial broadcasting in a 3-cube.

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

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 $I_3^1$ 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 $C_0$, $C_1$, and $C_2$, 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 $\binom{i}{0}^k$. 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], Johnson 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 $j$th spanning tree, where $j \in \{0, 1, \ldots, n-1\}$, the root first extends an edge across dimension $j$ to node $x = (0^n1\cdots10^j)$. Then, construct a spanning tree rooted at node $x$ in subtree $(x_{n-j}x_{n-j+1}^*)$ according to the sequence of dimensions $(j+1) \mod n, (j+1) \mod n, \ldots, (j+1) \mod n$. By connecting each node (but node $x$) in subcube $(x_{n-j}x_{n-j+1}^*)$ to its corresponding node in subcube $(x_{n-j}10^j)$ by a new edge, the $j$th 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.
An example of one-port broadcasting of 3 elements based on the nEDST 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 \(\lceil M/n \rceil + n\) steps, all but the links entering the root.

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 \(I^k_n\), 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 nEDST. 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 \(I^k_n\) in a similar way. 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 \(0 \leq j \leq k\), we attach nodes in \(S_1\) and preserve them edge-disjoint. Fig. 5 gives a global view of attaching nodes in \(S_0\) 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, 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 \(u_0\).

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 \(0 \leq j \leq k\), we attach nodes in \(S_0\). Fig. 6 gives a global view of appending a subtree. As before, leaf nodes along link \(k\) in the appended subtrees are deleted to generate a spanning tree in \(I^k_{n+1}\).

3) Root in \(S_b\): Since \(S_0\) 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 \(\{S_0, S_1\}\) with the root being in \(S_0\). 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 \(I^k_{n+1}\).

Proof: We give the proof of case 1 (root in \(S_1\)). 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 \(S_0\)) 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.
The diameter of the incomplete hypercube $I_n^k$ is $n + 1$. When the root is in a subcube other than $S_0$ or $S_{n-1}$, the furthest node away from the root is $i$, i.e., the distance of $i$-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 $S_0$ or $S_{n-1}$ is optimal.

**Theorem 2:** The maximum height of $k + 1$ EDST's in $I_n^k$ is optimal when the root is in $S_{n-1}$ or $S_0$, and $k > 0$.

**Proof:** If the root is in $S_0$, $k$ out of $k + 1$ neighbors of the root are in $S_0$. The further nodes away from nodes in $S_0$ 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 $S_{n-1}$, $r$ is in $S_1$, where $r$ is the node with complementing address bits of $r$. Clearly, node $r$ is $n + 1$ hops away from node $r$. Since each node in $S_0$ has $k + 1$ parent nodes, all incoming links of a node in $S_0$ are used in the EDST's. For neighbors of $r$ to them implies that the paths from $r$ through $r$ 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 $\alpha$. The higher $\alpha$ an algorithm attains, the better the algorithm performs. The other factor is the propagation delay for a packet—latency, denoted by $\beta$; 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 $[M/\alpha] + \beta - 1$.

**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
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 \(S_i\)) 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 \(S_{2n-k+1}\), the height of the \((k + 1)\)EDST is \(1 + (n - k - 1) + (k + 1) + 1 = n + 2\) (the nodes in \(S_i\) have one hop deeper than their corresponding nodes in \(S_i\)). Precisely, the height is \(x + 1\), where \(x\) is the longest distance from the source to other nodes in \(I^n\). 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.

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 \(r = 0\) and \(t_r = 1\). The general complexities (7), 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


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 under 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 \( \Theta(\sqrt{n}) \).

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. Bagherzadeh 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.