**Abstract.** Many parallel machines based on the interconnections of the boolean hypercube are now commercially available. A distinctive feature of the hypercube is its *universality* — programs written for simpler architectures, such as grids for example, can be transported onto the hypercube with minimal overhead. These simulations are typically obtained by embedding the simpler architecture within the hypercube.

This paper presents efficient embeddings of binary trees in the hypercube. We provide a novel and optimal embedding of a complete binary tree in which all but one tree edges are mapped onto adjacent processors on the hypercube, and the remaining edge is routed through an unused processor. With suitably designed processors and protocols, communication through the forwarding processor should incur no extra delay.

We also present efficient embeddings of binary trees that are not complete, and show that any $N$-node binary tree can be embedded with edges of length $\log\log N + O(1)$ in a hypercube with no more than $2N$ processors. Our results extend to graphs with small separators.

# How to Embed Trees in Hypercubes

Sandeep N. Bhatt, Ilse C.F. Ipsen

Research Report YALEU/DCS/RR-443
December 1985

This work was supported in part by the Office of Naval Research under contracts N000014-82-K-0184 and N00014-85-K-0461.

## 1. Introduction

The boolean cube ($n$-cube, hypercube) has emerged as a powerful and popular network for parallel computing. Following the 64-processor Cosmic Cube built at Caltech [19], a number of parallel hypercube machines have been designed and some are commercially available. Already, there is a choice between the 128-processor INTEL iPSC/d7, the 1024-processor NCUBE/Ten, the 64000-processor bit-sliced Connection Machine from Thinking Machines and the 256-processor Ametek/System 14 [6]. Similarly, the Shuffle-exchange and FFT networks, both closely related to the hypercube, have spawned the NYU Ultracomputer [18] and the 256-processor BBN/Butterfly [6].

The hypercube offers a rich interconnection structure with large bandwidth, logarithmic diameter, and is ideally suited for divide-and-conquer applications. More importantly, the hypercube supports efficient routing algorithms [1, 20] and can simulate every realistic architecture with small overhead. While the same is true of the shuffle-exchange and FFT networks which require fewer connections per processor, the hypercube has the advantage of being able to simulate networks such as grids with *no* communication overhead.

Two factors render the hypercube especially attractive in practice : efficient *broadcasting* and *simulation*. Efficient broadcasting of data among processors is crucial in applications such as Gaussian elimination with pivoting [10, 15], inner-product computations [13, 16], and eigenvalue problems [13]. Simulation of other networks with no overhead is especially convenient for reasons of 'portability' : programs written for such machines can be run directly on the hypercube. For instance, it has long been known that multi-dimensional grids of suitable dimensions can be embedded as subgraphs of the hypercube by means of gray codes [12]. As a consequence, many grid-based numerical algorithms can be efficiently implemented on the hypercube, examples include multi-grid methods for solving partial diffrential equations [5], and LU factorizations [10, 11]. Fox [9] presents a wide range of further scientific applications.

Binary trees constitute an important class of structures, frequently found in applications, as they capture the essence of divide-and-conquer strategies and serve as efficient data structures in hosts of applications [2]. Binary trees also arise in the solution of banded systems [14] and in the solution of sparse systems by direct elimination methods [7, 8]. A complete binary tree with $2^n - 1$ nodes can be embedded with edges of dilation at most two in a $2^n$-node hypercube, or with edges of dilation one in a $2^{n+1}$ node hypercube [15, 21]; embeddings with unit dilation in a $2^n$-node hypercube cannot exist [17, 21].

The binary trees generated by general divide-and-conquer methods and by all adaptive numerical methods are, in general, neither complete nor binary. To our knowledge, the problem of computing efficient embeddings for arbitrary bounded-degree trees has not been addressed. In this paper we present some preliminary results and fast algorithms for finding good embeddings.

Section 2 presents the best possible embedding of a complete binary tree within a hypercube of the same size: every pair (with a single exception) of nodes adjacent in the tree are also adjacent in the hypercube, while both nodes in the remaining pair are adjacent to the unused node. Hence, exactly one tree edge suffers dilation two. With suitably designed processors and protocols, communication through the forwarding processor should incur no delay. Section 3 considers embeddings of arbitrary binary trees within a hypercube. We show that every $N$-node binary tree can be embedded in a hypercube with $2N$ nodes such that every tree edge sufferes dilation at most $\log\log N + O(1)$. Conclusions and suggestions for further research are offered in Section 4.

## 2. Optimal Embeddings for Complete Binary Trees

This section presents the best way to embed a complete binary tree with $2^n - 1$ vertices within a hypercube with $2^n$ vertices. We also note a number of implications of our embedding. Before proceeding with the main result, we establish a few preliminary definitions.

**Definition 2.1.** A *hypercube* of dimension $n$ (or *n-cube*) is an undirected graph consisting of $2^n$ vertices addressed $0 \ldots 2^n - 1$. Two vertices are connected by an edge if and only if the binary representations of their addresses differ in exactly one bit position.

Let $\delta_j$ be the binary vector of length $n$ with a 1 in position $j$ and zeroes in all other positions. If $u$ and $v$ are the addresses of two vertices in the hypercube then $u$ and $v$ are adjacent if and only if $u = v \oplus \delta_j$ ($\oplus$ stands for the componentwise *xor*-operation). It is easily seen that each vertex in an $n$-cube has degree $n$ and that the hypercube has no cycles of odd length.

**Definition 2.2.** $C_n$ is the complete binary tree with $2^n - 1$ nodes and height $n$.

**Definition 2.3.** An *embedding* of $C_n$ into the $n$-cube is a 1-1 mapping of the nodes of the tree into the hypercube. The *dilation* of an edge $(u, v)$ in the tree equals the Hamming distance between the images of $u$ and $v$ under the embedding. The *dilation* of the embedding equals the maximum dilation over all edges.

The following theorem establishes that $C_n$ cannot be embedded in the $n$-cube with unit dilation. In other words, $C_n$ is not a subgraph of the $n$-cube. The proof below, suggested by Stan Eisenstat and Mike Fischer, simplifies previous proofs in [17, 21].

**Theorem 2.1.** *For $n \geq 3$, $C_n$ cannot be embedded in an $n$-cube with unit dilation.*

*Proof.* If $C_n$ were a subgraph of the $n$-cube, then $n - 1$ edges must be added to each leaf of $C_n$ because the degree of each node in the hypercube is $n$. In all, therefore, $(n-1)2^{n-1}$ edges need to be added with one end-point per leaf. Let us call the other end-point of each of these edges a 'receptor'.

Since the hypercube does not have odd cycles, no leaf can also be a receptor. Similarly, no grandparent of a leaf can be a receptor. Therefore, the number of receptors is at most $2^n - 2^{n-1} - 2^{n-3}$. One of these receptors (the unused node in the hypercube) can accept $n$ edges, the root can accept $n - 2$ and every other receptor can accept $n - 3$.

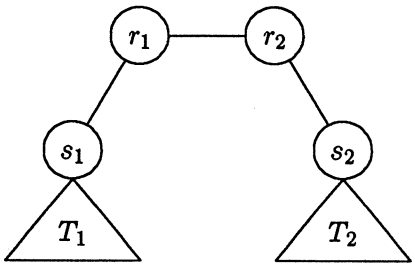
The total number of edges that can be accepted by the receptors is therefore no greater than $(n-3)(2^n - 2^{n-1} - 2^{n-3} - 2) + n - 2 + n$. For $n \geq 3$ this quantity is strictly less than the number of edges emanating from the leaves, which means that the edges cannot be added as desired. ∎

**Theorem 2.2.** *For every $n$, $C_n$ can be embedded within the $n$-cube with one edge of dilation two, and every other edge of dilation one.*

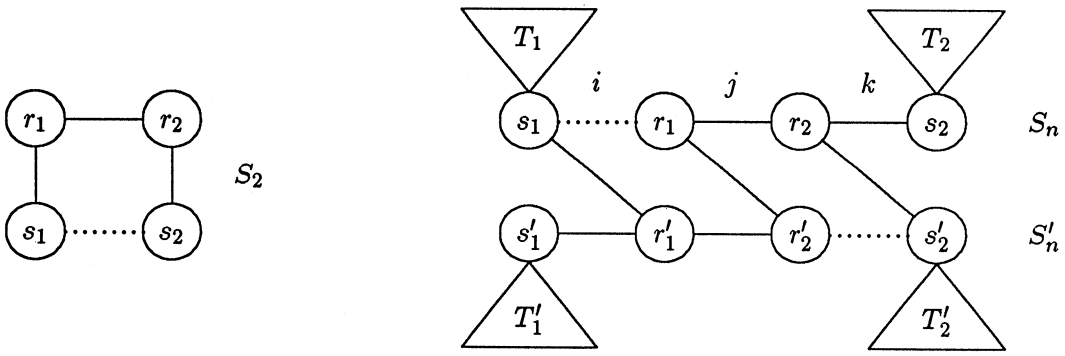
*Proof.* We claim that the $n$ vertex 'two-rooted' tree of Figure 1(a) (formed by adding an extra vertex between the root and one child of $C_n$ can be embedded with unit-dilation in the $n$ cube. The theorem follows directly thereafter.

To prove the claim for two-rooted trees, we perform an induction on the size of the tree. Figure 1(b) shows a unit-dilation embedding of the two-rooted tree $S_2$ with 4 nodes within the 2-cube. Assume inductively that the two-rooted tree $S_n$ with $2^n$ nodes can be embedded with unit-dilation in the $n$ cube.

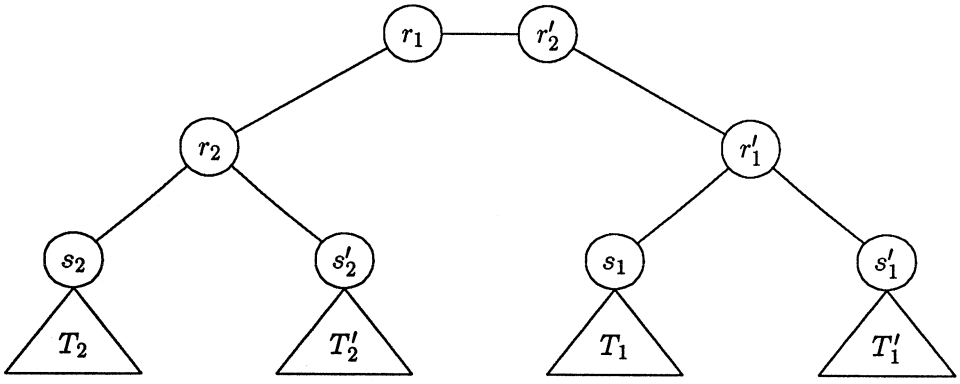
For the inductive step, consider two copies of a unit-dilation embedding of $S_n$ in an $n$-cube, as in Figure 1(b). Without loss of generality, assume that node $r_1$ is at the origin of its subcube (that is, it has an address consisting of all zeroes). Translate the lower embedding by inverting the $j$th




(a) A Two-Rooted Tree.




(b) Recursive Construction of $S_{n+1}$.




(c) Levels of $S_{n+1}$.

**Figure 1:** Two-Rooted Trees.

bit of each node. Formally, if a tree node was embedded at address $u$ then after the translation the tree node will appear at address $u \oplus \delta_j$. After the translation node $r_2'$ will be at address $r_2 \oplus \delta_j = r_1$, i.e, $r_2'$ and $r_1$ are at the origins of their respective $n$-cubes after the shift. Observe that the dilation is invariant under translations.

Next, rotate the embedding in the lower cube by permuting (for every node) the address bits in positions $i, j, k$ into positions $k, i, j$. While $r_2'$ and $r_1$ remain fixed, $r_1'$ is rotated so that its address in the lower cube is identical to the address of $s_1$ in the top cube. Similarly, $r_2$ and $s_2'$ have identical addresses within their respective subcubes.

Combining the two $n$-cubes into an $(n+1)$-cube, vertices in each of the pairs $(s_1, r_1')$, $(r_1, r_2')$, $(r_2, s_2')$ are adjacent because their addresses differ in exactly the new dimension. By making $r_1'$ and $r_2$ the roots of larger complete binary trees as in Figures 1(b) and (c), $r_1$ and $r_2'$ become the roots of a new two-rooted tree $S_{n+1}$. Figure 1(c) shows the larger two-rooted tree constructed inductively. To complete the proof, observe that each edge connecting the two embeddings is an edge in the $(n+1)$-cube resulting in a unit-dilation embedding of $S_{n+1}$.

∎

**Remarks.**

1. By embedding the two-rooted tree within a hypercube of the same size we have implicitly embedded two disjoint complete binary trees each of size $2^{n-1} - 1$ with unit dilation in an $n$-cube. This strengthens the unit dilation embeddings of trees in hypercubes of twice the size reported in [15, 21].
2. In embedding the complete binary tree $C_n$ within an $n$-cube observe that the single edge with dilation two from the root to one of its children passes through an unused vertex. By suitably designing processors so that an unused processor can be switched to directly link two of its neighbors, the delay across the two links should not, in practice, be noticeably larger than the delay across a single link.
3. Every node of the embedded complete binary tree is at most distance 4 from its 'reflections', nodes at the same level whose path to the lowest common ancestor differs only at the highest level. Consequently, if we insert edges for nodes that are Hamming distance 4 apart, the nodes at a level form a hypercube. This property can be exploited in algorithms where communication occurs along the levels of tree in addition to tree edges.

## 3. Embedding Arbitrary Binary Trees

While the recursive embedding of the previous section gives the best possible embedding of a *complete* binary tree, it does not provide much insight into efficient embeddings of *arbitrary* binary trees. In this section, we examine embeddings of arbitrary binary trees derived from recursive strategies similar to those used in VLSI layouts of tree structures [4]. We present two different strategies: in the first we embed trees with unit dilation in a polynomially larger hypercube, while in the second we increase the dilation but reduce the size of the hypercube. Thus, dilation is traded for expansion, where 'expansion' refers to the size of the hypercube relative to the size of the tree embedded.

**Theorem 3.1.** *Every $N$-node binary tree can be embedded with unit dilation in a hypercube of $O(N^{1.71})$ nodes.*

*Proof.* We use the well-known result that any $N$ node binary tree can be separated into two disconnected trees, each containing no more than $\lceil 2N/3 \rceil$ nodes, by removing a single edge, say, $(u, v)$. Recursively embed the two smaller trees (with unit dilation) within two smaller hypercubes. Translate one of the cubes so that, when connecting nodes in corresponding positions of the two

cubes (and thus forming a cube of twice the size), there is an edge between $u$ and $v$. Of course, the embeddings within the smaller cubes are unaffected by the translation and every edge in the embedding has unit dilation.

If $D(N)$ is the number of hypercube dimensions used in a recursive embedding of an $N$-node tree, then

$$D(1) = 0, \qquad D(N) \leq D(\lceil 2N/3 \rceil) + 1,$$

which yields $D(N) \leq \log_{3/2} N$. The number of nodes in the hypercube is therefore no greater than $2^{D(N)} \leq N^{\log_{3/2} 2} = O(N^{1.71})$.

∎

Although Theorem 3.1 gives a unit dilation embedding, the expansion proportional to $N^{.71}$ is too large to be useful. At this point it is not clear whether this result can be improved, and, in particular, whether it is possible to embed every $N$-node binary tree in an $O(N)$-node hypercube with unit dilation. Bisecting the tree by means of a more sophisticated recursive strategy reduces the expansion to a minimum of $2^{\lceil \log N \rceil}/N$, but at the expense of increasing the dilation to $\log\log N + O(1)$. We next show how to embed every $N$-node binary tree in a hypercube with expansion 2 and dilation $\log\log N + O(1)$. While this embedding is primarily of theoretical interest, we expect that the bounds can be reduced considerably and made useful in practice.

This improved embedding is based on techniques developed in [4] for VLSI layout : the nodes of an arbitrary binary tree $T$ are mapped onto a smaller complete binary tree $C$ so that not 'too many' nodes of $T$ are mapped onto the same node of $C$ and, at the same time, any two adjacent nodes of $T$ are 'close' in $C$. This problem was solved in [4] for classes of graphs harder to separate than binary trees, but the same technique can be applied to binary trees as well.

The main result necessitates a few preliminary lemmas.

**Lemma 3.1.** *Let $T$ be an $N$-node binary tree with one of $k$ colors assigned to each node. Let $n_i$ be the number of vertices of color $i$, $1 \leq i \leq k$, $\sum_i n_i = N$. By removing $k \log N$ or fewer edges, $T$ can be bisected into two components of size $\lfloor N/2 \rfloor$ and $\lceil N/2 \rceil$, respectively, such that each component has at least $\lfloor n_i/2 \rfloor$ vertices of color $i$, $1 \leq i \leq k$.*

*Proof.* Follows directly from [4].

∎

**Lemma 3.2.** *Every $N$-node binary tree $T$ can be mapped onto an $N$-node complete binary tree $C$ so that*

1. *at most $6 \log (N/2^t) + 24$ nodes of $T$ are mapped onto any node of $C$ at level $t$ (distance $t$ from the root),*
2. *any two nodes adjacent in $T$ are mapped to nodes at most distance 3 apart in $C$.*

*Proof.* For convenience we assume that $N$ is a power of 2. The idea is to recursively bisect $T$, placing successive sets of bisector nodes (nodes whose removal bisects the tree) within successively lower levels of $C$, until $T$ is decomposed into single nodes. Specifically, the nodes placed at the root of $C$ bisect $T$ into two subgraphs $T_1$ and $T_2$. Similarly, nodes mapped to the left child of the root bisect $T_1$ , while the ones mapped to the right child bisect $T_2$. In addition, level $t$ of $C$ will also receive nodes of $T$ which

1. have not already been mapped to levels $t-1$ and $t-2$ of $C$, and
2. are adjacent to nodes at level $t-3$ of $C$.

Hence, adjacent nodes in $T$ are mapped to nodes of $C$ that are at most distance 3 apart.

The corresponding procedure is given below. The application of Lemma 3.1 with three colors ensures that the number of nodes of $T$, mapped to a single node of $C$, is bounded as required.

*Step 1.* Initialize every node of $T$ to color 0; bisect $T$; and place the bisector nodes at the root (level 1) of $C$.

*Step 2.* For each subgraph created in Step 1, recolor with color 1 nodes adjacent to the bisector of Step 1; and place a 2-color bisector for the subgraph at the corresponding level 2 node of $C$.

*Step 3.* For each subgraph created in Step 2, recolor with color 2 nodes adjacent to the bisector of Step 2; and place a 3-color bisector for the subgraph at the corresponding level 3 node of $C$.

*Step t.* $4 \leq t \leq \log |T|$, for each subgraph created in the previous step, place every node of color $t(\mathrm{mod}\, 3)$ at the corresponding level-$t$ node of $C$; recolor with color $t(\mathrm{mod}\, 3)$ nodes of color 0 that are adjacent to those of color $t(\mathrm{mod}\, 3)$; and place a 3-color bisector for the remaining subgraph at the corresponding level t node of $C$.

If $n_t$ is the maximum number of nodes mapped to a level $t$ node of $C$, then from Lemma 3.1

$$n_1 = \log N, \quad n_2 = 2\log N/2, \quad n_3 = 3\log N/4,$$

and the use of a 3-color bisector at each step implies in general

$$n_t \leq 3\log \frac{N}{2^t} + \frac{1}{2} n_{t-3}$$

The solution to this recurrence yields the desired result.

∎

Observe that the decomposition of Lemma 3.2 can be obtained in time polynomial in the size of $T$, since a $k$-color bisector for a tree can be found in polynomial time if $k$ is fixed.

Upon completion of the embedding of Lemma 3.2, the complete binary tree $C$ is truncated after $\log N - \log 6 \log N$ levels; the resulting tree consists of a total of $\frac{1}{3} N \log N$ nodes. Each leaf of this truncated tree contains, in addition to the nodes mapped onto it, the nodes of the incident subtree. Consequently, no more than $6 \log N$ nodes are mapped onto any leaf. A simple calculation also shows that no other node has more than $6 \log N$ nodes mapped onto it. This establishes the following result.

**Corollary 3.1.** *Every $N$-node binary tree $T$ can be mapped onto a $\frac{N}{3 \log N}$-node complete binary tree $C$ so that*

1. *at most $6 \log N$ nodes of $T$ are mapped onto any node of $C$, and*
2. *any two nodes adjacent in $T$ are mapped to nodes at most distance 3 apart in $C$.*

Finally, we proceed to the main result.

**Theorem 3.2.** *Every $N$-node binary tree $T$ can be embedded in a $2^{1+\lceil \log N \rceil}$-node hypercube with dilation $9 + \log \log N$.*

*Proof.* Use Corollary 3.1 to obtain a mapping of the $N$-node binary tree $T$ in the $\frac{N}{3 \log N}$-node complete binary tree $C$ so that every node of $C$ contains at most $6 \log N$ nodes of $T$. Next, embed $C$ within a hypercube of $\log N - \log(3 \log N)$ dimensions and dilation 2, using the optimal strategy of the previous section. Finally, expand each node of $C$ into a subcube of $\log(6 \log N)$ dimensions, and embed the $6 \log N$ nodes mapped onto any node of $C$ into the subcube.

To determine the maximal dilation, consider an edge $(u, v)$ in $T$. Either $u$ and $v$ are mapped onto the same node of $C$, or else they are mapped to nodes of $C$ which are distance 3 apart. In the first case, $u$ and $v$ are mapped within the same subcube of dimension $\log(6 \log N)$, so the dilation of the edge $(u, v)$ is certainly no greater than $\log 6 + \log\log N$. In the second case, the subcubes containing $u$ and $v$ are connected by a path of length at most six (the nodes in $C$ containing $u$ and $v$ are at a distance of at most 3, and each edge in $C$ has dilation 1 or 2). Traversing each dimension within the subcubes at most once, and the dimensions along the path connecting the subcubes, shows that there must be a path of length at most $6 + \log(6 \log N) \leq 9 + \log\log N$ connecting $u$ and $v$.

The embedding of $C$ requires a hypercube of dimension $1 + \log N$, $\log N - \log(3 \log N)$, and a hypercube of dimension $\log 6 \log N$ for each node within $C$. The number of nodes in the final hypercube therefore comes to at most $2N$.

∎

## 4. Conclusions

The primary contribution of this paper is the optimal embedding for complete binary trees presented in Section 2. This embedding is likely to have considerable impact in practice. The results of Section 3, while primarily theoretical in nature, represent the first nontrivial results on embedding graphs other than grids in the hypercube. More significantly, Section 3 raises a number of interesting questions.

First, can every binary tree be embedded in a hypercube with dilation 2 and unit expansion? We know of no binary trees that could be potential counterexamples. Recently, the technique of Section 3 has been extended to show that every binary tree can be embedded with dilation 10 and expansion 4 [3]. It would be satisfying to obtain the tightest result, while using a less involved technique than in Section 3. A related issue concerns the development of faster algorithms to embed trees efficiently.

The results of Section 3 extend to graphs more general than binary trees. In particular, the result is valid for every graph with a $O(1)$ separator theorem. This includes all trees of bounded degree as well as bounded degree outerplanar graphs. What other classes of graphs can be efficiently embedded in hypercubes? Especially important in practice are embeddings of bounded degree planar graphs.

A number of other engineering issues remain to be formally studied. For example, what is a good strategy for partitioning a large tree onto a small hypercube so that the load is equally shared among processors? In many applications, the tree structure grows and shrinks dynamically, and only the leaves are active. Are there good strategies for maintaining dynamic trees? And finally, a critical concern in the context of parallel processing is the need to develop efficient embedding techniques that are able to tolerate faults.

## 5. Acknowledgements

Stan Eisenstat and Mike Fischer suggested the simpler proof of Theorem 2.1. We thank them, and also Bill Gropp and Lennart Johnsson for helpful discussions.

## References


[1] Batcher, K., Sorting Networks and their Applications, *Proc. Spring Joint Computer Conf.,* AFIPS Press, 1968, pp. 307–14.

[2] Bentley, J.L. and Kung, H.T., A Tree Machine for Searching Problems, *Proc. Int. Conf. Parallel Processing,* IEEE, 1979, pp. 257–66.

[3] Bhatt, S.N., Chung, F.R.K., Leighton, T. and Rosenberg, A., *Optimal Embeddings of Binary Trees in the Boolean Hypercube,* Research Report in Preparation, Dept Computer Science, Yale University, 1985.

[4] Bhatt, S.N. and Leighton, F.T., *A Frame Work for Solving VLSI Graph Layout Problems,* JCSS, 28 (1984), pp. 300–43.

[5] Chan, T.F. and Saad, Y., *Multigrid Algorithms on the Hypercube Multiprocessor,* Research Report 368, Dept Computer Science, Yale University, 1985.

[6] Dongarra, J.J. and Duff, I.S., *Advanced Architecture Computers,* Technical Memorandum 57, Argonne National Laboratory, 1985.

[7] Duff, I.S., *Parallel Implementation of Multifrontal Techniques,* Technical Memorandum 49, Argonne National Laboratory, 1985.

[8] Eisenstat, S.C., Schultz, M.H. and Sherman, A.H., Applications of an Element Model for Gaussian Elimination, *Sparse Matrix Computations,* Academic Pres, 1976, pp. 85–96.

[9] Fox, C.F., Concurrent Processing for Scientific Calculations, *IEEE Comp. Sci. Conf. COMPCON,* 1984, pp. 70–3.

[10] Geist, G.A., *Efficient Parallel LU Factorization with Pivoting on a Hypercube Multiprocessor,* Technical Report ORNL 6211, Oak Ridge National Laboratory, 1985. Submitted for Publication.

[11] Geist, G.A. and Heath, M.T., *Parallel Cholesky Factorization on a Hypercube Multiprocessor,* Technical Report ORNL 6190, Oak Ridge National Laboratory, 1985. Submitted to *Parallel Computing.*

[12] Gilbert, E.N., *Gray Codes and Paths on the N-Cube,* The Bell System Technical Journal, (May 1958).

[13] Ipsen, I.C.F. and Jessup, E., *Solving the Symmetric Tridiagonal Eigenvalue Problem on the Hypercube,* Research Report in Preparation, Dept Computer Science, Yale University, 1985.

[14] Johnsson, S.L., *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution of Tridiagonal Systems of Equations,* Research Report 339, Dept Computer Science, Yale University, 1985.

[15] ———, *Communication Efficient Linear Algebra Computations on Hypercube Architectures,* Research Report 361, Dept Computer Science, Yale University, 1985.

[16] Saad, Y. and Schultz, M.H., *Data Communication in Hypercubes,* Research Report 428, Dept Computer Science, Yale University, 1985.

[17] ———, *Some Topological Properties of The Hypercube Multiprocessor,* Research Report 389, Dept Computer Science, Yale University, 1984.

[18] Schwartz, J.T, *Ultracomputers,* ACM TOPLAS, 2 (1980), pp. 484–521.

[19] Seitz, C.L., *The Cosmic Cube,* CACM, 28 (1985), pp. 22–33.

[20] Valiant, L.G., *A Scheme for Fast Parallel Communication,* SIAM J. Comp., 11 (1982), pp. 350–61.

[21] Wu, A.Y., *Embedding of Tree Networks into Hypercubes,* Jour. Par. Distr. Comp., 2 (1985), pp. 238–49.