TR 361

JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 4, 133–172 (1987)

Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures*

S. LENNART JOHNSSON

Departments of Computer Science and Electrical Engineering, Yale University, New Haven, Connecticut 06520-2167

Received October 4, 1985

This paper presents a few algorithms for embedding loops and multidimensional arrays in hypercubes with emphasis on proximity preserving embeddings. A proximity preserving embedding minimizes the need for communication bandwidth in computations requiring nearest neighbor communication. Two storage schemes for "large" problems on "small" machines are suggested and analyzed, and algorithms for matrix transpose, multiplying matrices, factoring matrices, and solving triangular linear systems are presented. A few complete binary tree embeddings are described and analyzed. The data movement in the matrix algorithms is analyzed and it is shown that in the majority of cases the directed routing paths intersect only at nodes of the hypercube allowing for a maximum degree of pipelining.

1. INTRODUCTION

Many of the conventional linear algebra problems use rows, columns, or diagonals as aggregate data structures, and algorithms are often formulated in terms of operations thereupon [7, 34, 35]. One-, two-, or multidimensional arrays are typical data structures. A large number of algorithms explicitly acknowledge this regularity in the form of vector algorithms for pipelined SIMD (Single Instruction Multiple Data streams) [9] architectures. Many algorithms have also been devised with the vector concept for SIMD multiprocessors configured as meshes, such as the ILLIAC IV, the ICL DAP, and the MPP [17, 16]. Vectors are also the predominant data structure in systolic

* This work was supported by the Office of Naval Research under Contract N00014-84-K-0043.

0743-7315/87 $3.00

algorithms for linear algebra computations. Many "fast" matrix algorithms and equation solvers make use of some type of divide-and-conquer strategy and tree-like data structures may be preferable to arrays.

In recent years multiprocessors of the MIMD (Multiple Instruction stream Multiple Data stream) type and of moderate to medium concurrency have been designed and built. Some of these architectures are bus oriented, others have the Ultracomputer [39, 11], or the Cosmic Cube [40] as prototype architectures. Both these architectures are extensible to highly concurrent systems. In the Cosmic Cube prototype the storage is completely distributed among the processing nodes, which are interconnected as a boolean cube. In the Ultracomputer, processors and storage are at opposite ends of a switching network. The processors execute their own instruction streams in both designs and are of the complexity of conventional microprocessors. Another prototype architecture for ultraconcurrent systems is the Connection Machine [12, 13]. This architecture has many of the characteristics of architectures capable of exploiting the technology of the future. The storage is distributed among nodes, as in the Cosmic Cube, but the processors are 1-bit processors with 16 such processors per chip. The chips are interconnected as a 12-cube. The programming model is of the SIMD variety. We refer to the highly parallel architectures as *ensemble architectures.*

In this paper we focus on boolean cube configured ensemble architectures and present graph embeddings associated with basic linear algebra algorithms. Some topological properties of boolean cubes are presented in Section 2. In Section 3 the embedding of loops and multidimensional arrays are discussed, and some important properties of *proximity* preserving embeddings by a *binary-reflected* Gray code [36] are derived. The communication complexity of converting from binary code to Gray code is analyzed. Section 3 also addresses some of the questions arising when the number of nodes in the arrays to be embedded in the cube exceeds the number of nodes in the cube. Finally, Section 3 contains algorithms for a few tree embeddings. Section 4 analyzes the routing paths and communication complexity in transposing a matrix. Section 5 contains algorithms for multiplication of dense matrices and Section 6 contains algorithms for the solution of dense systems of equations by direct methods. The analysis is focused on the routing paths of matrix elements. Section 6 considers both the factorization phase and the solution of triangular systems of equations.

2. HYPERCUBES

There are 2^n nodes in an n-dimensional boolean cube. There are two coordinate points in each dimension. The nodes can be given addresses such that the addresses of adjacent nodes differ in precisely one bit. The boolean cube

is a recursive structure. An n-dimensional cube can be extended to an $(n + 1)$-dimensional cube by connecting corresponding vertices of two n-dimensional cubes. One has the highest-order address bit 0 and the other the highest-order bit 1. The recursive nature of the boolean cube is illustrated in Fig. 1.

Each node has n neighbors. The maximum distance between an arbitrary pair of nodes is n and the average distance is $n/2$. The number of nodes at distance k from a node is $\binom{n}{k}$. The total number of internode connections is $n2^{n-1}$. There are n disjoint paths between any pair of processors. Of these paths k are of length k and $n - k$ of length $k + 2$ [37]. A measure of the wiring complexity of a boolean cube is the area required for a planar layout. In the Thompson grid model [42, 43] for (planar) layout, a boolean n-cube can be laid out in area $O(2^{2n})$. The maximum wire length is of order $O(2^n)$ [31].

3. Embedding of Data Structures

One-, two-, or multidimensional arrays are natural data structures for matrix problems, iterative methods for systems of equations, and a variety of problems in computational physics. Four-dimensional grids are used for quantum electrodynamics and quantum chromodynamics computations. Nearest neighbor communication suffices in such data structures. Periodic boundary conditions are easily realized by local communication if the arrays have end-around connections. Other types of algorithms may employ some form of divide-and-conquer strategy and tree-like data structures and proces-

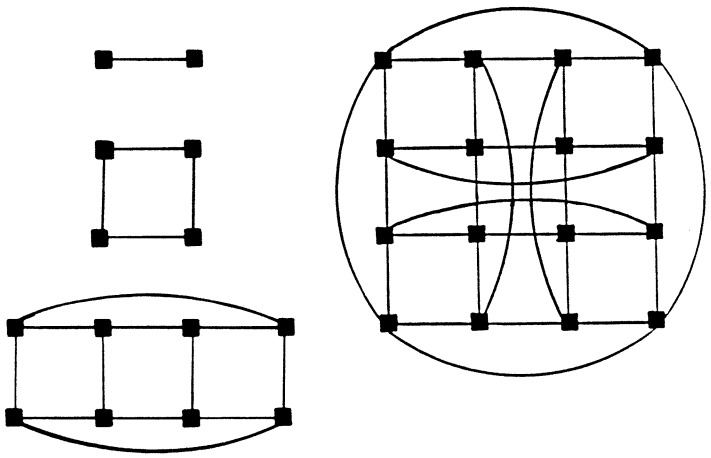

FIG. 1. Boolean n-cubes for $n = 1$–4.

sor connectivity would offer computations with local communication. Dense matrix multiplication and factorization algorithms (LU-decomposition, Cholesky factorization, Given's method, etc.) are examples of algorithms that can be mapped effectively onto one- and two-dimensional grids, and combinations of such grids and trees (for instance, the mesh-of-trees configuration [31]). The Fast Fourier Transform, bitonic sort [2], parallel prefix computations, recursive doubling [28], odd–even cyclic reduction [21], nested dissection [10], and multigrid methods are examples of computations making use of divide-and-conquer strategies and data structures in the form of trees, or combinations thereof such as the familiar butterfly network, or combinations of trees and meshes in the form of a hierarchy of grids.

3.1. *Loop Embeddings*

3.1.1. *Binary Encoding*

Embedding a loop L of length $|L| = 2^n$ in an n-cube by a binary encoding of the indices of the nodes in the loop, assuming that successive nodes are numbered 0–L–1, does not preserve proximity. For instance, nodes $2^{n-1} - 1$ and 2^{n-1} differ in all the bits in their binary encoding, and hence the corresponding nodes of the loop are at a distance of n. The binary encoding has the property that if $i = (i_{n-1} i_{n-2} \cdots i_{k+1} 0 i_{k-1} \cdots i_1 i_0)$ and $j = i + 2^k$ then clearly i and j differ only in bit k and hence are adjacent. Note, however, that if $i = (i_{n-1} i_{n-2} \cdots i_{k+1} 1 i_{k-1} \cdots i_1 i_0)$, then the distance between i and $j = i + 2^k$ equals the length of the carry propagation path. The binary encoding is feasible for many divide-and-conquer computations such as the FFT, bitonic sort, and recursive doubling.

3.1.2. *Gray Code Encoding*

A loop embedding that preserves proximity is easily obtained for $|L| = 2^n$ by encoding the indices of the nodes in the loop in a *binary-reflected* Gray code [36]. We will make use of the following two alternative definitions of the binary-reflected Gray code. Let the n-bit code of 2^n integers be $G(n)$ and represented in matrix form as

$$G(n) = \begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ G_{2^n-2} \\ G_{2^n-1} \end{pmatrix}.$$

Then

$$G(n+1) = \begin{pmatrix} 0G_0 \\ 0G_1 \\ \vdots \\ 0G_{2^n-2} \\ 0G_{2^n-1} \\ 1G_{2^n-1} \\ 1G_{2^n-2} \\ \vdots \\ 1G_1 \\ 1G_0 \end{pmatrix}, \quad \text{or alternatively,} \quad G(n+1) = \begin{pmatrix} G_0 0 \\ G_0 1 \\ G_1 1 \\ G_1 0 \\ G_2 0 \\ G_2 1 \\ \vdots \\ G_{2^n-1} 1 \\ G_{2^n-1} 0 \end{pmatrix}.$$

The Gray codes of successive integers differ in precisely 1 bit. Let $T(n)$ be the sequence of bit indices on which a transition takes place in proceeding from integer 0 to integer $2^n - 1$ in the n-bit code with the most significant bit labeled $n - 1$. Then, the binary-reflected Gray code can be defined through the recursion $T(n+1) = T(n), t_n, T(n)$. Furthermore, let the binary encoding of $i = (i_n i_{n-1} i_{n-2} \cdots i_1 i_0)$ and the Gray code encoding be $G_i = (g_n g_{n-1} g_{n-2} \cdots g_1)$. Then the encoding and decoding is defined by $g_j = (i_j + i_{j-1}) \bmod 2$ and conversely $i_j = (\Sigma_{k=j+1}^{n} g_k) \bmod 2$.

Let $d_L(i, j)$ denote the distance between nodes i and j in the loop and $d_C(G_i, G_j)$ denote the distance between nodes i and j when embedded in the cube by a binary-reflected Gray code encoding. The quantity $\max_i d_C(G_i, G_{i+1})$ is called *dilation*.

LEMMA 3.1. *Any loop of length* $|L| = 2^{n-1} + 2k$, $k = \{1, 2, \ldots, 2^{n-2}\}$, *can be embedded in an n-cube with dilation* 1 *by a binary-reflected Gray code.*

Proof. Let nodes $\{0, 1, \ldots, 2^{n-1} + k - 1\}$ of the loop be embedded according to the binary-reflected Gray code of the loop node index, and loop nodes $\{2^{n-1} + k, 2^{n-1} + k + 1, \ldots, 2^{n-1} + 2k - 1\}$ be embedded in the cube nodes corresponding to the Gray codes of $\{2^n - k, 2^n - k + 1, \ldots, 2^n - 1\}$. The Gray code of $2^{n-1} + k - 1$ is $(1G(n-1)_{2^{n-1}-k})$ and the Gray code of $2^n - k$ is $(1G(n-1)_{k-1})$. But $G(n-1)_{k-1} = (0G(n-2)_{k-1})$ and $G(n-1)_{2^{n-1}-k} = (1G(n-2)_{k-1})$ by construction of the binary-reflected Gray code. ■

Remark. The embedding is not unique.

The embedding used in the proof is shown below for a loop of length $2^n + 2$.

- G_0
- $0G_1$
- $\vdots$
- $0G_{2^{n-2}}$
- $0G_{2^{n-1}}$
- $1G_{2^{n-1}}$

  $1G_{2^{n-2}}$

  $\vdots$

  $1G_1$
- $1G_0$

LEMMA 3.2. *Any loop of length* $|L| = 2^{n-1} + 2k + 1$, $k = \{0, 1, \ldots, 2^{n-2} - 1\}$, *must have at least one edge of length* 2 *when embedded in a boolean cube. Embedding loop nodes* $\{0, 1, \ldots, 2^{n-1} + k\}$ *according to their Gray codes and loop nodes* $\{2^{n-1} + k + 1, 2^{n-1} + k + 2, \ldots, 2^{n-1} + 2k\}$ *in the cube nodes corresponding to the Gray codes of* $\{2^n - k, 2^n - k + 1, \ldots, 2^n - 1\}$ *yields an embedding in which all but one edge are of length* 1, *and one edge is of length* 2.

Proof. The first part of the lemma is shown easily by contradiction. Assume that all edges are of length 1. Then in traversing the loop from any node back to itself, an odd number of bit transitions is experienced in the address of the starting node, which clearly must be an address different from that of the starting address. The proof of the second part follows that of Lemma 3.1. ■

Another useful property of the binary-reflected Gray code is the following.

LEMMA 3.3. *The binary-reflected Gray code encoding of* i *and* $j = (i + 2^k) \bmod 2^n$, $k > 0$, *differs in precisely* 2 *bits.*

Proof. Let the binary encoding of i be $(i_n i_{n-1} \cdots i_0)$ and that of j be $(j_n j_{n-1} \cdots j_0)$. Furthermore, let the Gray code of i be $G(n) = (g_n g_{n-1} \cdots g_1)$ and that of j be $H(n) = (h_n h_{n-1} \cdots h_1)$. Then $i_m = j_m$, $m = \{0, 1, \ldots, k-1\}$ and $j_m = \bar{i}_m$, $m = \{k, k+1, \ldots, s\}$, where $s \geq k$ is the bit where the carry stops propagating. It follows from the encoding formula that $h_m = g_m$, $m = \{1, 2, \ldots, k-1, k+1, \ldots, s\}$, and $h_k = \bar{g}_k$, $h_{s+1} = \bar{g}_{s+1}$. ■

COROLLARY 3.1. *Nearest neighbor communication on hierarchical, regular grids, obtained by omitting every other grid point for successively coarser grids requires communication over two edges for all but the finest grid.*

COROLLARY 3.2. *All but one butterfly in a butterfly network embedded by Gray code encoding requires communication across two edges.*

The property of binary-reflected Gray codes stated in Lemma 3.3 and Corollaries 3.1 and 3.2 is important for algorithms such as nested dissection, multigrid methods, the FFT, bitonic sort, recursive doubling, and cyclic reduction.

3.1.3. *Conversion between Binary Encoding and Binary-Reflected Gray Code Encoding*

Application programs typically include the use of several different "elementary" algorithms. Different embeddings may be optimal for different phases of a computation. The rearrangement of the data from one embedding to another may be preferable compared to using a nonoptimal embedding for part of the computations. By construction, the highest-order bits in the binary-reflected Gray code encoding of an integer and its binary encoding coincide. The encodings of the integer $2^n - 1$ differ in $n - 1$ bits, and the maximum routing distance is $n - 1$ interprocessor links. To carry out the conversion in $n - 1$ routing steps no two elements must compete for the same communications link at any given time. For the transformation to be pipelinable it is necessary that the source–destination paths be edge disjoint. An element needs to be routed in dimension j if $g_j \oplus b_j = 1$.

LEMMA 3.4. *A Gray code encoding can be rearranged to a binary encoding in n-1 routing steps.*

Proof. The proof is by induction. The lemma is clearly true for a 1-cube. Assume it is true for a k-cube. Then for a $(k + 1)$-cube the Gray codes $G(k + 1)$ of the processors with addresses encoding the integers $\{0, 1, \ldots, 2^k - 1\}^T$ and the integers $\{2^k, 2^k + 1, \ldots, 2^{k+1} - 1\}^T$ are

$$\begin{pmatrix} 0G(k)_0 \\ 0G(k)_1 \\ \vdots \\ 0G(k)_{2^k-2} \\ 0G(k)_{2^k-1} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1G(k)_{2^k-1} \\ 1G(k)_{2^k-2} \\ \vdots \\ 1G(k)_1 \\ 1G(k)_0 \end{pmatrix}$$

The highest-order bit in the Gray code encoding and in the binary encoding coincide, and the k lowest-order bits of $G(k + 1)$ for the integers $\{0, 1, \ldots, 2^k - 1\}$ coincide with $G(k)$, which can be converted to binary code in $k - 1$ routing steps according to the induction assumption. By an exchange operation in dimension $k - 1$ (dimensions are labeled 0–k), the conversion problem for the integers $\{2^k, 2^k + 1, \ldots, 2^{k+1} - 1\}$ becomes that of converting

$$\begin{pmatrix} 1G(k)_0 \\ 1G(k)_1 \\ \vdots \\ 1G(k)_{2^k-2} \\ 1G(k)_{2^k-1} \end{pmatrix} \quad \text{into} \quad \begin{pmatrix} 1B(k)_0 \\ 1B(k)_1 \\ \vdots \\ 1B(k)_{2^k-2} \\ 1B(k)_{2^k-1} \end{pmatrix},$$

where $B(k)_i$ is the k-bit binary encoding of i. But this conversion can be made in $k-1$ steps according to the induction assumption, and the $(k+1)$-bit Gray code can be converted in k steps. ■

A similar proof can be carried out for a conversion algorithm proceeding from the lowest-order bit to the (second-) highest-order bit by considering

$$\begin{pmatrix} G(k)_0 0 \\ G(k)_0 1 \\ G(k)_1 1 \\ G(k)_1 0 \\ \vdots \\ G(k)_{2^{k-1}} 1 \\ G(k)_{2^{k-1}} 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} B(k)_0 0 \\ B(k)_0 1 \\ B(k)_1 0 \\ B(k)_1 1 \\ \vdots \\ B(k)_{2^{k-1}} 0 \\ B(k)_{2^{k-1}} 1 \end{pmatrix}.$$

An exchange operation is required in the lowest-order dimension between selected pairs of nodes.

COROLLARY 3.3. *The directed routing paths are edge disjoint.*

Proof. A dimension is routed only once, and the routing is an exchange operation. ■

Remark. If the order in which dimensions are routed is different for elements with different source nodes, then it is no longer guaranteed that the directed routing paths are edge disjoint. For instance, if elements from source nodes 10 and 11 in the example above are routed on bit 0 first, but elements from source nodes 14 and 15 are routed on bit 0 last, then two elements traverse the directed edges between processors 10 and 11.

It follows from the corollary that if each node contains a set of elements, the communication of these elements can be pipelined.

COROLLARY 3.4. *Conversion of an n-bit binary-reflected Gray code encoding with M elements per node to a binary encoding, or vice versa, can be performed in $n + M - 2$ communication steps on an n-cube.*

Routing the elements, such that successively lower- (or higher-) order bits are correct, amounts to reflections around the "pivot" points in the Gray code defined by the transition sequence $T(n)$. Table I illustrates the sequence of reflections that converts a 4-bit Gray code to binary code. Processor addresses are given in binary code and the integers stored in a processor are given in decimal representation.

3.1.4. *Rotation of Linear Arrays Embedded by Gray Code Encoding*

Many linear algebra algorithms can be formulated using rotations as an operator on aggregate data structures. In a boolean n-cube, i and

TABLE I
CONVERSION OF GRAY CODE TO BINARY CODE

Gray code	Reflection on bit 2	Reflection on bit 1	Reflection on bit 0
0 0000	0 0000	0 0000	0 0000
1 0001	1 0001	1 0001	1 0001
2 0011	2 0011	2 0011	3 0011
3 0010	3 0010	3 0010	2 0010
4 0110	4 0110	7 0110	6 0110
5 0111	5 0111	6 0111	7 0111
6 0101	6 0101	5 0101	5 0101
7 0100	7 0100	4 0100	4 0100
8 1100	15 1100	12 1100	12 1100
9 1101	14 1101	13 1101	13 1101
10 1111	13 1111	14 1111	15 1111
11 1110	12 1110	15 1110	14 1110
12 1010	11 1010	11 1010	10 1010
13 1011	10 1011	10 1011	11 1011
14 1001	9 1001	9 1001	9 1001
15 1000	8 1000	8 1000	8 1000

$(i + j) \bmod 2^n$ are at a distance of at most n. In a linear array embedded by Gray code encoding of the indices, a rotation by i steps implies a routing $G(n)_j \rightarrow G(n)_{(i+j) \bmod 2^n}, j = \{0, 1, \ldots, 2^n - 1\}$. The minimal number of routing steps for each j is $|G(n)_j \oplus G(n)_{(i+j) \bmod 2^n}|$, where $|x|$ denotes the number of bits equal to 1 in the binary encoding of x. Even with a minimal number of routing steps for each element, the order in which the dimensions are routed has to be determined. This freedom can be used to minimize the intersections between different directed routing paths.

A rotation of i steps can be decomposed into a sequence of rotations of the form 2^r by considering the binary encoding of i. By Lemma 3.3, j and $(j + 2^r) \bmod 2^n$ differ in precisely 2 bits for $r > 0$ and in 1 bit for $r = 0$. Hence, the rotation can be performed by a sequence of communications in two dimensions, one for each nonzero bit of i, with the exception of the lowest-order bit, which requires only one communication. One of the two dimensions (dimension r) is the same for all j, but the other varies. A rotation by this algorithm may require up to $2n$ communications. An element may be communicated between a pair of processors in both directions, as is the case for $j = 0$ and a rotation of $i = 6$ performed as a rotation of length 2 followed by a rotation of length 4.

If the rotation is implemented as a sequence of reflections, i.e., by correcting successive bits that differ in G_j and $G_{(i+j) \bmod 2^n}$, then clearly an element is subject to at most n routing steps. However, more than one matrix element

may traverse the same edge. For instance, consider a rotation of three steps and the paths traversed by elements 0 and 1 in a lowest- to highest-order-dimension routing in a cube of dimension at least 3. We conjecture that there exist routing algorithms that result in edge disjoint paths for all the different source–destination pairs. We have generated such routing schemes for up to 5-cubes.

3.2. *Embeddings of Multidimensional Arrays*

Generalization of the embedding of loops to the embeddings of multidimensional arrays is straightforward. An $N_1 \times N_2 \times \cdots \times N_r$ mesh can be embedded in a cube of dimension $\lceil \log_2 N_1 \rceil + \lceil \log_2 N_2 \rceil + \cdots + \lceil \log_2 N_r \rceil$ by simply assigning $\lceil \log_2 N_i \rceil$ cube dimensions to dimension i of the mesh.

Figure 2 shows the embedding of a 4×4 mesh by a binary-reflected Gray code in a 4-cube. With the ordering of dimensions used in Fig. 2, dimensions 0 and 2 are assigned to the encoding of column indices, and dimensions 1 and 3 to row indices.

The naive embedding of multidimensional arrays is efficient for $N_i = 2^{n_i}$, but for $2^{n_i} < N_i < 2^{n_i+1}$ the *expansion*

$$e = \frac{2(\lceil \log_2 N_1 \rceil + \lceil \log_2 N_2 \rceil + \cdots + \lceil \log_2 N_r \rceil)}{N_1 \times N_2 \times \cdots \times N_r}$$

may be very large. In the worst case $e \approx 2^r$.

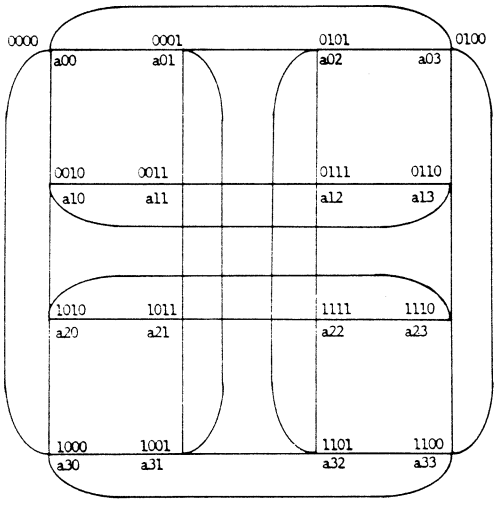

FIG. 2. Embedding of a 4×4 mesh in a 4-cube.

Embeddings with reduced expansion can be obtained at the expense of increased dilation. Aleliunas and Rosenberg [1] have studied the embedding of rectangular grids in square grids. For boolean cubes one can obtain embeddings with an expansion smaller than that of the naive embedding by proceeding in two steps:

- Embed the original mesh, the *guest mesh*, in a mesh, the *host mesh*, having $2^{m_i} = 2^{\lfloor \log_2 N_i \rfloor}$ nodes in some dimensions and $2^{m_j} = 2^{\lceil \log_2 N_j \rceil}$ nodes in other dimensions;
- Embed the host mesh in the cube by a binary-reflected Gray code.

We will give some results for simple embeddings of an $N_1 \times N_2$ guest mesh, in an $M_1 \times M_2$ host mesh, where $M_2 = 2^{m_2}$ and $N_2 = (1 + \alpha)2^{m_2}$, $\alpha < 1$. The naive embedding of the $N_1 \times N_2$ mesh requires a cube of $\lceil \log_2 N_1 \rceil + \lceil \log_2 N_2 \rceil = \lceil \log_2 N_1 \rceil + m_2 + 1$ dimensions. A *break-and-fold* embedding [32] employs the same principle as a carpenter's ruler. With the assumptions made, only one break-and-fold operation is necessary. It is verified easily that $M_1 = 2N_1$ and the number of required cube dimensions is the same as for the naive embedding.

Aleliunas and Rosenberg describe a simple, so-called *step* technique, and a *compression* technique in which one dimension is compressed and the other is expanded in the embedding of a two-dimensional guest mesh in a two-dimensional host mesh. The step and compression techniques are illustrated in Fig. 3.

In the simple step technique the first row of the mesh changes direction after M_2 nodes. Numbering rows and columns from 0, row i changes direction at column $M_2 - 1 - i$ and row $i + N_2 - M_2$. The number of dimensions needed is $m_2 + \lceil \log_2(N_1 + N_2 - M_2) \rceil$. One cube dimension is saved if $N_2 - M_2 \leq 2^{\lceil \log_2 N_1 \rceil} - N_1$. Note that determining which dimension is reduced

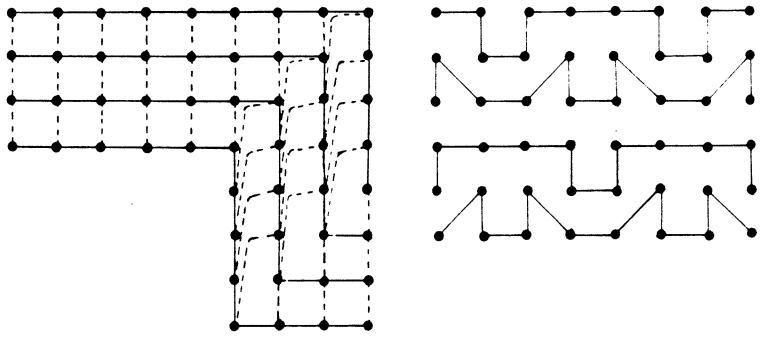

FIG. 3. Embedding of an $N_1 \times N_2$ mesh in an $M_1 \times M_2$ mesh by step embedding and by the compression technique.

is important. If $N_1 \ll N_2$ and $\alpha \approx 1$ then the step embedding might require a number of dimensions in the cube that exceeds $2m_2 \gg m_2 + \lceil \log_2 N_1 \rceil$. The dilation of the step embedding is 3.

The compression technique [1] embeds a guest mesh in a host mesh by repetitively embedding a mesh of size $2a \times 2b$ in a mesh of size $2b \times 2a$. The compression technique is like a "buckling" process in which for a two-dimensional guest mesh one dimension is compressed and nodes are allowed to move into the other dimension of the host mesh. A line of length b is compressed to length a and $b - a$ nodes are moved into the other dimension. The integers a and b are related by the equation $b = (a - 1)d_1 + d_2$, where d_1 and d_2 are positive integers and the dilation is $\max(d_1, d_2)$. The compression technique yields an embedding of an $N_1 \times N_2$ mesh in a $b\lceil N_1/a \rceil \times d\lceil N_2/b \rceil$ mesh. If $d_1 = 1$ and $d_2 = 2$, then $b = a + 1$ points are mapped to a points in the same dimension and one point is mapped into the other dimension. If the equation $(1 + \alpha)2^{m_2}/(a + 1) = \alpha 2^{m_2}$ has an integer solution, i.e., $2^{m_2}/(N_2 - 2^{m_2})$ is an integer, then there is clearly a choice of a for the assumed values of d_1, d_2, for which the compression of the dimension with N_2 nodes yields precisely 2^{m_2} nodes. With the choice of d_1 and d_2 reversed, i.e., $d_1 = 2$ and $d_2 = 1$, the equation for efficient choice of a is instead $a = 2^{m_2}/(2^{2m_2} - N_2)$. With either of these two choices of d_1 and d_2, the dilation $d = 2$. The number of rows M_1 of the host mesh is approximately $N_1 + \lceil \alpha N_1 \rceil$. The number of required dimensions for the host mesh is $m_2 + \lceil \log_2(N_1 + \lceil \alpha N_1 \rceil) \rceil$, which may be 1 less than for the naive embedding of the guest mesh, the best possible.

Embeddings with a smaller average edge length, as well as a smaller number of edges of maximum length, can be found for the boolean cube [15].

3.3. *Finiteness*

For most applications and multiprocessors it is necessary to identify several nodes of the guest graph with a given node in the host graph.

3.3.1. *Consecutive and Cyclic Storage*

We consider square matrices of size $N \times N$ stored in a boolean cube of dimension $2k$, $N \geq 2^k$. Generalization to $M \times N$, $M, N \geq 2^k$, matrices is straightforward. We consider two schemes for identifying matrix elements with nodes of a $2^k \times 2^k$ array, which can be embedded in the cube as described previously. In *consecutive* storage all elements $(i, j) = \{0, 1, \ldots, N - 1\} \times \{0, 1, \ldots, N - 1\}$ of the $N \times N$ array A that satisfy the relations $p = \lfloor i/\lceil N/2^k \rceil \rfloor$, $q = \lfloor j/\lceil N/2^k \rceil \rfloor$ are identified with element $(p, q) = \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\}$ of the $2^k \times 2^k$ array $\hat{A}$. Each processor stores a submatrix of size $\lceil N/2^k \rceil \times \lceil N/2^k \rceil$ or $\lfloor N/2^k \rfloor \times \lfloor N/2^k \rfloor$. In *cyclic* storage all elements (i, j) of A that satisfy the relations $p = i \bmod 2^k$, $q = j \bmod 2^k$ are

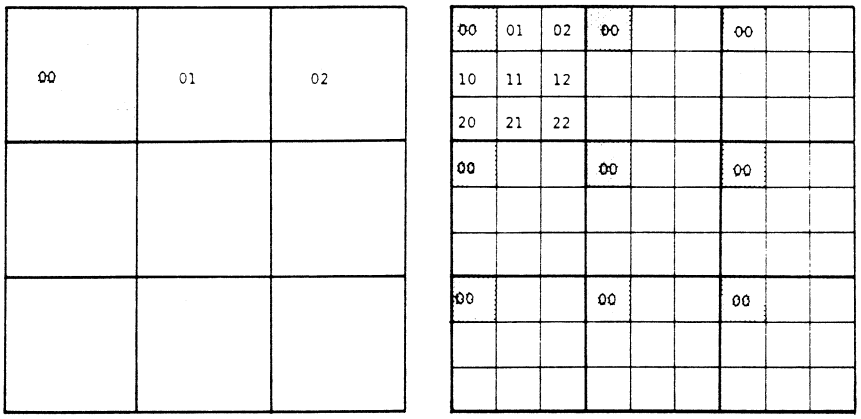

FIG. 4. Consecutive and cyclic storage of a matrix.

identified with element (p, q) of the array $\hat{A}$. The consecutive and cyclic storage schemes are illustrated in Fig. 4.

With the consecutive scheme for identifying multiple-array elements, algorithms devised for the case of $N = 2^k$ can be employed with the apparent change of *granularity* of operations. For linear algebra computations, operations on single elements are replaced by operations on submatrices of size $N/2^k \times N/2^k$. In the cyclic storage scheme the apparent granularity is the same as in the case $N = 2^k$. The submatrices can be viewed as storage planes that are brought to the processing plane (Fig. 5). This simple model is representative of a SIMD architecture. In a pipelined MIMD architecture the sequencing of planes to a processor is preserved, but the planes are broken up into their elements that are delayed in time with respect to each other because of synchronization requirements [27, 33].

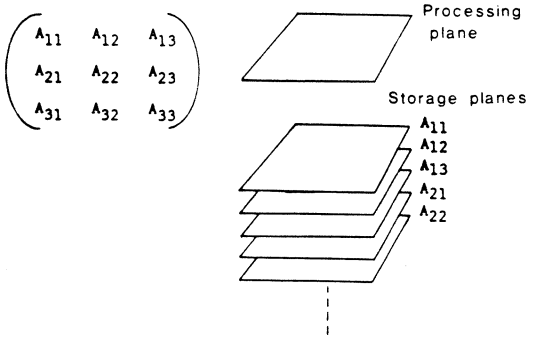

FIG. 5. Processing and storage planes.

In some computations where all elements are subject to the same number of operations, the communication and arithmetic complexities may be independent of whether cyclic or consecutive storage is used, but if some elements participate in more operations than others, then the processor utilization may be affected, and hence also the computational complexity. Matrix multiplication is an example of the former kind of computation, and matrix factorization is an example of the latter [24]. This point will become apparent in Sections 5 and 6.

For an ensemble architecture with communication overhead that is nonzero, or that is not proportional to the number of elements communicated, and that has pipelined arithmetic units, operations of fine grain should, in general, be merged for optimum use. Conversely, if the consecutive storage scheme is used it may be desirable to partition the elemental operations to increase the utilization of the ensemble. The detailed optimization that accounts for specific architectural parameters is left out of our treatment.

3.3.2. *Conversion between Cyclic and Consecutive Storage*

We first consider the conversion from consecutive to cyclic storage order for a one-dimensional array of N elements mapped onto a one-dimensional array of 2^k elements embedded in a k-cube [20], then give some results for two-dimensional arrays. For a complete treatment see [26].

THEOREM 3.1. *The conversion from consecutive storage to cyclic storage (or vice versa) of a 2^n-element array stored in a 2^k-element linear array embedded in a k-cube by binary encoding of the array indices can be carried out in $(N/2^{k+1} + k - 1)$-element transfer times, ignoring overhead, if a processor can support concurrent communication on (all) its ports.*

Proof. The proof is by induction. Let there be $2^m = 2^{n-k}$ elements per processor stored in a linear array. Consider a one-dimensional cube and let the elements in processor 0 be numbered from $0 - 2^m - 1$ and those in processor 1 be numbered $2^m - 2^{m+1} - 1$. A local unshuffle operation orders all the even elements before all the odd elements in each processor. An exchange operation between array element $2^{m-1} + r$ of processor 0 with array element r of processor 1 for $r = \{0, 1, \ldots, 2^{m-1} - 1\}$ moves all the even elements to processor 0 and in order from the first to the last, and similarly all the odd elements in order to processor 1. The consecutive storage order is converted to cyclic order and the theorem is true for a one-dimensional cube, assuming pipelining of the element transfers.

Assume it is true for a $(j - 1)$-dimensional cube. For a j-cube we first perform the conversion in each of its two constituent $(j - 1)$-dimensional subcubes labeled the 0 subcube and the 1 subcube. The 0 subcube contains elements $\{0 - (2^{m+j-1} - 1)\}$ with element $s + r2^{j-1}$ stored in array location r, $0 \leq r < 2^m$, of processor s, $0 \leq s < 2^{j-1}$. The 1 subcube contains elements

$\{2^{m+j-1} - (2^{m+j} - 1)\}$ with element $2^{m+j-1} + s + r2^{j-1}$ stored in array location r of processor $s + 2^{j-1}$. After a local unshuffle operation location r, $0 \leq r < 2^{m-1}$, in the 0 subcube contains element $s + 2r2^{j-1} = s + r2^j$ and location r in the 1 subcube contains $2^{m+j-1} + s + 2r2^{j-1} = 2^{m+j-1} + s + r2^j$. Storage location $r + 2^{m-1}$, $0 \leq r < 2^{m-1}$, contains $s + (2r + 1)2^{j-1} = s + r2^j + 2^{j-1}$ and $2^{m+j-1} + s + (2r + 1)2^{j-1} = 2^{m+j-1} + s + r2^j + 2^{j-1}$, respectively. An exchange operation between location $r + 2^{m-1}$ of processor s and location r of processor $s + 2^{m+j-1}$ for $r = \{0, 1, \ldots, 2^{m-1} - 1\}$ and $s = \{0, 1, \ldots, 2^{j-1} - 1\}$ completes the conversion.

The additional exchange operation for the j-cube is in the new dimension and it follows that pipelining is possible and the proof is complete. ■

Carrying out the recursion in reverse order transforms a consecutive storage order to a cyclic storage order.

COROLLARY 3.5. *The transpose of a $2^n \times 2^n$ matrix stored in column (or row) major order in a linear array of 2^n elements embedded in a boolean n-cube can be performed in $2^{n-1} + n - 1$ steps.*

Note that forming the transpose of an $M \times N$ rectangular matrix and conversion of a linear array with MN elements from consecutive to cyclic storage are not necessarily equivalent operations.

Figure 6 illustrates the conversion algorithm.

Clearly, the local unshuffle operation need not be carried out explicitly. Note that exchanges are always performed on half of the local address space, regardless of the recursion step. This property is not true in forming the transpose of a rectangular matrix.

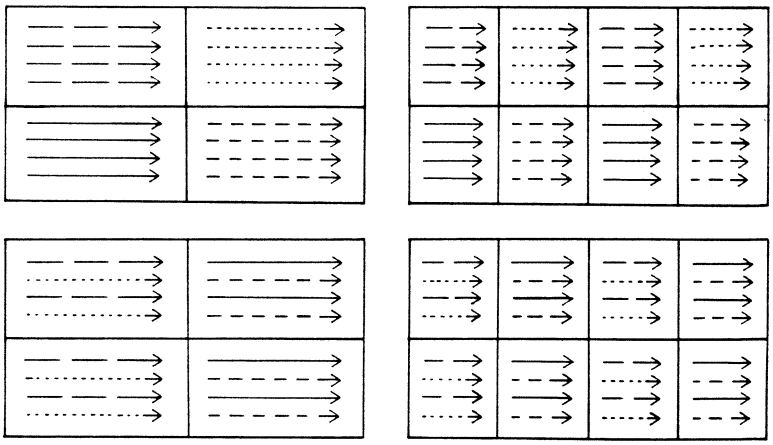

FIG. 6. Transforming cyclic storage to consecutive storage. *Left*, recursion step 1; *right*, recursion step 2.

THEOREM 3.2. *The conversion of storage form from consecutive to cyclic (or vice versa) for a $2^n \times 2^n$ matrix stored in a $2^k \times 2^k$ array embedded in a $2k$-cube by the binary encoding of the row and column indices can be performed in $2^{2(n-k)-1} + 2k - 1$ steps.*

The theorem can be proved by induction similarly to that of Theorem 3.1. For each induction step there is communication of $2^{2(n-k)-1}$ elements between each pair of nodes and in two dimensions (row and columns). The dimensions are distinct, allowing for the pipelining of successive transformation steps.

The conversion can be performed with the same complexity, if the linear array instead of being embedded by a binary encoding is embedded by a binary-reflected Gray code encoding. One algorithm with this property is obtained if in each step of the conversion from consecutive storage to cyclic storage the sequences in the two subcubes are assumed to be stored cyclically, but one in normal order and the other in reverse order. If the 0 subcube contains the normal order sequence and the 1 subcube the reverse order sequence, then the unshuffle and exchange operations yield a sequence stored cyclically in normal order in the combined cube. Reversing the roles of the two subcubes yields a combined sequence stored cyclically in reverse order. The theorem below follows:

THEOREM 3.3. *The conversion from consecutive storage to cyclic storage (or vice versa) of a 2^n-element array stored in a 2^k-element linear array embedded in a k-cube by a binary-reflected Gray code encoding of the array indices can be carried out in $(N/2^{k+1} + k - 1)$-element transfer times ignoring overhead, if a processor can support concurrent communication on (all) its ports.*

3.4. *Tree Embeddings*

3.4.1. *Spanning Trees*

Broadcasting a message from a single source to all other nodes (one-to-all distribution) can be implemented by algorithms generating some form of a spanning tree. *Spanning Binomial Trees* (SBT) [14] and complete binary trees [3, 6] are two often considered spanning trees. The SBT algorithm is used most frequently on boolean cubes and is very simple to implement. However, it is nonoptimal for broadcasting of large data sets. Optimum broadcasting algorithms are derived in [14], which also contains an analysis of broadcasting and personalized communication based on spanning binomial trees and complete binary trees.

The following algorithm generates a spanning binomial tree rooted at node 0. The processor address is $(i_{n-1}\, i_{n-2} \cdots i_0)$ and the highest-order bit that is 1 is k with $k = -1$ for processor $(00 \cdots 0)$.

Algorithm SBT

For $k = 0$ to $n - 1$ concurrently for all $\{i_{k-1} i_{k-2} \cdots i_0\}$ **do**
Processor $(00 \cdots 00 i_{k-1} i_{k-2} \cdots i_0)$ sends a message to processor $(00 \cdots 01 i_{k-1} i_{k-2} \cdots i_0)$
Processor $(00 \cdots 01 i_{k-1} i_{k-2} \cdots i_0)$ receives a message from processor $(00 \cdots 00 i_{k-1} i_{k-2} \cdots i_0)$
enddo

In some algorithms data distribution takes place from different sources during the course of the computations and a certain partial order of arrival is required for correctness. For instance, in Gaussian elimination a row/column has to be updated with respect to all preceding pivots before becoming a pivot row/column.

LEMMA 3.5. *Initiating successive SBT broadcasts at nodes corresponding to the Gray code of successive integers, after each such node has completed its transmission operations for preceding sources, guarantees that the order of arrival of messages at all processors is the same as the order of distribution.*

Proof. The distribution from processor i reaches all nodes at distance k in k routing steps. Processor $i + 1$ is at distance 1 from node i for all i due to the Gray code encoding. A processor at distance k from i is at least at distance $k - 1$ from processor $i + 1$, and processor $i + 1$ starts its distribution two routing steps after processor i since it transmits the message from processor i during the routing step that succeeds the one during which the distribution from i is initiated. ■

3.4.2. *Embeddings of Complete Binary Trees*

Complete binary trees are of interest for the solution of tridiagonal systems by odd–even cyclic reduction [21], the solution of banded systems by substructuring techniques [22], and the solution of systems of equations by nested dissection with the equations originating from regular meshes [38].

3.4.2.1. *Embeddings with expansion 1.*

THEOREM 3.4. *An embedding of a complete binary tree of* $2^n - 1$ *nodes in an n-cube, by labeling the tree nodes in inorder and embedding the tree by a binary encoding of the node indices, yields an embedding in which a parent node and its left descendant are at distance* 1, *the parent and its right descendant are at distance* 2, *and the right and left descendants are at distance* 1 *from each other.*

Proof. The proof is by induction. The claim is clearly true for a two-level tree. Assume it is true for a k-level tree. The root of the k-level tree labeled in inorder has index $2^{k-1} - 1$, i.e., bit k is 0 and all lower-order bits are 1. For

a $(k+1)$-level tree the index of the new root node is $2^k - 1$ and the new root and its left descendant differ in bit k only. The labeling of the right descendant of the root is $2^{k-1} - 1 + 2^k$, since there are $2^k - 1$ nodes in the left subtree and there is also the root. Hence, the left and right descendants of the new root differ in 1 bit (bit k) and the root and the right descendants differ in bits k and $k-1$ and the theorem follows. ■

THEOREM 3.5. *An embedding of a complete binary tree of* $2^n - 1$ *nodes in an n-cube, by labeling the tree nodes in inorder and embedding the tree by a binary-reflected Gray code encoding of the node indices, yields an embedding in which a leaf node is at distance* 1 *from its parent node and all other nodes are at distance* 2 *from their respective parent node. Left and right descendants of a node are always at distance* 2 *from each other.*

The proof of Theorem 3.5 is immediate from Lemma 3.3 and the inorder labeling of the tree.

Remark. From the definition of the Gray code it follows that successive nodes in an inorder-labeled complete binary tree are at distance 1, and that successive nodes in subtrees rooted at the root of the tree are at distance 2. This is in essence Corollary 3.1.

THEOREM 3.6. *The* $(n-1)$*-level subtree formed by the interior nodes of an n-level complete binary tree can be embedded in an* $(n-1)$*-dimensional subcube (of the n-cube embedding the n-level tree) by an exchange operation between selected adjacent processors. The control of the operation can be entirely local.*

Proof. From the binary-reflected Gray code

$$G(n) = \begin{pmatrix} G(n-1)_0 0 \\ G(n-1)_0 1 \\ G(n-1)_1 1 \\ G(n-1)_1 0 \\ G(n-1)_2 0 \\ G(n-1)_2 1 \\ \vdots \\ G(n-1)_{2^{n-1}-1} 1 \\ G(n-1)_{2^{n-1}-1} 0 \end{pmatrix}$$

it is clear that every other odd integer starting with the second is embedded in processors with even addresses. The $(n-1)$-level subtree rooted at the root of the n-level tree labeled in inorder $(0 - 2^n - 1)$ contains all nodes with an odd index. The leaf nodes of the $(n-1)$-level subtree are in the odd subcube. By an exchange operation between even processors storing odd-indexed nodes with odd-indexed processors having the same leading $n-1$ bits (which

stores the preceding even integer by construction of the code), all odd integers are mapped into the odd subcube as illustrated in Table II.

Let node $i = 2j + 1$, $j = \{0, 1, \ldots, 2^{n-1} - 1\}$. Then j is embedded in processor $(G(n-1)_j 1)$ and induction completes the proof. ■

3.4.2.2. *Embeddings with dilation 1.* It is interesting to note that there exists a dual form of the dilation 2 expansion $1 + 1/(2^n - 1)$ complete binary tree embedding in which the dilation is 1 and the expansion is $2 + 2/(2^n - 1)$.

THEOREM 3.7. *A complete binary tree of* $2^n - 1$ *nodes can be embedded in an* $(n+1)$*-cube with dilation* 1.

We give an algorithm for the embedding, and prove that the embedding generated by the algorithm satisfies the theorem. For other embeddings of complete and arbitrary binary trees in boolean cubes see [3, 6].

In the algorithm below an n-level tree embedded in an $(n+1)$-cube is extended to an $(n+1)$-level tree in an $(n+2)$-cube by adding a leaf level. The $(n+1)$-cube is partitioned into four subcubes $(0xx\cdots x|0xx\cdots x)$, $(0xx\cdots x|1xx\cdots x)$, $(1xx\cdots x|0xx\cdots x)$, and $(1xx\cdots x|1xx\cdots x)$. We refer to these subcubes as the 00, 01, 10, and 11 subcubes. Half of the new leaf nodes are taken to be the image of the leaf nodes in the n-level tree in the added $(n+1)$-cube. The other leaf nodes are mapped into the old $(n+1)$-cube. Half of the leaf nodes are mapped into subcube 01, a quarter into 10, and a quarter into 11 recursively, as illustrated in Fig. 7. L_n denotes the number of leaf nodes in the n-level tree. The mapping of the new leaf nodes within the old subcube is such that for each leaf node of the n-level tree that is in the old subcube 01, one new leaf node is taken as the image in the old subcube 11. Similarly, one new leaf node for each old leaf node in the old subcube 11 is taken to be the image in the old subcube 10. One new leaf node for each old leaf node in the old subcube 10 is mapped into the subcube itself.

If the mapping of the leaf nodes of the n-level tree to nodes within the old $(n+1)$-dimensional cube is conflict free at some stage, then it follows that

TABLE II
NODE EMBEDDINGS AFTER AN EXCHANGE OPERATION

Node index	Gray code
1	$G(n-1)_0 1$
3	$G(n-1)_1 1$
5	$G(n-1)_2 1$
⋮	⋮
$2^n - 3$	$G(n-1)_{2^{n-1}-2} 1$
$2^n - 1$	$G(n-1)_{2^{n-1}-1} 1$

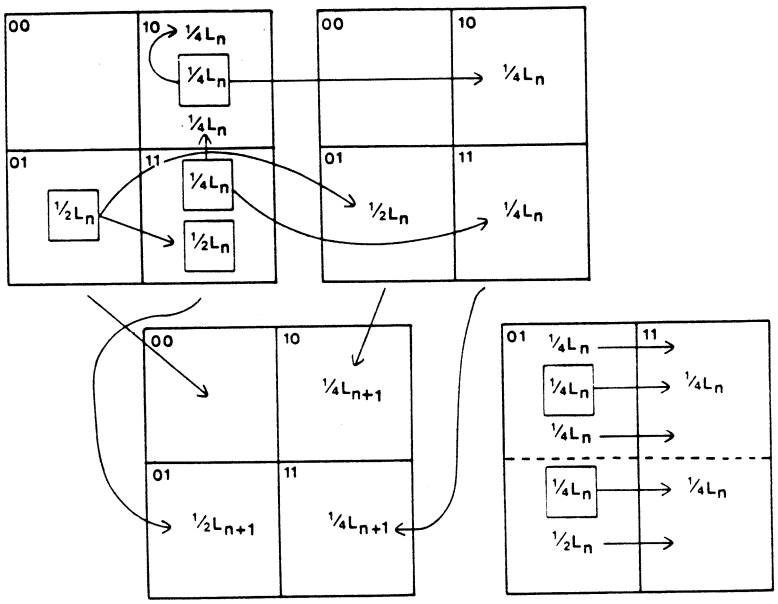

FIG. 7. Recursive embedding of complete binary trees in boolean cubes.

the mapping of leaf nodes from subcube 01 into subcube 11 also is conflict free in the next embedding step. The two subcubes are further subdivided to show the origin of the elements that reside in them. The fact that the mapping in the lower half is conflict free follows from the assumption that the mapping in the 01 and 11 subcubes was conflict free in the preceding recursion step. That the mapping of leaf nodes for the upper half is conflict free follows from the assumption that the mapping from subcube 11 to 10 and within subcube 10 is conflict free. The arguments for the mapping to and within subcube 10 are more complex.

Note that an alternative tree embedding is obtained by complementing the bits in the binary encoding of the node numbers. The embedding so obtained is not disjoint from the embedding generated by the algorithm above.

We now give an algorithm implementing the above strategy. Let the root of the complete binary tree be at level 1 and the leaves at level n. Embed the root of the tree in cube node $(00\cdots0)$, the right child of the root in node $(00\cdots010)$, and the left child in node $(00\cdots0100)$. A *left-edge* connects a node with its left child and a *right-edge* connects a node with its right child. Label the edges of the tree with the bit of a node's address that is complemented in obtaining the address of the child connected by the edge. Hence, the left-edge of the root is labeled 2 and the right-edge is labeled 1 with the address bits labeled 0 through $n-1$. The right-edges of nodes at level i, $2 \leq i$

$< n$ are labeled $i + 1$, i.e., with the level index of the children nodes. The left-edge of node $(00\cdots0100)$ is labeled 0, and the left-edge of node $(00\cdots010)$ is labeled 2. Then, for levels i, $3 \leq i < n$, the left-edges connecting a node p_i, where i is the level index, with a node at level $i + 1$ are labeled as follows:

- If the node p_i is the left child of node p_{i-1}, then copy the label of the right-edge of p_{i-1} to the left-edge of p_i.
- If the node p_i is the right child of node p_{i-1}, then copy to the left-edge of p_i the label of the left-edge of the node immediately to the right of p_{i-1}. If p_{i-1} is the rightmost node at level $i - 1$, then copy the label of the left-edge of the first node at level $i - 1$.

The result of this labeling scheme is shown in Fig. 8 for a six-level complete binary tree.

To prove that the above algorithm satisfies Theorem 3.7 we need a few lemmas.

LEMMA 3.6. *Consecutive left-edges between levels i and* $i + 1$, $1 \leq i < n$, *scanning the tree in right-to-left (or left-to-right) order never carry the same label.*

Proof. It is easily seen to be true for $i = 1$. Assume it is true for some i. Then for level $i + 1$ the rightmost left-edge and every other left-edge are labeled $i + 1$. The labels of the other left-edges are obtained by copying the labels of the left-edges of the nodes at level i starting with the second rightmost left-edge and terminating with the first, and the lemma follows. ■

LEMMA 3.7. *For any node at level i the label of its left-edge is never used again in the subtree rooted at i.*

Proof. Considered two adjacent subtrees at level i with roots p and q. Let the left-edge of p be labeled α and the left-edge of q be β, $\alpha, \beta \leq i$ and $\alpha \neq \beta$

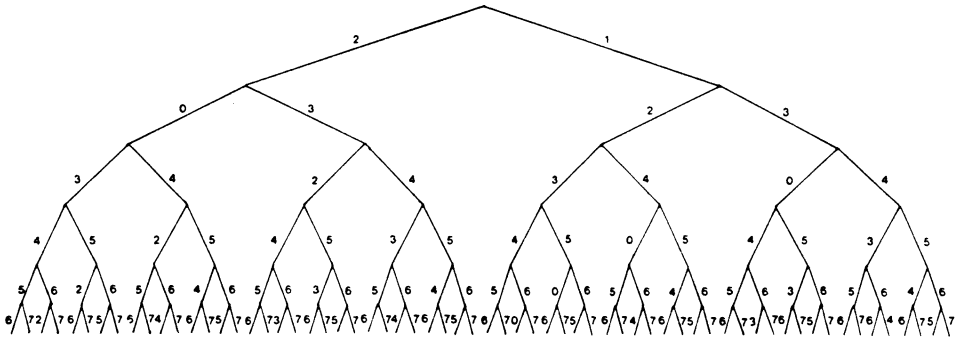

FIG. 8. Labeling of a complete binary tree for dilation 1 embedding in a boolean cube.

from Lemma 3.7. Right-edges are labeled with the index of the lower of the two levels it connects. Hence, α or β will never appear on a right-edge. Left-edges of a node r, being the left child of a node s, are labeled with the label of a right-edge of s. The smallest value of such a label is $i + 1 > \alpha, \beta$. Left-edges of a node r, being the right child of a node s, copy the label of the left-edge of the node adjacent to s. Left-edges from nodes on the path from p to the rightmost leaf node of the subtree rooted at p copy the labels of the left-edges forming the path from node q to the leftmost leaf node of the subtree rooted at q, with the exception of the last edge. But, the edges forming the path from q to the leaf are labeled $\beta, i + 1, i + 2, \ldots, n - 1$, which are all distinct from α, and the lemma follows. ■

LEMMA 3.8. *Consider a subtree at level i rooted at node p. Then, the label $i + 1$ of its right-edge appears only on left-edges reachable by the traversal of exactly one more right-edge in the right subtree of p.*

Proof. The only way the label $i + 1$ can appear on a left-edge in the right subtree is by a copy action from another left-edge. Let q be adjacent to p, to its right, and at level $i + 1$. Furthermore, let s be the left child of q. Then, the left-edge of s is labeled $i + 1$. Let the right child of p be u. Then, the label $i + 1$ is copied from the subtree of q to the left-edge of the right child of u. This label is then copied to the left-edge of any node that is the right child of a node on the path from u to the leftmost leaf node of the subtree rooted at u, which can be shown by induction. Indeed, with r denoting a right-edge and l a left-edge, the label of r occurs only on the last edge of paths of the form rl^*rl in the subtree defined by the first right-edge, where l^* denotes an arbitrary number of left-edges in sequence. ■

We now prove Theorem 3.7.

Proof. From Lemma 3.7 it follows that the nodes within the right and left subtrees of any node are distinct. It also follows from this lemma that no node in the left subtree of any node p can have the same address as p. It remains to be shown that there exists no node in the right subtree of p with the same address as p. Every label has to be encountered an even number of times for two addresses to coincide. Every right-edge introduces a new label. But, any label on a right-edge appears only on left-edges that require the traversal of precisely one additional right edge by Lemma 3.8. ■

4. MATRIX TRANSPOSITION

The transposition of a matrix can be formed by an exchange of antidiagonal blocks on successively smaller (larger) submatrices. This recursive procedure [41, 8] is illustrated in Fig. 9.

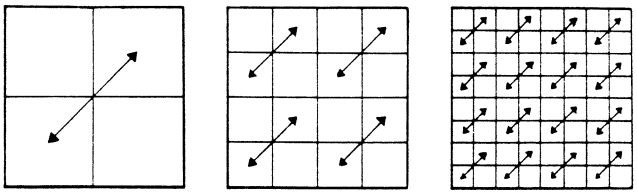

FIG. 9. Recursive transposition of a matrix.

In the following we assume that the recursive transposition proceeds toward successively smaller submatrices. In the first step of the recursive procedure the interchange of data is performed on the highest-order bit of the binary encoding of the row indices *and* the highest-order bit of the column indices. In the second step the interchange is performed on the second-highest-order bit of the row *and* column indices, for all combinations of the highest-order bits, i.e., four submatrices. The number of index sets that differ in one bit of the row and column indices increases by powers of 4 as the procedure progresses toward lower-order bits.

With the consecutive storage scheme of a $2^n \times 2^n$ matrix in a $2^k \times 2^k$ array, it is easily seen that the first k steps imply interarray element communication and that the last $n - k$ steps are local. With the cyclic storage scheme the first $n - k$ steps are local, whereas the last k steps require interarray element communication. After the first $n - k$ steps there are $2^{2(n-k)}$ submatrices of size $2^k \times 2^k$ to transpose. Each submatrix has one element per processor.

THEOREM 4.1 [25]. *The transpose of a $2^n \times 2^n$ matrix stored cyclically or consecutively in a $2^k \times 2^k$ array embedded in a boolean cube by a separate binary encoding of the row and column indices can be formed in a time proportional to $2^{2(n-k)} + 2k - 1$, assuming a communications overhead proportional to the number of elements being transferred, and concurrent communication on multiple (possibly all) ports.*

Proof. Let $n = k$ and a processor address be $(B(r)|B(c))$, where $B(r)$ and $B(c)$ denote the binary encoding of the row and column index. For $k = 1$ the theorem is clearly true. Assume it is true for $k = m$. For $k = m + 1$ we exchange the submatrices $(1xx\cdots x|0xx\cdots x)$ and $(0xx\cdots x|1xx\cdots x)$. This exchange operation requires two communications, one in each of the new dimensions. The transpose is now obtained by transposing four submatrices of size $2^m \times 2^m$ in four distinct subcubes, and the theorem follows for $n = k$.

For $n > k$ the matrix is partitioned into submatrices of size $2^{n-k} \times 2^{n-k}$. With cyclic storage the last k steps require interprocessor communication whereas in consecutive storage the first k steps require communication.

Successive steps of the recursive transposition procedure perform exchanges on processor interconnections in different dimensions. It follows

that the communication of elements of different submatrices can be pipelined for all steps of the transposition operation, since the directed paths from origin to destination intersect only at nodes. ■

With the array embedded according to a binary-reflected Gray code, successive row and column indices are always located in neighboring nodes of the cube. However, the communication required by the recursive procedure is between nodes storing elements of rows and columns whose binary encoding differs in successively higher- or lower-order bits. Each such communication requires communication in two dimensions by Lemma 3.3. Nevertheless, the transpose can be formed in the same time as for a binary encoding.

THEOREM 4.2 [23]. *The transpose of a $2^n \times 2^n$ matrix stored cyclically or consecutively in a $2^k \times 2^k$ array embedded in a boolean cube by a separate binary-reflected Gray code encoding of the row and column indices can be formed in a time proportional to $2^{2(n-k)} + 2k - 1$ by performing reflections on successively lower- (higher-) order bits in the encoding of row and column indices, assuming a communications overhead proportional to the number of elements being transferred, and concurrent communication on multiple (possibly all) ports.*

Proof. For the proof we assume that the transpose is formed by performing reflections on successively lower-order bits in the row and column encodings in alternating order. Elements stored in location (i, j) of the $2^k \times 2^k$ array are stored in processor $(G(k)_i G(k)_j)$ and shall be moved to processor $(G(k)_j G(k)_i)$. The theorem is clearly true for $k = 1$. Assume it is true for $k = m$. Then for $k = m + 1$ the task of forming the transpose is that of routing elements from $(G(m+1)_i G(m+1)_j)$ to $(G(m+1)_j G(m+1)_i)$ for all $(i, j) = \{0, 1, 2, \ldots, 2^{m+1} - 1\} \times \{0, 1, 2, \ldots, 2^{m+1} - 1\}$. But:

$$
(G(m+1)_i G(m+1)_j)
$$

$$
= \begin{cases}
(0G(m)_i 0G(m)_j), & (i,j) = \{0,1,2,\ldots,2^m-1\}\{0,1,2,\ldots,2^m-1\}, \\
(1G(m)_{2^{m+1}-i-1} 0G(m)_j), & \\
\qquad (i,j) = \{2^m, 2^m+1, \ldots, 2^{m+1}-1\}\{0,1,2,\ldots,2^m-1\}, & \\
(0G(m)_i 1G(m)_{2^{m+1}-j-1}), & \\
\qquad (i,j) = \{0,1,2,\ldots,2^m-1\}\{2^m, 2^m+1, \ldots, 2^{m+1}-1\}, & \\
(1G(m)_{2^{m+1}-i-1} 1G(m)_{2^{m+1}-j-1}), & \\
\qquad (i,j) = \{2^m, 2^m+1, \ldots, 2^{m+1}-1\}\{2^m, 2^m+1, \ldots, 2^{m+1}-1\}. &
\end{cases}
$$

The routing $(1G(m)_{2^{m+1}-i-1} 0G(m)_j) \rightarrow (0G(m)_{2^{m+1}-i-1} 1G(m)_j)$ requires one communication in each of two dimensions. The routing $(0G(m)_i$

$\times\ 1G(m)_{2^{m+1}-j-1}) \rightarrow (1G(m)_i 0G(m)_{2^{m+1}-j-1})$ also requires one communication in each of two dimensions. The remaining routing steps are confined to four subcubes. The two submatrices being exchanged have a reflected representation, with respect to both row and column indices, in their new location. This implies only that in the next recursion step an exchange operation is performed between the diagonal blocks in these subcubes instead of antidiagonal blocks, and the desired routing can be performed in $2m$ communication steps by the induction assumption. The theorem follows by the observation that distinct dimensions are routed in the different steps of the algorithm allowing for pipelining of the data movement of elements with the same source and destinations. ■

In forming the transpose of a matrix, half of the edges of the cube in a given dimension are used in a given step. For optimum transpose algorithms see [26].

It is interesting to compare the complexity of forming the transpose on a boolean cube with that of forming the transpose on a two-dimensional mesh.

4.1. *Forming the Matrix Transpose on a Torus*

We first note that the maximum distance a matrix element has to travel is $2 \times (2^k - 1)$ for a two-dimensional square mesh and 2^k for a torus with 2^k processors in each dimension. Second, we note the following lower bound:

THEOREM 4.3. *The transpose of a $2^n \times 2^n$ matrix stored consecutively or cyclically in a $2^k \times 2^k$ array embedded naively in a torus with 2^k processing elements in each dimension requires at least a time that is proportional to $2^{2n}(1 - 2^{-k})/(8(2^k - 1))$.*

Proof. The lower bound is immediate by noting that $2^{2n}(1 - 2^{-k})/2$ elements must be transferred through $4(2^k - 1)$ ports. ■

THEOREM 4.4. *The transpose of a $2^k \times 2^k$ matrix stored in row major order in a torus with 2^k processors in each dimension can be performed in 2^k routing steps.*

Proof. For a torus of $2^k \times 2^k$ processors we shift the element stored in processor (i, j), $i = \{0, 1, \ldots, 2^{k-1} - 1\}$, $i < j \leq i + 2^{k-1}$, $r = j - i$ steps in the direction of decreasing column indices and then r steps in the direction of increasing row indices. Similarly, the element stored in processor (i, j), $j < i \leq j + 2^{k-1}$, $j = \{0, 1, \ldots, 2^{k-1} - 1\}$, is shifted $r = j - i$ steps in the direction of decreasing row indices and then r steps in the direction of increasing column indices. For (i, j), $i = \{2^{k-1}, 2^{k-1} + 1, \ldots, 2^k - 1\}$, $j < i - 2^{k-1}, j > i$, element (i, j) is moved $r = (j - i) \bmod 2^k$ steps in the direction of decreasing column indices; then the same number of steps in the direction of increasing row indices, i.e., element (i, j) is moved to processor $((i + (j - i) \bmod 2^k) \bmod 2^k$,

$(j - (j - i) \bmod 2^k) \bmod 2^k$). Element (i, j), $i < j - 2^{k-1}$, $i > j$, $j = \{2^{k-1}, 2^{k-1} + 1, \ldots, 2^k - 1\}$, is moved $r = (i - j) \bmod 2^k$ steps in the direction of decreasing row indices and r steps in the direction of increasing column indices. ■

COROLLARY 4.1. *If there are no end-around connections, then the transpose of a $2^k \times 2^k$ matrix on the same size torus can be formed in $2(2^k - 1)$ steps by shifting superdiagonal r, $r > 0$, r processors in the direction of decreasing column indices, and then r processors in the direction of increasing row indices. Subdiagonal r is shifted r processors in the direction of decreasing row indices and then r steps in the direction of increasing column indices.*

The proof is by direct evaluation. It is clear that no competition for communication links occurs.

COROLLARY 4.2. *By pipelining the element transfers for different submatrices and using two distinct paths between each pair of source/destination processors, a $2^n \times 2^n$ matrix stored cyclically or consecutively in an array of $2^k \times 2^k$ elements embedded naively in a torus of the same size can be formed in a time proportional to $(\lceil 2^{2(n-k)}/2 \rceil + 1)2^{k-1}$.*

In comparing the complexity estimates for forming the transpose of a matrix on a two-dimensional mesh with that of the boolean cube it is concluded that the cube offers a speedup over the mesh by a factor of approximately 2^k for $n \gg k$, and by a factor of $2^k/k$ for $n \approx k$.

5. MATRIX MULTIPLICATION

The task is to compute $C \leftarrow C + A \times B$. If the matrices are of size $2^k \times 2^k$ and the cube has $2k$ dimensions, then a parallel version of the conventional algorithms requiring $(2 \times 2^k - 1)2^{2k}$ arithmetic operations requires a time of at least $2 \times 2^k - 1$ on 2^{2k} processors capable of one arithmetic operation at a time. Additional time may be required due to data alignment, synchronization, or data movement. The multiplication of two $N \times N$ matrices, $N > 2^k$, can be accomplished in a time that is at most $(N/2^k)^3$ times the time for the multiplication of $2^k \times 2^k$ matrices.

We first briefly describe an algorithm for the multiplication of $2^k \times 2^k$ matrices embedded in a boolean $2k$-cube by a binary encoding of row and column indices. This algorithm is due to Dekel *et al.* [5]. We show that the data movement for multiple matrix multiplications is pipelinable. We then give three algorithms for the multiplication of matrices embedded by a binary-reflected Gray code.

5.1. *Multiplication of Matrices Embedded by Binary Encoding, MMC1*

Assume that each processor has three registers, E, F, H. Initially $E(i, j) \leftarrow A(i, j)$, $F(i, j) \leftarrow B(i, j)$, $H(i, j) \leftarrow C(i, j)$, $(i, j) = \{0, 1, \ldots, 2^k - 1\} \times \{0, 1,$

$\ldots, 2^k - 1\}$. In the algorithm by Dekel *et al.* [5] the set-up phase during which the multiplier and multiplicand are aligned consists in the data movement

$$E(i,j) \leftarrow E(i, i \oplus j) \qquad \text{and} \qquad F(i,j) \leftarrow F(i \oplus j, j).$$

Clearly, $E(i,j) \times F(i,j)$ are valid product terms for all i and j. In the multiplication phase all elements of a row of A need to visit every processor in that row, and every element of B needs to be communicated to every processor in that column with C computed *in-place*. Dekel *et al.* use a recursive procedure in which the desired data movement is performed in half-rows, then an exchange operation between the two halves, followed again by data movement in half-rows. The order of the dimensions in which exchange operations are performed is the same as the order in which the dimensions are used in a binary-reflected Gray code.

THEOREM 5.1 [5]. *The computation $C \leftarrow C + A \times B$ on a boolean $2k$-cube can be performed in $2^k + k - 1$ communication and 2^k arithmetic steps with A, B, C being $2^k \times 2^k$ matrices embedded by a binary encoding of row and column indices, each in half of the address space.*

During the multiplication phase the resources are fully utilized. In the set-up phase, elements of a row of A move within the subcube embedding that row, and elements of a column of B move within the subcube embedding that column. The movements of different rows of A and columns of B are independent. The routing required in the subcube defined by row index i is $j \leftarrow i \oplus j$. This routing consists of exchanges in different dimensions, namely those in which the binary encoding of i has a bit equal to 1. Each dimension is routed only once. Hence, routing the dimensions in the same order for all j guarantees that any communications link is traversed by only one element in a given direction, and the next lemma follows:

LEMMA 5.1. *The set-up phases for multiple matrix multiplications can be pipelined such that the set-up phase for an $N_1 \times N_2$ matrix requires $\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil + k - 1$ communications, given that a processor can support concurrent communication on (all of) its ports.*

The following result is now immediate:

THEOREM 5.2. *The computation $C \leftarrow C + A \times B$ on a boolean $2k$-cube can be performed by algorithm MMC1 in $\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil (2^k - 1) + \max(\lceil N_1/2^k \rceil, \lceil N_3/2^k \rceil) \lceil N_2/2^k \rceil + k - 1$ communication and $\lceil N_1/2^k \rceil \times \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil 2^k$ arithmetic steps with A an $N_1 \times N_2$ matrix and B an $N_2 \times N_3$ matrix, and the matrices stored cyclically or consecutively in a $2^k \times 2^k$ array embedded in the cube by a binary encoding of row and column indices, each in half of the address space.*

We note that all k-dimensions used to embed a row are being used in the multiplication phase, and also in the set-up phase.

5.2. *Multiplication by Rotation of Gray Code Encoded Matrices, MMC2*

Cannon [4] describes an algorithm for SIMD-type ensembles configured as tori. In the set-up phase row i of A is rotated i steps in the direction of decreasing column indices, and column j of B is rotated j steps in the direction of decreasing row indices: $E(i, j) \leftarrow E(i, (j + i) \bmod 2^k)$, $F(i, j) \leftarrow F((i + j) \bmod 2^k, j)$.

Clearly, $E(i, j) \times F(i, j)$ are valid product terms for $(i, j) = \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\}$. The set-up phase implements the skewing of data streams required for synchronization, as seen in many systolic algorithms [30, 27]. In the multiplication phase the following computations are performed in Cannon's algorithm:

```
for k = 1 step 1 until 2^k do
  for all (i, j) in parallel do
    H(i, j) ← H(i, j) + E(i, j) × F(i, j)
    E(i, j) ← E(i, (j + 1) mod 2^k)
    F(i, j) ← F((i + 1) mod 2^k, j)
  end
end
```

Adapting this algorithm to boolean cubes is straightforward for the multiplication phase due to the assumed Gray code embedding. In the set-up phase the maximum distance a matrix element needs to move is k, and it occurs for an element (i, j) such that $|G_j \oplus G_{(i+j) \bmod 2^k}| = k$. Assuming that the rotation can be performed such that the data movements can be pipelined, the following complexity result follows:

THEOREM 5.3. *The computation $C \leftarrow C + A \times B$ on a boolean $2k$-cube can be performed by algorithm MMC2 in $\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil (2^k - 1) + \max(\lceil N_1/2^k \rceil, \lceil N_3/2^k \rceil) \lceil N_2/2^k \rceil + k - 1$ communication and $\lceil N_1/2^k \rceil \times \lceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil 2^k$ arithmetic steps, with A an $N_1 \times N_2$ matrix and B an $N_2 \times N_3$ matrix, and the matrices stored cyclically or consecutively in a $2^k \times 2^k$ array embedded in the cube by a binary-reflected Gray code encoding of row and column indices, each in half of the address space.*

5.3. *Multiplication by Reflection of Gray Code Encoded Matrices, MMC3*

In this algorithm the distribution of any element of a row i of A is emanating from processor $(G_i G_0)$ (the first column), and the distribution of any element of column j of B is emanating from processor $(G_0 G_j)$ (the first row). In

addition to the registers E and F, two registers P and Q are needed, if the distribution is made by the SBT algorithm. The elements of A have to be moved to the first column, and the elements of B to the first row. This movement is accomplished by shifts, one step at a time. In shift r the data movement is $E(i, j) \leftarrow E(i, j + 1)$, $j = \{0, 1, \ldots, 2^k - r - 1\}$, and $F(i, j) \leftarrow F(i + 1, j)$, $i = \{0, 1, \ldots, 2^k - r - 1\}$. The communication is between adjacent processors due to the Gray code encoding.

Instead of an explicit set-up phase, as in the previous algorithms, the synchronization may be performed by starting the distributions (and computations) in the subcube storing column j at time $|G(k)_j|$ and in the subcube storing row i at time $|G(k)_i|$. By this initiation procedure it is clear that a_{i0} reaches the processor storing element (i, j) of C at time $|G(k)_i| + |G(k)_j|$. Similarly, b_{0j} reaches the same processor at time $|G(k)_j| + |G(k)_i|$ for all $(i, j) = \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\}$. From the shifting of elements toward the first row and column, it follows that the elements a_{im} and b_{mj} reach processor (i, j) at time $|G(k)_i| + |G(k)_j| + m$, if no communication conflicts occur. Upon receipt of the elements of A and B the multiplication of the contents of the P and Q registers can be performed, i.e., $H(i,j) \leftarrow H(i,j) + P(i,j)Q(i,j)$. The following lemma asserts that indeed no communication conflicts occur:

LEMMA 5.2. *The directed path* $G_{2^k-1}, G_{2^k-2}, \ldots, G_1, G_0$ *and the paths of the SBT algorithm are edge disjoint.*

Proof. The edges of the SBT are all directed toward the node with the higher address, i.e., $(00\cdots0i_{r-1}i_{r-2}\cdots i_0) \rightarrow (00\cdots1i_{r-1}i_{r-2}\cdots i_0)$, $r = \{0, 1, \ldots, k - 1\}$. The proof is by induction. It is clearly true for $k = 1$. Assume it is true for $k = m$. Then for $k = m + 1$ the edges of the SBT are directed from nodes $(0xx\cdots x)$ to nodes $(1xx\cdots x)$ in the appropriate subcubes. But, the directed path from element $2^{m+1} - 1$ to element 0, $G_{2^{m+1}-1}, G_{2^{m+1}-2}, G_1, G_0$, traverses only one of the edges joining the two subcubes, namely the edge $(1100\cdots0) \rightarrow (0100\cdots0)$, and in the $1 \rightarrow 0$ direction. ■

It follows that the data movement for successive matrix multiplications can be pipelined. The directed routing paths for a 4-cube are shown in Fig. 10.

THEOREM 5.4. *The computation* $C \leftarrow C + A \times B$ *on a boolean* $2k$*-cube can be performed by algorithm MMC*3 *in* $\lceil N_1/2^k\rceil\lceil N_2/2^k\rceil\lceil N_3/2^k\rceil 2^k + 2k - 1$ *communication and* $\lceil N_1/2^k\rceil\lceil N_2/2^k\rceil\lceil N_3/2^k\rceil 2^k$ *arithmetic steps with* A *an* $N_1 \times N_2$ *matrix and* B *an* $N_2 \times N_3$ *matrix, and the matrices stored cyclically or consecutively in a* $2^k \times 2^k$ *array embedded in the cube by a binary-reflected Gray code encoding of row and column indices, each in half of the address space.*

The proof of the theorem is immediate from Lemma 5.2.

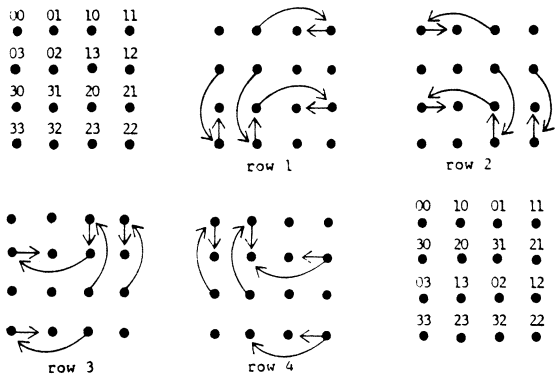

FIG. 10. Routing paths in a 4-cube for the multiplication by reflection in row/column 0.

Remark. The multiplication-by-reflection algorithm can also be employed for matrices embedded by a binary encoding of row-and-column indices. Distribution from the processors embedding row-and-column 0 is made as in the case of Gray code encoded matrices, but elements are transported to the processors of reflection-by-exchange operations according to the transition sequence in a binary-reflected Gray code, as in the algorithm by Dekel *et al.* Note, however, that in this algorithm some of the interprocessor communication links used by the SBT algorithm are also used in transporting matrix elements to the processors storing row-and-column 0.

Remark. Note that in the multiplication algorithm based on reflection, the synchronization requirement between the two dimensions yields a *latency* term of $2k$. In the algorithm employing rotation, and in the algorithm by Dekel *et al.*, there is no coupling between the dimensions in the set-up phase and the latency is reduced to k communication steps.

5.4. *An Outer-Product Algorithm with Minimal Communication, MMC4*

Algorithm MMC3 can be viewed as an outer-product algorithm with all outer products initiated in processor 0. The outer products $a_{*m}b_{m*}$ are performed in order of successively increasing values of m starting with $m = 0$. In order to reduce the data movement for the computation of the outer products the distribution of the row and column elements can be performed by a spanning tree algorithm directly from the processors where the elements are stored.

5.4.1. *Outer-Product Algorithm Version 1, MMC4.1*

In this algorithm each processor performs all rank-1 updates $a_{*m}b_{m*}$ in order of increasing index m. This order is mandatory for factorization algorithms with pivoting on the diagonal.

LEMMA 5.3. *The computation* $C \leftarrow C + A \times B$ *on a boolean* $2k$*-cube, where* A *and* B *are* $2^k \times 2^k$ *matrices embedded by Gray code encoding, requires* $3(2^k - 1) + 2k$ *communication steps if distribution of matrix elements is performed by the SBT algorithm, outer products are computed in order of successively increasing indices, and the communication of matrix elements for successive outer products is completed in order.*

Proof. Spanning binomial trees rooted in adjacent processors can be initiated only in every other communication step, if a processor is to complete the communication for a subtree before initiating the communication of a new subtree. Still another step is required for the communication from the subcube holding the row and column of the current outer product to the one holding the row and column of the next outer product. Hence, three communication steps are required between the initiations of successive outer products, and the column and row trees require k steps each. ■

If A is an $N_1 \times N_2$ matrix and B an $N_2 \times N_3$ matrix, then it is possible to increase the resource utilization through pipelining of the data movement.

THEOREM 5.5. *The computation* $C \leftarrow C + A \times B$ *on a boolean* $2k$*-cube can be performed by algorithm MMC4 in* $(\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil + 2)2^k + 2k - 3$ *communication and* $\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil 2^k$ *arithmetic steps with* A *an* $N_1 \times N_2$ *matrix and* B *an* $N_2 \times N_3$ *matrix, and the matrices stored cyclically or consecutively in a* $2^k \times 2^k$ *array embedded in the cube by a binary-reflected Gray code encoding of row and column indices, each in half of the address space.*

5.4.2. *Outer-Product Algorithm Version 2, MMC4.2*

In this algorithm outer products are initiated in order of successively increasing indices during successive communication steps. The number of such steps is reduced by approximately a factor of 3 for matrices of size equal to the ensemble size.

THEOREM 5.6. *The computation* $C \leftarrow C + A \times B$ *on a boolean* $2k$*-cube can be performed by initiating outer products in order of successively increasing indices during successive communication steps in* $\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil 2^k + 2k - 1$ *communication and* $\lceil N_1/2^k \rceil \lceil N_2/2^k \rceil \lceil N_3/2^k \rceil 2^k$ *arithmetic steps with* A *an* $N_1 \times N_2$ *matrix and* B *an* $N_2 \times N_3$ *matrix, and the matrices stored cyclically or consecutively in a* $2^k \times 2^k$ *array embedded in the cube by a binary-reflected Gray code encoding of row and column indices, each in half of the address space.*

Proof. We assume that the distribution from each source node is performed by the SBT algorithm, and initiated during communication step $m + |G(k)_m \oplus G(k)_j|$ for element b_{mj} and step $m + |G(k)_m \oplus G(k)_i|$ for element

a_{im}. Then elements a_{im} and b_{mj} arrive at the processors storing element c_{ij} during step $m + |G(k)_m \oplus G(k)_j| + |G(k)_m \oplus G(k)_i|$ for $(i, j) = \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\}$.

The proof of the lack of communication conflicts is by induction. It is easily verified to be true for $k = 1$. Assume it is true for $k = r$. Then for $k = r + 1$ there are three new subcubes. We refer to the old subcube as the 00 cube and to the new subcubes as the 01, 10, and 11 subcubes. The communication of the elements of A residing in subcube 00 to the subcube with highest column index 1 is entirely in the $0 \rightarrow 1$ direction on the links in the added dimension for column encoding. The spanning trees for the elements of B in the 01 subcube are generated in the same order as the spanning trees for B in the 00 subcube, but are initiated one communication step later than the corresponding ones in the 00 subcube, since an element of A reaches the node in the 01 subcube one step after its image node in the 00 cube, by virtue of the SBT algorithm. The spanning trees for the elements of B in rows $0 - 2^r - 1$ and columns $2^r - 2^{r+1} - 1$ do not have any directed edges in common by the induction assumption, and the edges of the spanning trees to the 11 cube are in the $0 \rightarrow 1$ direction for the new dimension for the rows. The same is true for the spanning tree edges to subcube 10, for the elements of B in rows $0 - 2^r - 1$ and columns $0 - 2^r - 1$. The distribution of the elements of A in rows $2^r - 2^{r+1} - 1$ and columns $0 - 2^r - 1$ are initiated one step after their corresponding element in subcube 00, and there is no conflict within the 10 subcube according to the induction assumption. The communication to the 11 subcube is in the $0 \rightarrow 1$ direction of the new dimension for columns. Hence, the matrix elements in the first 2^r outer products traverse cube edges during different cycles (but up to 2^r elements traverse the same edge). The proof is completed by observing that the communication of the elements of B in rows $2^r - 2^{r+1} - 1$, and for all columns, is in the $1 \rightarrow 0$ direction for the new dimension for rows. Similarly, the communication for the elements of A in columns $2^r - 2^{r+1} - 1$ is in the $1 \rightarrow 0$ direction in the added dimension for columns. No conflicts occur within subcube 11, due to the induction assumption, and the proof is complete. ■

6. Solving Linear Systems of Equations

6.1. *Gauss–Jordan Elimination*

If the cube is of a size that matches the matrix dimensions, then the system $AX = B$ can be solved by Gauss–Jordan elimination in approximately the same time as is required for LU-decomposition alone. Gauss–Jordan elimination fully utilizes the processors, whereas LU-decomposition does not. We assume that the matrix A is embedded in the cube by a separate Gray code

encoding of row and column indices, and store the inverse in product form with the elements of the factors replacing the corresponding elements of A.

$$A^{-1} = J^{N-1}J^{n-2}\cdots J^0, \qquad J^j = \begin{pmatrix} 1 & & & f_{0j} & & \\ & 1 & & f_{1j} & & \\ & & \ddots & \vdots & & \\ & & & 1 \quad f_{j-1j} & & \\ & & & f_{jj} & & \\ & & & f_{j+1j} \quad 1 & & \\ & & & \vdots & \ddots & \\ & & & f_{N-1j} & & 1 \end{pmatrix}$$

The elements of a factor are the result of the application of all preceding factors to the matrix A. The application of a factor can be initiated as soon as an element of it is known, and the application of successive factors can be pipelined [30, 18]. In the application of the jth factor a multiple of the jth row is added to all other rows. The multiple is determined by the elements of the jth column. Let $A^0 = J^0A$ and $A^j = J^jA^{j-1}$. The application of each factor J^j is a rank-1 update of A^{j-1}; an outer product is formed and added to A^{j-1}, with the exception of row j.

COROLLARY 6.1. *Gauss–Jordan elimination on a $2^k \times 2^k$ matrix embedded in a 2k-cube by Gray code encoding of row and column indices can be performed in $3(2^k - 1) + k$ communication steps and $2(2^k + k - 1)$ arithmetic steps, assuming one arithmetic operation per processor and step.*

The communication complexity is lower than that of multiplication by algorithm MMC4.1 by k since the row length at the end of Gauss–Jordan elimination is 1 instead of 2^k.

Remark. If partial pivoting is performed, then the communication complexity increases to order $O(k2^k)$.

6.2. *Triangulation of an $N \times N$ System and Forward Elimination*

In order to maximize the processor utilization we assume that the concatenated matrix AB is stored cyclically in an array of size $2^k \times 2^k$ that in turn is embedded in the 2k-cube by a separate Gray code encoding of row and column indices. With consecutive storage the processors embedding the first row and column become idle after the elimination of the first $N/2^k$ variables by Gaussian elimination. Additional processors become idle for every set of $N/2^k$ variable eliminations. In Gauss–Jordan elimination and consecutive storage, 2^k processors (that store columns) become idle for every $N/2^k$ variable eliminations.

In the algorithm analyzed below, pivoting on the diagonal is assumed. Gauss–Jordan elimination is performed on the pivot block row, and Gaussian elimination on the rows below the block row. The factor J^j is of the form

$$J^j = \begin{pmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & 1 & & f_{\lfloor j/2^k \rfloor 2^k j} & & & \\ & & & & 1 & f_{\lfloor j/2^k \rfloor 2^k + 1 j} & & & \\ & & & & & \ddots & & & \\ & & & & & f_{jj} & & & \\ & & & & & \vdots & \ddots & & \\ & & & & & \vdots & & \ddots & \\ & & & & & f_{N-1 j} & & & 1 \end{pmatrix}$$

and $(AB)^j = J^j(AB)^{j-1}$ is of block form, with blocks of size $2^k \times 2^k$ as illustrated in Fig. 11.

Denote the block rows of $(AB)^{j-1}$ containing column j for rows i, $\lfloor j/2^k \rfloor 2^k \leq i \leq N-1$, by $p = \{\lfloor j/2^k \rfloor, \ldots, \lfloor (N-1)/2^k \rfloor\}$ and similarly the block columns by $q = \{\lfloor j/2^k \rfloor, \ldots, \lfloor (N+R-1)/2^k \rfloor\}$. Then $J^j_{ij} = (AB)^{j-1}_{ij}$, $i = \{\lfloor j/2^k \rfloor 2^k, \ldots, N-1\}$, and the computations in the application of J^j for $p > \lfloor j/2^k \rfloor$, $q > \lfloor j/2^k \rfloor$ is a complete outer product. For $p = q = \lfloor j/2^k \rfloor$ the computations are the same as in the Gauss–Jordan elimination described above. For $p = \lfloor j/2^k \rfloor$, $q > \lfloor j/2^k \rfloor$ the computations are identical to an outer-product computation with the exception of row j. For $p > \lfloor j/2^k \rfloor$, $q = \lfloor j/2^k \rfloor$ the computations are outer-product computations for columns $j+1$ through $(\lfloor j/2^k \rfloor + 1)2^k - 1$.

For the diagonal blocks the Gauss–Jordan algorithm described previously can be used. For the blocks below the diagonal block, division is performed on the elements of the pivot column, and the resulting factors are distributed to all 2^k processors embedding the corresponding row. A processor has to distribute $\lfloor (N-j-1)/2^k \rfloor$ factors plus one factor for the diagonal block.

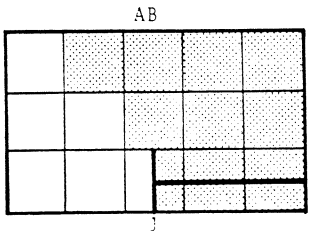

FIG. 11. The partially factored matrix AB.

Similarly, there are $\lfloor (N + R - j - 1)/2^k \rfloor$ elements of row j for off-diagonal blocks to be distributed throughout each subcube storing a column, plus one element for the diagonal block.

The computations on different blocks can be pipelined in a manner similar to the case of matrix multiplication. For the triangulation and forward elimination on the right-hand side we derive the following:

THEOREM 6.1. *The factorization of an $N \times N$ matrix on a $2k$-cube can be performed in* $(\lceil (N + R)/2^k \rceil^2 - (\lceil R/2^k \rceil + 1)^2 + 5\lceil (N + R)/2^k \rceil - 3(\lceil R/2^k \rceil + 1) + 4)/2 + k - 3$ *communications and* $(3\lceil (N + R)/2^k \rceil - \lceil R/2^k \rceil + 1)\lceil R/2^k \rceil \times (\lceil R/2^k \rceil + 1)/6$ *arithmetic operations.*

6.3. *QR-Factorization*

The data movement of QR-factorization using Given's rotations on a mesh can be made identical to that of LU-decomposition by using row j to eliminate all elements in column j [19]. In adapting this algorithm to a boolean cube, the row used to eliminate the elements in column j is distributed in a linear fashion, since it is changed in each elimination operation. The rotation factors can still be distributed by a spanning tree algorithm. In computing the upper triangular factor R for a $2^k \times 2^k$ matrix on a $2k$-cube the communication complexity is the same as for LU-decomposition.

6.4. *Solving Triangular Systems of Equations*

We analyze three algorithms for the solution of triangular systems of equations $Lz = y$ (y and z are vectors, L a lower triangular matrix) on a boolean cube. One algorithm is for strictly triangular matrices, another for block triangular matrices with identity diagonal blocks. The second algorithm makes use of a spanning tree algorithm for broadcasting of z_i to rows $0, 1, \ldots, i - 1$ and for the additions in the inner-product computation. The third algorithm also applies to block tridiagonal matrices and makes use of a matrix multiplication algorithm.

6.4.1. *Accumulation of Inner Products Partially* in-Space, *Partially* in-Place

The following algorithm is a variation of the column-sweep algorithm of Kuck [29]. Assume that L is a $2^k \times 2^k$ lower triangular matrix stored in a $2k$-cube by Gray code encoding of row and column indices, and that y is initially stored in the set of processors encoding column 0. Then z_0 is computed in the processor embedding l_{00} and broadcast in the subcube storing column 0 of L. After one communication, the product l_{10} can be subtracted from y_1 and the result communicated to the processor storing l_{11}. The results of the computation $y^0 = y - l_{*0} z_0$ is communicated to the set of processors storing column 1. The communications for a new variable can be initiated every

two communication steps. Column j computes $y^j = y^{j-1} - l_{*j}z_j$. Employing the same broadcasting algorithm for all broadcasts, for instance, the SBT algorithm, guarantees that a newly computed variable never arrives at a processor in fewer than, but possibly the same as, the number of communications as the partial product sum from an adjacent column. The total number of communications is $2(2^k - 1)$.

For L equal to $N \times N$ the number of communications in sequence can be kept the same for every set of 2^k variables, if the fraction of the right-hand side corresponding to the diagonal $2^k \times 2^k$ block for which a solution is being computed is communicated first, followed only by the set of partial products required for updating the right-hand side for the next set of variables to be computed. Partial products are accumulated *in-place*, requiring storage of a maximum of $\lfloor N/2^k \rfloor$ partial product terms per processor, until they need to be accumulated *in-space* in preparation for the computation of the next set of variables. Figure 12 attempts to illustrate the communication in this triangular system solver.

THEOREM 6.2. *The solution of a triangular linear system $LZ = Z$, where L is $N \times N$ and stored cyclically in a $2^k \times 2^k$ array embedded in a $2k$-cube by a binary-reflected Gray code, can be performed in $2(\lceil N/2^k \rceil 2^k - 1)$ communications in sequence, and $\lceil N/2^k \rceil^2 + 3(2^k - 1)$ arithmetic operations.*

Remark. If instead of the matrix L its transpose L^T is stored by a Gray code encoding of row and column indices, then the components of z are broadcast within subcubes storing columns of L (which now corresponds to rows of the array in which L^T is stored cyclically) and partial products are accumulated within subcubes storing rows of L. y is communicated along rows of the embedded array and the processors storing the diagonal elements have to subtract the partial product sum arriving along a column from the component of y.

For multiple right-hand sides the communication complexity does not increase as long as $2R \leq 2 \times 2^k$. The first term is the number of communica-

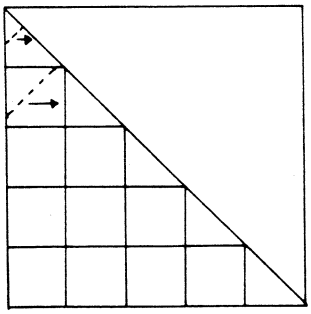

FIG. 12. Communication in a linear recurrence solver.

tions needed for a given variable for all right-hand sides and the second term is the number of communications occurring between the instances when a given processor is in the same state with respect to the computations in two successive sets of 2^k variables. The arithmetic complexity increases proportionally.

THEOREM 6.3. *The solution of a triangular linear system* $LZ = Y$, *where* L *is* $N \times N$ *and* Y *is* $N \times R$ *and stored cyclically in a* $2^k \times 2^k$ *array embedded by a binary-reflected Gray code in a* $2k$ *cube, can be obtained in* $\max(2 \times 2^k, 2R)(\lceil N/2^k \rceil - 1) + 22^k + R - 3$ *communications and* $(\lceil N/2^k \rceil^2 + 3(2^k - 1))R$ *arithmetic operations by an algorithm that partially accumulates partial products* in-place *partially* in-space *by means of pipelined spanning tree algorithms.*

6.4.2. *Solving a Block Triangular Linear System*

If L has identity matrices on the diagonal, as is the case if Gauss–Jordan elimination is used for the diagonal blocks, then the first 2^k z-values are known without computation. The distribution of these 2^k z-values to the corresponding subcubes can be made in k routing steps, and the distribution within the subcubes in an additional k routing steps. The accumulation of the inner products requires an additional k routing steps. The accumulation for different z-values can be made in the appropriate processor column so that only $2k$ routing steps are needed per 2^k z-values. The right-hand side can be shifted into position concurrently with other communications. Component j is shifted $j \bmod 2^k$ steps. As in the column sweep algorithm described above, only the partial products needed for the next set of z-values need to be accumulated *in-space;* the remaining are accumulated *in-place.* Hence, the total number of communications is $(\lceil N/2^k \rceil - 1)2k + k$ and the number of arithmetic operations is $\lceil N/2^k \rceil(\lceil N/2^k \rceil - 1)$. For R right-hand sides and pipelining of the communications for different right-hand sides we arrive at the following complexity result.

THEOREM 6.4. *The solution of a block triangular linear system* $LZ = Y$ *with identity blocks on the diagonal and where* L *is* $N \times N$ *and* Y *is* $N \times R$ *and stored cyclically in a* $2^k \times 2^k$ *array embedded by a binary-reflected Gray code in a* $2k$ *cube can be obtained in* $(\lceil N/2^k \rceil - 1)(2k + R - 1) + k$ *communications and* $\lceil N/2^k \rceil(\lceil N/2^k \rceil - 1)R$ *arithmetic operations by an algorithm that partially accumulates partial products* in-place *partially* in-space *by means of pipelined spanning tree algorithms.*

6.4.3. *Accumulation of Inner Products Entirely* in-Place

We restrict the analysis to the case where the triangular matrix L has identity blocks on the diagonal. Then, solving the triangular system of equations

is essentially a matrix–vector multiplication problem for a single right-hand side and a matrix–matrix multiplication problem for multiple right-hand sides. Any one of the multiplication algorithms described previously can be used. Each set of 2^k variables is computed through matrix multiplications of the form $Y \leftarrow Y - L(i) \times X(i)$, where $L(i)$ is the ith block column of L, $X(i)$ is the ith block row of X, and Y is an $\lceil N/2^k \rceil - i \times \lceil R/2^k \rceil$ block matrix. The total number of $2^k \times 2^k$ matrix multiplications required to compute the solution X is $\lceil R/2^k \rceil \lceil N/2^k \rceil (\lceil N/2^k \rceil - 1)/2$, each requiring 2×2^k arithmetic operations and 2^k communications.

THEOREM 6.5. *The solution of a block triangular linear system $LZ = Y$ with identity blocks on the diagonal and where L is $N \times N$ and Y is $N \times R$ and stored cyclically in a $2^k \times 2^k$ array embedded by a binary-reflected Gray code in a $2k$ cube can be obtained in $\lceil N/2^k \rceil (\lceil N/2^k \rceil - 1) \lceil R/2^k \rceil 2^{k-1} + \max((\lceil N/2^k \rceil - 1)/2, \lceil R/2^k \rceil) \lceil N/2^k \rceil + k - 1$ communications and $\lceil N/2^k \rceil (\lceil N/2^k \rceil - 1) \lceil R/2^k \rceil 2^k$ arithmetic operations by employing a matrix multiplication algorithm.*

Comparing the communication complexity with that of the algorithm in which inner products are accumulated partially *in-space* we note that a straightforward application of a matrix multiplication algorithm yields a higher communication complexity. If a matrix multiplication algorithm is used, then the same values of the solution are communicated multiple times, which is not the case for the preceding algorithms.

ACKNOWLEDGMENTS

Thanks go to Sandeep Bhatt for many discussions on the embeddings of binary trees in hypercubes, for the main ideas in the proof of the dilation 1 embedding of complete binary trees in hypercubes, and for valuable suggestions upon reading the manuscript. Ching-Tien Ho contributed ideas on the multiplication-by-reflection algorithm. Many thanks also go to Andrea Pappas and Chris Hatchell for their assistance with the manuscript.

REFERENCES

1. Aleliunas, R., and Rosenberg, A. L. On embedding rectangular grids in square grids. *IEEE Trans. Comput.* **C-31**, 9 (Sept. 1982), 907–913.
2. Batcher, K. E. Sorting networks and their applications. In *Spring Joint Computer Conference*. IEEE, New York, 1968, pp. 307–314.
3. Bhatt, S. N., and Ipsen, I. C. F. How to embed trees in hypercubes. Tech. Rep. YALEU/CSD/RR-443, Department of Computer Science, Yale University, Dec. 1985.
4. Cannon, L. E. A cellular computer to implement the Kalman filter algorithm. Ph.D. thesis, Montana State University, 1969.
5. Dekel, E., Nassimi, D., and Sahni, S. Parallel matrix and graph algorithms. *SIAM J. Comput.* **10** (1981), 657–673.

6. Desphande, S. R., and Jenevin, R. M. Scaleability of a binary tree on a hypercube. Tech. Rep. TR-86-01, University of Texas at Austin, Jan. 1986.
7. Dongarra, J. J., Gustafson, F. G., and Karp, A. Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Rev.* **26,** 1 (Jan. 1984), 91–112.
8. Eklundh, J. O. A fast computer method for matrix transposing. *IEEE Trans. Comput.* **C-21,** 7 (1972), 801–803.
9. Flynn, M. J. Very high-speed computing systems. *Proc. IEEE* **12** (1966), 1901–1909.
10. George, A. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.* **10** (1973), 345–363.
11. Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K. P., Rudolph, L., and Snir, M. The NYU ultracomputer—Designing an MIMD shared memory parallel computer. *IEEE Trans. Comput.* **C-32,** 2 (1983), 175–189.
12. Hillis, W. D. The connection machine. Tech. Rep. Memo 646, MIT Artificial Intelligence Laboratory, 1981.
13. Hillis, W. D. *The Connection Machine.* MIT Press, New Haven, CT, 1985.
14. Ho, C.-T., and Johnsson, S. L. Tree embeddings and optimal routing in hypercubes. Tech. Rep. YALEU/CSD/RR-, Yale University, Department of Computer Science, in preparation.
15. Ho, C.-T., and Johnsson, S. L. On the embedding of meshes in Boolean cubes. Tech. Rep. YALEU/CSD/RR-, Department of Computer Science, Yale University, in preparation.
16. Hwang, K., and Briggs, F. A. (Eds.). *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984.
17. Hwang, K. (Ed.). *Supercomputers: Design and Applications.* IEEE Comput. Soc., New York, 1984.
18. Johnsson, S. L. Computational arrays for band matrix equations. Tech. Rep. 4287:TR:81, Computer Science, California Institute of Technology, May 1981.
19. Johnsson, S. L. Pipelined linear equation solvers and VLSI. *Proc. Microelectronics '82,* Institution of Electrical Engineers, Australia, May 1982, pp. 42–46.
20. Johnsson, S. L. Combining parallel and sequential sorting on a Boolean n-cube. In *International Conference on Parallel Processing.* IEEE Comput. Soc., New York, 1984, pp. 444–448. Presented at the 1984 Conf. on Vector and Parallel Processors in Computational Science II.
21. Johnsson, S. L. Odd–even cyclic reduction on ensemble architectures and the solution tridiagonal systems of equations. Tech. Rep. YALEU/CSD/RR-339, Department of Computer Science, Yale University, Oct. 1984.
22. Johnsson, S. L. Solving narrow banded systems on ensemble architectures. *ACM Trans. Math. Software* **11,** 3 (Nov. 1985), 271–288. Also available as Report YALEU/CSD/RR-418, Nov. 1984.
23. Johnsson, S. L. Communication efficient basic linear algebra computations on hypercube architectures. Tech. Rep. YALEU/CSD/RR-361, Department of Computer Science, Yale University, Jan. 1985.
24. Johnsson, S. L. Fast banded systems solvers for ensemble architectures. Tech. Rep. YALEU/CSD/RR-379, Department of Computer Science, Yale University, Mar. 1985.
25. Johnsson, S. L. Data permutations and basic linear algebra computations on ensemble architectures. Tech. Rep. YALEU/CSD/RR-367, Department of Computer Science, Yale University, Feb. 1985.
26. Johnsson, S. L., and Ho, C.-T. Matrix transpose on Boolean cubes. Tech. Rep. YALEU/CSD/RR-, Department of Computer Science, Yale University, in preparation.

27. Johnsson, S. L., Weiser, U., Cohen, D., and Davis, A. Towards a formal treatment of VLSI arrays. *Proc. Second Caltech Conference on VLSI,* Caltech Computer Science Department, Jan. 1981, pp. 375–398.
28. Kogge, P. M., and Stone, H. S. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.* **C-22,** 8 (1973), 786–792.
29. Kuck, D. J. A survey of parallel machine organization and programming. *ACM Comput. Surveys* **9,** 1 (1977), 29–59.
30. Kung, H. T., and Leiserson, C. L. Algorithms for VLSI processor arrays. Introduction to *VLSI Systems.* Addison-Wesley, Reading, MA, 1980, pp. 271–292.
31. Leighton, F. T. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, New Haven, CT, 1983.
32. Leiserson, C. E. Area-efficient graph layouts (for VLSI). *Proc. 21st IEEE Symp. Foundations Comput. Sci.* IEEE Comput. Soc. New York, 1980, pp. 270–281.
33. Leiserson, C. E., Rose, F. M., and Saxe, J. B. Optimizing synchronous circuitry by retiming. In *Third Caltech Conference on Very Large Scale Integration.* Comput. Sci. Press, Rockville, MD, 1983, pp. 87–116.
34. Madsen, N. K., Rodrigue, G. H., and Karush, J. I. Matrix multiplication by diagonals on a vector/parallel processor. *Inform. Process. Lett.* **5,** 2 (1976), 41–45.
35. Madsen, N. K., and Rodrigue, G. H. Odd–even reduction for pentadiagonal matrices. In *Parallel Computers—Parallel Mathematics,* North-Holland, Amsterdam, 1977, pp. 103–106.
36. Reingold, E. M., Nievergelt, J., and Deo, N. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1977.
37. Saad, Y., and Schultz, M. H. Topological properties of hypercubes. Tech. Rep. YALEU/DCS/RR-389, Department of Computer Science, Yale University, June 1985.
38. Saied, F. M., and Johnsson, S. L. Concurrency in sparse elimination. Tech. Rep. YALEU/CSD/RR-, Department of Computer Science, Yale University, Sept. 1985.
39. Schwartz, J. T. Ultracomputers. *ACM Trans. Program. Languages and Systems* **2** (1980), 484–521.
40. Seitz, C. L. The cosmic cube. *Comm. ACM* **28,** 1 (1985), 22–33.
41. Stone, H. S. Parallel processing with the perfect shuffle. *IEEE Trans. Comput.* **C-20** (1971), 153–161.
42. Thompson, C. D. Area-time complexity for VLSI. *Proc. 11th ACM Symposium on the Theory of Computing.* ACM, New York, 1979, pp. 81–88.
43. Thompson, C. D. A complexity theory for VLSI. Tech. Rep., Department of Computer Science, Carnegie-Mellon University, 1980.