ELSEVIER

Parallel Computing 21 (1995) 243–264

PARALLEL COMPUTING

Transposition of banded matrices in hypercubes: A nearly isotropic task [†]

Emmanouel A. Varvarigos [a], Dimitri P. Bertsekas [b]

[a] *Electrical and Computer Engineering, University of California, Santa Barbara, CA 093106, USA*
[b] *Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

Received 30 November 1992; revised 3 June 1993, 4 February 1994

Abstract

A class of communication tasks, called isotropic, was introduced in [15], and minimum completion time algorithms for all tasks in this class were found. Isotropic tasks are characterized by a type of symmetry with respect to origin node. In this paper we consider the problem of transposing a sparse matrix of size $N \times N$ with a diagonal band of size $2^{\beta+1}+1$, which is stored by columns in a hypercube network of $N = 2^d$ processors. We propose an assignment of matrix columns to hypercube nodes such that the transposition becomes a 'nearly isotropic' task, that is, it looks 'almost identical' to all nodes. Under this assignment, we give an algorithm to transpose the matrix in 2^{β} steps. We prove that the algorithm given is optimal over all affine assignments of columns to processors. We also derive a lower bound on the minimum number of steps required to transpose a banded matrix, which holds for any possible assignment of matrix columns to hypercube processors. In the case that $2^{\beta+1}+1 = \Theta(N^c)$, for some constant $c \in (0, 1]$, we prove that the completion time of our transposition algorithm is of the same order of magnitude with the lower bound. We further show that $\lfloor d/\beta \rfloor$ banded matrices, each of bandwidth $2^{\beta+1}+1$, can be stored by columns in a hypercube so that all of them can be concurrently transposed in $2^{\beta+1}$ steps. Finally, we modify our algorithms so that they apply to arbitrary matrix bandwidths and multiple column storage by each processor, while maintaining their efficiency.

Keywords: Linear algebra; Banded matrices; Communication algorithm; Transposition; Hypercubes; Isotropic task

[†] Research supported by NSF under Grant NSF-DDM-8903385 and by the ARO under Grant DAAL03-86-K-0171.

[*] Corresponding author. Email: bertsekas@lids.mit.edu

1. Introduction

Routing algorithms have been studied by several authors under a variety of assumptions on the communication network connecting the processors of a parallel computing system. Saad and Shultz [11–13] have introduced a number of generic communication problems that arise frequently in numerical and other methods. For example they consider the problem where each processor is required to send a separate packet to every other node; following [2], we call this the *total exchange* problem. Saad and Schultz have assumed that all packets take unit time to traverse any communication link. Johnsson and Ho [8] have developed minimum and nearly minimum completion time algorithms for similar routing problems as those of Saad and Schultz but using a different communication model and a hypercube network. Bertsekas et al. [3], Bertsekas and Tsitsiklis [2], and Edelman [4] have used the communication model of Saad and Schultz to derive minimum completion time algorithms for several communication problems in a hypercube.

Varvarigos and Bertsekas [15] introduced a class of communication tasks, called *isotropic*, which are characterized by transmission requirements that are symmetric with respect to origin node (a precise definition will be given later). For example, the total exchange problem is an isotropic task; the communication problem 'looks identical' to every node. The structure of isotropic tasks can be exploited particularly well in networks that have themselves a symmmetric structure, such as a hypercube and a wraparound mesh. A central result of [15] is that executing isotropic tasks is equivalent to solving a matrix decomposition problem. Using this fact one can find algorithms to execute such tasks in minimum time.

In this paper we turn our attention to 'nearly isotropic' tasks. The quotes are used to indicate that this is a class of tasks with no specific or rigorously defined borders. The term is used in a heuristic way to refer to communication tasks for which the tools, algorithms and ideas developed for isotropic tasks can also be useful. Loosely speaking, a 'nearly isotropic' task is a task which looks almost identical to all nodes. It is clear that there is an incentive to formulate new routing problems in terms of 'nearly isotropic' tasks, whenever this is reasonable, to take advantage of the corresponding simple and elegant analysis.

We focus on a particular 'nearly isotropic' task, namely the transposition of a banded matrix of size $N \times N$ stored by columns (or by rows) in a hypercube network of $N = 2^d$ processors. We show that for a particular assignment of matrix columns to hypercube processors the task is close to being isotropic. We then give an algorithm to transpose a $2^{\beta+1} + 1$-diagonal matrix in 2^β steps.

Algorithms for the transposition of matrices in a hypercube have been proposed in [7], and [6]. Johnsson and Ho [7] devised several algorithms for the case where the matrix is stored in a hypercube so that each processor is assigned a column (or a number of columns) of the matrix, using a binary or a Gray encoding assignment of columns to processors. Ho and Ragnunath [6] treated the case where a block partitioned matrix is stored in a hypercube, so that each processor stores a block of the matrix. Both of these papers consider fully-dense matrices. In this paper we will assume that the matrix to be transposed is banded, and it is stored in the hypercube so that each processor stores a column (or a row) of the matrix. For such matrices, we provide transposition algorithms that are faster than their dense matrix counterparts. In particular, the transposition of a dense $N \times N$ matrix takes $N/2$ steps (with our communication model), while the transposition of B-diagonal matrix of the same size can be performed in at most $B-2$ steps (which is independent of N), and even faster for particular values of B. In some cases it is desirable to store a matrix so that a processor stores more than one column. We modify our algorithms to deal with this case, while maintaining their efficiency.

We will say that an algorithm is *of optimal order* if its worst case time complexity is asymptotically within a constant factor from the optimal completion time itself. We generally prove that an algorithm is of optimal order by showing that the leading term of its worst case time complexity is some multiple of the leading term of an expression which is a lower bound to the time required by any algorithm. We generally derive the optimal completion time by deriving a lower bound to the completion time of any algorithm and by constructing an algorithm that attains the lower bound; this latter algorithm is said to be *optimal.*

We have not proved that our banded matrix transposition algorithm is of optimal order over *all* possible column-to-processor assignment and for *any* matrix bandwidth. The algorithm is optimal for the proposed column-to-processor assignment, but it is possible that better assignments exist. We prove, however, that it is impossible to do better if the matrix is stored in the hypercube using an affine transformation (see [5]) for the assignment of columns to processors. The binary and the Gray encoding assignments that are the two most often used in practice are two such possibilities. Embeddings that use transformations involving shuffles, bit reversal dimension permutation, or combinations of these, also fall in this category. We show that the algorithm that we propose is optimal (for $B = 2^{\beta+1}+1$) or at most twice the optimal (for general B) over all transposition algorithms that use affine assignments of columns to processors. We also derive a lower bound showing that when the bandwidth B is $\Theta(N^c)$ for some constant $c \in (0, 1]$, the time complexity of our algorithm is of the optimal order, where optimality is considered over all (not only affine) column-to-processor assignments, and over all transposition algorithms for each particular assignment. Moreover, our algorithm is optimal when the matrix is dense, and of the optimal order when the bandwidth B is $O(1)$. We also give a way to store $\lfloor d/\beta \rfloor$ matrices, each of bandwidth $2^{\beta+1}+1$, in a hypercube so that they can be can be concurrently transposed in time $2^{\beta+1}$. When $\lfloor d/\beta \rfloor > 2$ the improvement in efficiency obtained in this way may be significant.

The paper is organized as follows. Section 2 reviews the basic results regarding isotropic tasks, as derived in [16] and [15]. Section 3 describes the banded-matrix transposition algorithm. In Section 4 we find lower bounds on the time required to transpose a banded matrix. In Subsection 4.1 we give a lower bound that holds for the class of affine assignments of columns to processors. In Subsection 4.2 we derive a universal lower bound that holds for any column-to-processor assignment, and we prove the optimality of our algorithm for a broad range of matrix bandwidths. Finally, in Section 5 we give an algorithm to transpose many banded matrices simultaneously.

2. Optimal algorithms for isotropic tasks

We first introduce some terminology. Given a hypercube with 2^d nodes, the *j-type* link (or j-link) of node $s = (s_{d-1} \ldots s_j \ldots s_0)$ is the link connecting nodes $(s_{d-1} \ldots s_j \ldots s_0)$ and $(s_{d-1} \ldots \bar{s}_j \ldots s_0)$. (We denote by $\bar{x}$ the complement of the binary number x, that is, $\bar{x} = 1 - x$.) Given two nodes s and t, the node $s \oplus t$ is the node with binary representation obtained by a bitwise exclusive OR operation of the binary representations of nodes s and t.

Information is transmitted along the hypercube links in groups of bits called *packets*. We assume that the time required to cross any link is the same for all packets, and is taken to be one time unit also called a step. Only one packet can travel along a link in each direction at any one time; thus, if more than one packet are available at a node and are scheduled to be transmitted on the same incident link of the node, then only one of these packets can be transmitted at the next time period, while the remaining packets must be stored at the node while waiting in a queue of infinite capacity. We assume that all incident links of a node can be used simultaneously for packet transmission and reception. Finally, we assume that each of the algorithms proposed in this paper is simultaneously initiated at all processors.

Definition 1. A *communication task* $\mathcal{G}$ is defined as a set of triplets (s, v, k), where s is a node (source), v is a node (destination), and k is a positive integer (the number of packets whose source is s and whose destination is g).

Definition 2. A communication task $\mathcal{G}$ is called *isotropic* if for each packet that node s has to send to node v, there is a corresponding packet that node $s \oplus x$ has to send to node $v \oplus x$, where s, v, and x are arbitrary nodes. Mathematically:

$$(s, v, k) \in \mathcal{G} \Rightarrow \quad \text{for all nodes } x \text{ we have } (s \oplus x, v \oplus x, k) \in \mathcal{G}.$$

An example of an isotropic task is the total exchange, were $\mathcal{G}$ consists of all the triplets $(s, v, 1)$ as s and v range over all the pairs of distinct nodes [one packet for every origin-destination pair (s, v)].

In the algorithms that we propose, the packets carry with them a d-bit string called a *routing tag*. The routing tag of a packet is initially set at $s \oplus v$, where s is the source and v is the destination of the packet. As the packet is transmitted from node to node, its routing tag changes. If at time t a packet resides at a node u and has v as destination, then its routing tag is $u \oplus v$. For example, a packet that is currently at node 001010 and is destined for node 101000, has routing tag 100010.

An important data structure that will be used by our routing algorithms is that

1	1	1
0	1	1
1	0	1
1	1	0
1	0	0
0	1	0
0	0	1

Fig. 1. The task matrix for the total exchange problem has $N-1$ rows and d columns. The figure illustrates the case where $d=3$.

of the *task matrix* of node s at time t, which will be denoted by $T_t(s)$. The task matrix $T_t(s)$ is defined for both isotropic and non-isotropic tasks and is binary matrix whose rows are the routing tags of all the packets that are queued at node s at time t. The routing tags appear as rows of the initial task matrices $T_0(s)$ in some arbitrarily chosen order. When no packets are queued at node s at time t, the task matrix $T_t(s)$ is by convention defined to be a special matrix denoted Z. A task is said to be completed at time t if $T_t(s)=Z$ for all s. The smallest t for which the task is completed under a given routing algorithm is called the *completion time* of the algorithm.

A communication task can equivalently be defined in terms of its initial task matrices $T_0(s)$, $s=0,\dots,N-1$. The task is isotropic if and only if the task matrices $T_0(s)$ are the same for all nodes s. In what follows, whenever there is no reason to distinguish among the nodes, we simply denote the task matrix at time t with T_t. When such a notation is used, we implicitly mean that $T_t(s)=T_t$, for all s. The initial task matrix for the total exchange problem is illustrated in Fig. 1.

The following lower bound for the completion time of any communication task (isotropic or non-isotropic) was proved in [15].

Theorem 1. *Let $\mathcal{T}$ be the completion time of any algorithm that executes a task with initial task matrices $T_0(s)$, $s=0, 1,\dots,N-1$. Let also $r_i(s)$ [or $c_i(s)$] denote the sum of the elements of the ith row (or column, respectively) of the task matrix $T_0(s)$. Then the following inequality holds*

$$\mathcal{T} \geq \max_{i,j}\left(\frac{1}{N}\sum_{s=0}^{N-1} c_j(s), \max_s r_i(s)\right), \tag{1}$$

where the outer maximization is carried out over all rows i and columns j.

For an isotropic task the preceding lower bound can be stated more succinctly by using the following definition.

Definition 3. The *critical sum* h of a matrix is equal to $\max_{i,j}(r_i, c_j)$, where r_i is the sum of the entries of row i, c_j is the sum of entries of column j, and the maximization is performed over all rows i and columns j. A row or column with sum of entries equal to h is called a *critical line*.

For an isotropic task, we have $T_0(s) = T_0$, $c_j(s) = c_j$, $r_i(s) = r_i$ for all nodes $s = 0, 1, \ldots, N-1$, so Theorem 1 yields $\mathcal{T} \geq \max_{i,j}(c_j, r_i) = h$, for any algorithm that executes the task. We state this fact as a corollary.

Corollary 1. *Let an isotropic communication task have initial task matrix T_0 and h be the critical sum of T_0. Then a lower bound for the time $\mathcal{T}$ required to complete the task is h.*

Isotropic tasks can be executed efficiently by a class of routing algorithms that satisfy a certain symmetry condition.

Definition 4. Given a task matrix $T_t(s)$ for each node s at time t, a *switching scheme* with respect to $T_t(s)$ is a collection of matrices $\{S_t(s) \mid s = 0, \ldots, N-1\}$ with entries 0 or 1. The matrix $S_t(s)$ has the same dimensions as $T_t(s)$, satisfies $S_t(s) \leq T_t(s)$ [i.e., if an entry of $T_t(s)$ is a zero, the corresponding entry of $S_t(s)$ must also be zero], and has at most one nonzero entry in each row or column. The switching scheme is called *symmetric* if for every t the matrices $S_t(s)$ are independent of s, that is, if for some matrix S_t we have $S_t(s) = S_t$ for all s.

Given a time $t \geq 0$ and a task matrix $T_t(s)$ for each node s, a switching scheme $\{S_t(s) \mid s = 0, \ldots, N-1\}$ with respect to $T_t(s)$ defines the packet (if any) that will be transmitted on each link at the time period beginning at time t. In particular, if the (i, j)th element of $S_t(s)$ is a one, the packet corresponding to the ith row of $T_t(s)$ will be transmitted on the jth link of node s. The requirement that each column of $S_t(s)$ contains at most one nonzero entry guarantees that at most one packet is scheduled for transmission on each link.

The task matrices at a given time period together with a corresponding switching scheme, define the task matrices for the next time period. Given a communication task defined by the task matrices $T_0(s)$, $s = 0, \ldots, N-1$, a routing algorithm can be defined as a sequence $\{S_0(s), S_1(s), \ldots\}$, such that $S_0(s)$ is a switching scheme with respect to the task matrix $T_0(s)$, $S_1(s)$ is a switching scheme with respect to the task matrix $T_1(s)$ [which is defined by $T_0(s)$ and $S_0(s)$], and, recursively, $S_{t+1}(s)$ is a switching scheme with respect to the task matrix $T_{t+1}(s)$ [which is defined by $T_t(s)$ and $S_t(s)$].

The following theorem, proved in [15], shows that if at some time t, the task matrices are the same for all nodes s, and a symmetric switching scheme with respect to $T_t(s)$ is used, then the next task matrices $T_{t+1}(s)$ will be the same for all nodes. As a result, for an isotropic task, one may use a routing algorithm defined by a sequence of symmetric switching schemes. Such a routine algorithm will be called *symmetric*. Its action is specified at a single node and is essentially replicated at all the other nodes; this is a very desirable property for implementation purposes.

Theorem 2. *Assume that for a given routing algorithm, at some time t we have a set of non-zero task matrices $T_t(s)$, which are the same for all nodes s. Then if S_t, a symmetric switching scheme with respect to $T_t(s)$ is used by the algorithm at time t, the task matrices $T_{t+1}(s)$ will be the same for all s. In particular, we have*

$$T_t(s) = T_t, \text{ for all } s \quad \Rightarrow \quad T_{t+1}(s) = T_{t+1}, \text{ for all } s, \tag{2}$$

where T_{t+1} is a task matrix consisting of the nonzero rows of the matrix $T_t - S_t$, except if $T_t = S_t$ in which case T_{t+1} is equal to the special matrix Z and the algorithm terminates.

From Theorem 2 we see that if the communication task is isotropic with initial task matrix T_0, we can specify a symmetric routing algorithm by a sequence of symmetric switching schemes S_0, $S_1, \ldots$ as follows:

Symmetric routing algorithm specification
The initial task matrix T_0 of the isotropic task is given. For $t = 0, 1, \ldots,$ given the task matrix T_t, S_t must be a symmetric switching scheme with respect to T_t; the task matrix T_{t+1} is then specified by the nonzero rows of the matrix $T_t - S_t$, unless $T_t = S_t$ in which case the algorithm terminates.

We see therefore that a symmetric routing algorithm that terminates after $k + 1$ time periods amounts to a decomposition of the initial task matrix T_0 into a sum

$$T_0 = \bar{S}_0 + \bar{S}_1 + \cdots + \bar{S}_k,$$

where each $\bar{S}_i$, $i = 0, \ldots, k$, is a binary nonzero matrix with the same dimension as T_0, and with at most one nonzero element in each column or row. The corresponding switching schemes S_i, $i = 0, \ldots, k$, consist of the nonzero rows of the matrices $\bar{S}_i$, $i = 0, \ldots, k$, respectively.

Thus, by restricting attention to symmetric routings, our original problem of finding optimal routings for isotropic communication tasks has been reduced to the simpler problem of 'clearing' the T_0 matrix (i.e. making all its entries equal to 0) in a mininum number of steps. At each step we are allowed to make 0 up to d entries, provided that these entries do not belong to the same row or column. The entries should not belong to the same row because at each step a packet cannot be transmitted on more than one link. The entries should not belong to the same column so that no two packets will use the same outgoing link. For any matrix, we use the term *line* to refer to a row or a column of a matrix.

Definition 5. A *permutation matrix* is any matrix with entries equal to 0 or 1 with the property that each line of the matrix has at most one nonzero entry.

It can be noted that the nonzero entries of a permutation matrix form an independent set of entries in the sense that no two of them belong to the same line. As a result, a set of entries of the task matrix which form a permutation submatrix can be cleared during the same step. In particular a permutation matrix S can be used as a switching scheme for any node at any time as long as the task matrix at that node and time satisfies $S \leq T$ (see Definition 4). The following theorem, proved in [15] extends slightly an old result by Hall (see [10], and [1] p. 120).

Theorem 3. *A nonnegative integer matrix with critical sum h can be written as the sum of h permutation matrices.*

The following is the main result concerning isotropic tasks.

Theorem 4 [15]. *The optimal completion time for an isotropic communication task is equal to the critical sum h of its task matrix.*

To prove Theorem 4, we use Theorem 3 to write the initial task matrix T_0 as the sum $\Sigma_{k=1}^{h}\bar{S}_k$ of permutation matrices $\bar{S}_1, \bar{S}_2, \dots, \bar{S}_h$. We then consider the symmetric switching scheme $\{S_k\}$, where for $k = 1, \dots, h$, S_k is obtained from $\bar{S}_k$ by removing the zero rows. Then the task matrix at times t with $1 \leq t < h$ consists of the nonzero rows of $T_0 - \Sigma_{k=1}^{t}\bar{S}_k$, and at time $t = h$ is equal to Z. Hence the communication task is completed after h steps. Since, by Theorem 1, h is also an upper bound, the corresponding symmetric routing must be optimal. This proof suggests the following algorithmic rule.

Optimal Completion Time Rule (abbreviated OCTR):
At each step an entry is cleared from each critical line of the task matrix.

As an example of application of the preceding results, consider finding optimal algorithms for the total exchange task. We have initially $N - 1$ packets queued at each node, one destined for each other node of the hypercube. Each column of the initial task matrix T_0 has $N/2$ ones. Therefore, the critical sum and also the optimal completion time is $N/2$. Any algorithm that works according to the OCTR is optimal as far as completion time is concerned, and there are at least $(N/2)!$ such algorithms.

3. Transposition of banded matrices

Beginning with this section we focus on the problem of transposing a $2^{\beta+1}+1$-diagonal matrix of size $N \times N$ stored by columns (or rows) in a hypercube of $N = 2^d$ processors. We first propose an assignment of the columns of the matrix to the hypercube nodes that makes the transposition a 'nearly isotropic' task, and then present algorithms to execute the task.

A $2^{\beta+1}+1$-diagonal matrix, where β is a nonnegative integer, is a matrix with

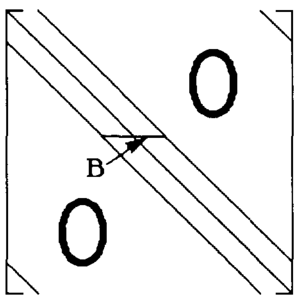

Fig. 2. A B-diagonal matrix.

entries a_{ij}, $i, j = 0, 1, \ldots, N-1$, such that $a_{ij} = 0$ whenever $2^\beta < |i-j| < N - 2^\beta$ (see Fig. 2). The integer

$$B = 2^{\beta+1} + 1$$

is called the *bandwidth* of the matrix. We first assume that the matrix is stored in a hypercube of $N = 2^d$ nodes, so that each processor stores a column of the matrix, and we subsequently generalize to the case where each processor stores multiple columns. We are free to choose the way in which the columns are assigned to the processors, and we are interested in assignments that result in fast transposition.

Transposing a fully-dense matrix stored by columns in a network of processors is equivalent to the total exchange task, since each processor i has to send the entry a_{ji} to every other processor j. The total exchange task requires time $N/2$ as discussed in Section 2. For a banded matrix, however, the zero entries do not have to be communicated, and the communication requirements are considerably less. As a result, it is possible to develop banded matrix transposition algorithms that are much faster than total exchange algorithms.

3.1. Column-to-processor assignment and transposition task matrix

The communication pattern that arises during the transposition of a banded matrix depends on the way the columns are assigned to processors. If we use the natural assignment, and store column i at processor i, the resulting communication pattern does not seem to have a favorable structure for fast execution. In this subsection we propose an assignment that makes the transposition task 'nearly isotropic', and in the next subsection we will use this assignment to derive efficient transposition algorithms.

We will use a kind of codes, called *Gray codes*, which are well known in coding theory. A Gray code of length k is a sequence of 2^k distinct binary numbers of k bits each, with the property that successive numbers in the sequence differ in exactly one bit. Furthermore, the first and the last number in the sequence also differ in exactly one bit. An example is the *reflected Gray code* discussed in ([2], p. 50).

Let $p(c)$ be the processor where column c is stored, and let c_1 (or c_2) represent the β most significant (or $d-\beta$ least significant, respectively) bits of c, that is, $c=c_2c_1$. We fix a Gray code sequence of length $d-\beta$. We consider the assignment that stores column c at the processor with binary representation

$$p(c)=c_1G_{c_2},$$

where G_{c_2} denotes the c_2^{th} number in the Gray code sequence. We call this the *Binary-Gray assignment*.

We denote by 0^β (or 1^β) the binary string of length β whose entries are all 0 (or 1, respectively). To transpose the matrix, processor $p(i)$ has to send a different packet to each processor $p(j)$ such that $|i-j|\le 2^\beta$ or $|i-j|\ge N-2^\beta$. Thus, processor $c_1G_{c_2}$ has to send a different packet to each one of the $2^{\beta+1}+1$ processors of the following set:

$$\begin{aligned}\mathscr{N}(c_1G_{c_2})=\Big\{&c_1G_{c_2\dot{-}1},(c_1\ddot{+}1)G_{c_2\dot{-}1},\dots,1^\beta G_{c_2\dot{-}1},\\&0^\beta G_{c_2},\dots,c_1G_{c_2},\dots,1^\beta G_{c_2},\\&0^\beta G_{c_2\dot{+}1},\dots(c_1\ddot{-}1)G_{c_2\dot{+}1},c_1G_{c_2\dot{+}1}\Big\}.\end{aligned}\tag{3}$$

The operation $\dot{+}$ (or $\dot{-}$) refers to modulo $2^{d-\beta}$ addition (respectively, subtraction), while the operation $\ddot{+}$ (or $\ddot{-}$) refers to modulo 2^β addition (respectively, subtraction).

Let $T(s)$ be the initial task matrix of node s that corresponds to the transposition task under the Binary-Gray assignment. The rows (routing tags) of $T(c_1G_{c_2})$ are obtained by forming the bitwise exclusive OR operation between node $c_1G_{c_2}$ and the nodes in the set $\mathscr{N}(c_1G_{c_2})$ of Eq. (3). Ordering appropriately the routing tags, the task matrix $T(c_1G_{c_2})$ can be written in the form shown in Fig. 3. By

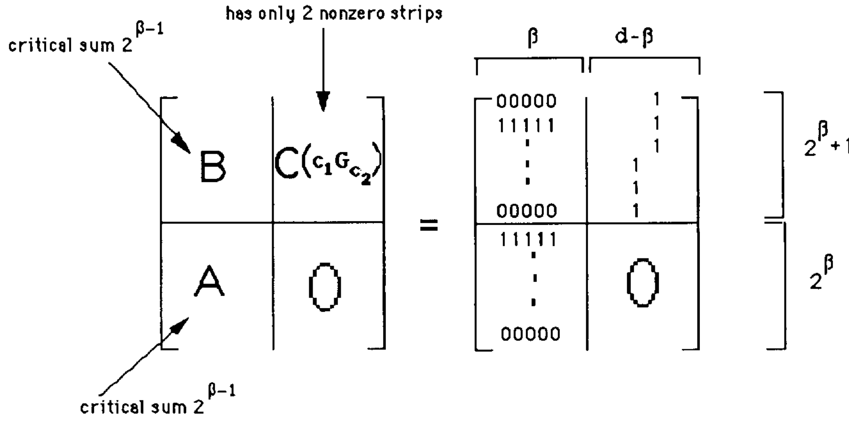

Fig. 3. The initial task matrix $T(c_1G_{c_2})$ of node $c_1G_{c_2}$.

convention, we let the exclusive OR operations between node $c_1G_{c_2}$ and the nodes $0^\beta G_{c_2}, 0^{\beta-1}1G_{c_2}, \ldots, 1^\beta G_{c_2}$ of $\mathcal{N}(c_1G_{c_2})$ form the lower half part

$$\left[A(c_1G_{c_2})O_{2^\beta\times(d-\beta)}\right]$$

of the task matrix $T(c_1G_{c_2})$, where $O_{2^\beta\times(d-\beta)}$ represents the all zero $2^\beta\times(d-\beta)$ matrix. It can be seen that the submatrices $A(c_1G_{c_2})$ are the same for all nodes $c_1G_{c_2}$; for brevity, we refer to them as submatrices A. The bitwise exclusive OR operations between $c_1G_{c_2}$ and the nodes in $\{c_1G_{c_2\dot{-}1}, (c_1\dot{+}1)G_{c_2\dot{-}1}, \ldots, 1^\beta G_{c_2\dot{-}1}\} \cup \{0^\beta G_{c_2\dot{+}1}, \ldots, (c_1\dot{-}1)G_{c_2\dot{+}1}, c_1G_{c_2\dot{+}1}\}$ form the upper half part

$$\left[B(c_1G_{c_2})C(c_1G_{c_2})\right]$$

of $T(c_1G_{c_2})$. Again it can be seen that the submatrices $B(c_1G_{c_2})$ are the same for all nodes $c_1G_{c_2}$; we refer to them as submatrices B.

Both A and B have as rows all the binary strings of length β. However, the upper right submatrix $C(c_1G_{c_2})$ of $T(c_1G_{c_2})$ is different for each node. Since G_{c_2} differs from $G_{c_2\dot{-}1}$ and $G_{c_2\dot{+}1}$ in a single bit, submatrix $C(c_1G_{c_2})$ has only one nonzero element per row, and only two nonzero columns. The nonzero columns appear at different positions for each node.

Submatrices A and B will be referred to as the *isotropic part* of the transposition task matrix. If $C(c_1G_{c_2})$ were zero, the task matrix $T(c_1G_{c_2})$ would be identical for all nodes, and the Binary-Gray assignment would have made the transposition task isotropic. Because $C(c_1G_{c_2})$ has a few nonzero elements, our column-to-processor assignment has made the transposition task *nearly isotropic*, in the sense the task matrices of different nodes are slightly different. For nearly isotropic task, it is natural to try to handle the isotropic part of the task by using the results of Section 2, and handle the remaining part in other ways. This is the approach followed here.

3.2. The transposition algorithm

In this subsection we give an algorithm to transpose a banded matrix stored in a hypercube according to the Binary-Gray assignment. In Section 4 we will show that the algorithm is of optimal order for a broad range of matrix bandwidths, where optimality is considered over al possible assignments.

If the task matrices were identical for all nodes, a symmetric routing scheme could execute the task. The next lemma indicates a way to make the task matrices identical.

Lemma 1. *If each packet that corresponds to a nonzero entry of the submatrix* $C(c_1G_{c_2})$ *where* $c_1 \in \{0, 1\}^\beta$ *and* $c_2 \in \{0, 1\}^{d-\beta}$, *is transmitted over the link corresponding to the nonzero entry, then the task matrices become identical for all nodes.*

Proof. First, note that the lemma does *not* follow from any of the results for isotropic tasks of Section 2 (for example, from Theorem 2), because neither the task matrices, nor the switching assignments corresponding to the transmissions mentioned in the lemma are the same for all nodes.

To prove the lemma note that, for each $t \in \{0, 1\}^\beta$, node $c_1G_{c_2}$ receives a packet with routing tag t either from node $c_1G_{c_2 \dotplus 1}$ or from node $c_1G_{c_2 \dot{-} 1}$. Since all the nodes receive a packet with tag t, for all $t \in \{0, 1\}^\beta$, and transmit all the packets which have different routing tags, the task matrices become identical. □

The clearance of the submatrices $C(c_1G_{c_2})$ involves packet transmissions on links of dimensions $0, 1, \ldots, d-\beta-1$. On the other hand, clearing submatrix A requires the use of dimensions $d-\beta$, $d-\beta+1, \ldots, d-1$ only. Therefore, packet transmissions associated with entries of A can take place simultaneously with packet transmissions associated with entries of $C(c_1G_{c_2})$. The links used to clear the submatrices $C(c_1G_{c_2})$ are of the form $(c_1G_{c_2}, c_1G_{c_2 \dot{-} 1})$ and $(c_1G_{c_2 \dot{-} 1}, c_1G_{c_2})$. For a given c_1 and varying c_2 these links belong to the ring $c_1G_0, c_1G_1, \ldots, c_1G_{2^{d-\beta}}$ (see [2], pp. 50–52). The clearance of $C(c_1G_{c_2})$ involves communication among nearest neighbors on these rings. For $c_1 \neq \hat{c}_1$ the rings corresponding to c_1 and $\hat{c}_1$ are disjoint.

Since each processor has to send a total of $2^\beta + 1$ packets to its neighbors on the ring (not all of them to the same neighbor), the submatrices $C(c_1G_{c_2})$ can be cleared in 2^β steps. Concurrently with $C(c_1G_{c_2})$, submatrix A can also start getting cleared by employing a symmetric routing scheme as described in Section 2. When clearing $C(c_1G_{c_2})$ we insist on the following rule: the entries of $C(c_1G_{c_2})$ that are cleared at each step are those that correspond to routing tags with the largest number of ones (ties are resolved in the same way for all nodes). In this way isotropic work is created at the largest possible rate to keep the links of dimensions $d-\beta$, $d-\beta+1, \ldots, d-1$ busy.

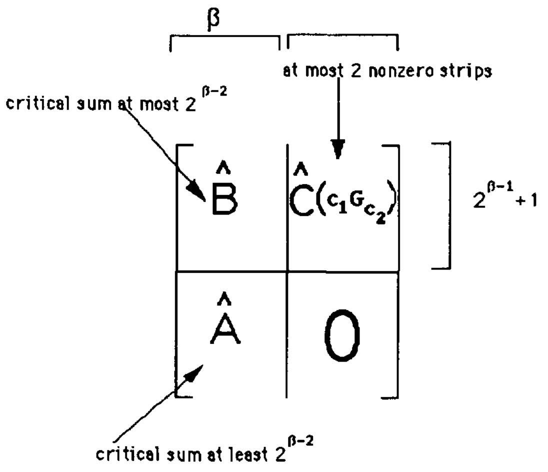

Fig. 4. The task matrix $\hat{T}(c_1G_{c_2})$ of node $c_1G_{c_2}$ after step $2^{\beta-1}$.

$$\begin{array}{c|c} \beta & d-\beta \\ \left(\begin{array}{c|c} 00000 & \quad 1 \\ 00000 & 1 \quad \\ \hline 10000 & \\ 01000 & \\ 00100 & O \\ 00010 & \\ 00001 & \end{array}\right) & \end{array}$$

Fig. 5. The task matrix of node $c_1G_{c_2}$ at time $2^{\beta}-1$.

Since the critical sum of A is $2^{\beta-1}$, the new task matrix $\hat{T}(c_1G_{c_2})$ of node $c_1G_{c_2}$ after $2^{\beta-1}$ steps will be of the form illustrated in Fig. 4. By that step, submatrix A will have been cleared, and a new submatrix $\hat{A}$ will have been formed in its position. The rows of the lower part $[\hat{A}O]$ of the new task matrix will be former rows of $[BC(c_1G_{c_2})]$ whose non-isotropic part has been cleared. The rows of the upper part $[\hat{B}\hat{C}_{c_1G_{c_2}}]$ will be former rows of $[BC(c_1G_{c_2})]$ whose non-isotropic part has *not* been cleared yet. Submatrix $\hat{C}(c_1G_{c_2})$ will have at most $2^{\beta-1}$ ones, while the critical sum of $\hat{A}$ will be greater than or equal to $2^{\beta-2}$, and the critical sum of $[\hat{B}\hat{A}]'$ will be equal to $2^{\beta-1}$. Following this procedure for a total $2^{\beta}-1$ steps, the task matrices take the form illustrated in Fig. 5. One additional step is then required to finish the task. Therefore, we have shown the following theorem:

Theorem 5. *The time required to transpose a $2^{\beta+1}+1$-diagonal matrix of size $N \times N$ stored according to the Binary-Gray assignment in an N-processor hypercube, is equal to 2^{β} steps.*

A B-diagonal matrix with $2^{\beta}+1 < B < 2^{\beta+1}+1$ can be treated as a $2^{\beta+1}+1$-diagonal matrix (with dummy packets for the zero entries within the band if necessary). This leads to the following corollary to Theorem 5.

Corollary 2. *A B-diagonal matrix of size $N \times N$ stored according to the Binary-Gray assignment* in an N-processor hypercube, can be transposed in at most $B-2$ steps.

In applications where the number of columns exceeds the number of available processors, it is necessary to store multiple columns in each processor. Consider the case where a B-diagonal $N \times N$ matrix is stored in a hypercube of P processors ($P < N$) so that each processor has either

$$q_M = \lceil N/P \rceil, \quad \text{or } q_m = \lfloor N/P \rfloor$$

contiguous columns. We denote

$$\gamma = \left\lceil \frac{B-1}{2q_m} \right\rceil.$$

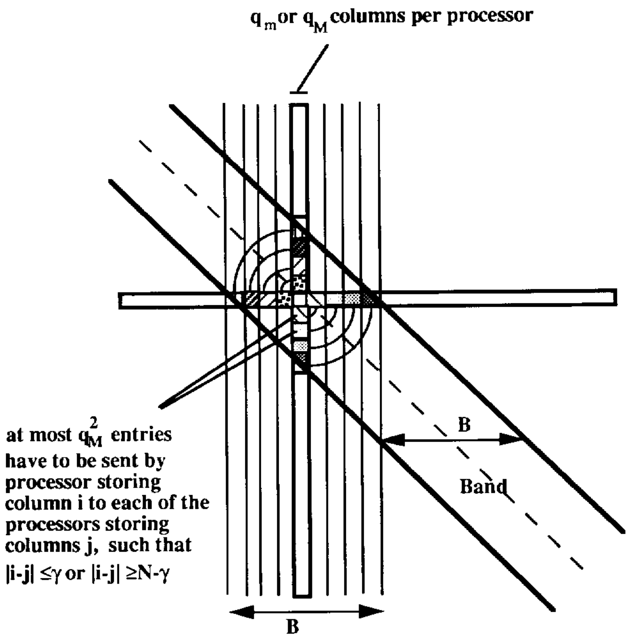
q_m or q_M columns per processor
at most q_M^2 entries have to be sent by processor storing column i to each of the processors storing columns j, such that |i-j| ≤γ or |i-j| ≥N-γ
B
Band
B

Fig. 6. Transposition of a banded matrix when $N > P$ and each processor stores the same number of columns plus or minus one. Then the matrix can be considered as a $2\gamma+1$-diagonal matrix of size $P \times P$. Each column (or row) of the $2\gamma+1$-diagonal matrix corresponds to q_m or q_M columns (or rows, respectively) of the original matrix, and each entry corresponds to (at most) q_M^2 entries of the original matrix.

To transpose the matrix (Fig. 6), each processor $p(i)$ has to send a set at most q_M^2 matrix entries to each of the processors $p(j)$ such that

$$|i-j| \le \gamma, \quad \text{or } |i-j| \ge N-\gamma.$$

Viewing the set of (at most) q_M^2 entries as a single communication, this is equivalent to transposing a $(2\gamma+1)$-diagonal matrix in a P-processor hypercube, and our earlier algorithms and results apply, with only slight loss of efficiency. To derive the corresponding running times, we distinguish two cases:

(a) The (at most) q_M^2 entries that have to be sent from node $p(i)$ to node $p(j)$ can be sent as a single packet requiring one unit of time to be transmitted over a link. (This corresponds to the communication time being dominated by overhead that is independent of the packet length.)

(b) Each entry of the matrix has to be sent as a separate packet with each packet requiring one unit of time to be transmitted over a link. (This also corresponds to the case where a set of q_M^2 entries is sent as a single packet, but the communication time is dominated by the transmission time over the link, which is proportional to packet length.)

By applying Theorem 5 and Corollary 2 to the equivalent $(2\gamma+1)$-diagonal matrix transposition problem stated above, we have:

Corollary 3. *Assuming the communication model of case (a) [or case (b)] above, a B-diagonal matrix of size $N \times N$ can be stored in a P-processor hypercube, so that it can be transposed in at most $2\gamma - 1$ time units [or $q_M^2(2\gamma - 1)$ time units, respectively].*

For other communication models (e.g. a model where q_M^2 entries are sent in a single packet with the transmission time of the packet being a constant plus a term which is linear in q_M^2), the estimates of the preceding corollary can be appropriately modified. Note that if γ is a power of 2, a slight modification of the preceding arguments shows that the transposition task can be executed even faster [in γ or $q_M^2\gamma$ time units, for the communication model of case (a) or case (b), respectively].

In the remainder of the paper we will assume that the size N of the matrix is equal to the number of processors, so that each processor stores one column of the matrix. In the next section we derive lower bounds on the time required to transpose a banded matrix in hypercube network of processors.

4. Lower bounds on the minimum time to transpose a banded matrix

The transposition algorithm presented in the preceding section is optimal for the Binary-Gray assignment of columns to processors, as can be seen from Theorem 1, which holds for both isotropic and non-isotropic tasks. A question that arises is whether there exists another column-to-processor assignment that results in a faster transposition algorithm. In Subsection 4.1 we will prove that this is not possible if we limit ourselves to the class of affine assignments. In Subsection 4.2 we will derive a lower bound on the time required to transpose a banded matrix under any possible assignment of columns to hypercubes, and we will show that the completion time of our algorithm is within a constant factor from this lower bound for large range of bandwidths.

4.1. A lower bound for affine column-to-processor assignments

The most usual way to store a matrix in a hypercube is the *binary assignment*, where column c is stored in processor c. With this assignment, it is not possible to transpose a matrix of bandwidth B in less than $(B-1)/2$ steps [note that B is always an odd number since the band is symmetric around the diagonal]. To see that, note that the processor which stores column c, has to send a packet to the processors which store column $c+x$ mod N, for all $(B-1)/2 \leq x \leq (B-1)/2$. At least $(B-1)/2$ of the routing tags of these packets have their least significant bit equal to one. Thus, it follows from Theorem 1 that the transposition task cannot be executed in less than $(B-1)/2$ steps. In fact, if $(B-1)/2$ is odd, it can be shown that the transposition task cannot be executed in less than $(B+1)/2$ steps. The limiting factor in the transposition of a matrix stored according to the binary assignment are the hypercube links of dimension 0.

We now consider a more general class of column-to-processor assignments, which we call *affine assignments*. Affine assignments (transformations) have been previously considered in a different context in [5].

Definition 6. In an affine column-to-processor assignment column c (viewed as a binary vector) is stored at processor

$$p(c) = Ac \oplus v,$$

for some nonsingular binary matrix A of dimension $d \times x$, and some binary vector v of dimension d.

The matrix-vector multiplication in the above definition is performed modulo 2. If A is singular, then the assignment obtained is not one-to-one. As proved in [5], the Gray coding, bit reversal, shuffle, and dimension permutation transformations as well as all the combinations of these result in affine assignments (for $v = 0$ and appropriate choices of A). The binary assignment is also an affine assignment (with A being the identity matrix). The Binary-Gray assignment used in Section 3 is also a particular case of an affine assignment, which is obtained by letting $v = 0$ and

$$A = \begin{pmatrix} O & I_{\beta \times \beta} \\ J_{(d-\beta)\times(d-\beta)} & O \end{pmatrix},$$

where $I_{\beta \times \beta}$ is the identity matrix of dimension $\beta \times \beta$, and $J_{(d-\beta)\times(d-\beta)}$ is a $(d-\beta)\times(d-\beta)$ matrix whose main and lower diagonals have all entries equal to one and all other entries equal to zero.

The following theorem gives a lower bound on the time required to transpose a banded matrix.

Theorem 6. *The transposition of a B-diagonal matrix stored by columns in a hypercube networks of processors using an affine column-to-processor assignment requires at least*

$$\frac{B-1}{2}$$

steps.

Proof. Let $p(c) = Ac \oplus v$ be the processor that stores column c, and let $b = (B-1)/2$. In order to transpose the matrix, each processor $p(c)$ has to send a different packet to processors $p(c+x)$ for all $-b \le x \le b$ (the addition $c+x$ corresponds to modulo N addition). Let t_x be the routing tag of the packet sent from processor $p(c)$ to processors $p(c+x)$, that is

$$t_x = Ac \oplus A(c+x) = A(c \oplus (c+x)).$$

Since A is nonsingular, its rightmost (0th) column has at least one entry equal to one, say the jth entry. Let $c_x = c \oplus (c+x)$, $-b \le x \le b$; then $t_x = Ac_x$. Among the

c_x's there are b disjoint pairs $(c_{x_1}, c_{x'_1}), (c_{x_2}, c_{x'_2}), \ldots, (c_{x_b}, c_{x'_b})$ such that c_{x_i} differs from $c_{x'_i}$ only in the least significant bit. Since the jth entry of the last column of A is equal to one, t_{x_i} and $t_{x'_i}$ will differ in the jth bit. Thus, at least b of the routing tags of the packets that have to be sent by processor $p(c)$ have their jth bit equal to one. Therefore, there is a total of at least $Nb = N(B-1)/2$ routing tags of packets involved in the transposition task that have a unit at the j^{th} position. Using Theorem 1, we conclude that the transposition task will require at least $(B-1)/2$ steps to execute. The limiting factor in the execution of the transposition task is that too many packets have to cross links of the jth hypercube dimension. □

Combining Corollary 3, and Theorems 5 and 6 we obtain the following.

Corollary 4. *The algorithm proposed (in Section 3 for the Binary-Gray assignment) executes the transposition task in time which is within a factor of two of the optimal execution time for any matrix bandwidth B, where optimality is considered over all affine column-to-processor assignments, and over al communication algorithms for a particular assignment. When $B = 2^{\beta+1} + 1$ for some β, the algorithm is strictly optimal over all affine assignments.*

Affine assignments form a large class of column-to-processor assignments, including the ones most often used in practice. The near-optimality of the proposed algorithm within this class of assignments is a strong indication of its efficiency. In the following subsection, we consider optimality over *all* possible assignments of columns to processors.

4.2. A universal lower bound

For $\beta = 0$ (tridiagonal matrix) our algorithm executes in a single step, and is obviously optimal for any assignment of columns to processors. For $\beta = 1$ (five-diagonal matrix) the algorithm requires two steps, which is also optimal over all possible assignments. To see this, let $p(c)$ be the processor that stores column c, $c \in \{0, 1, \ldots N-1\}$. For the transposition to be executed in a single step, the nodes $p(c-1)$, $p(c+1)$ [as well as the nodes $p(c-2)$, and $p(c+2)$] have to be neighbors of $p(c)$. To transpose the five-diagonal matrix in a single step, processors $p(c-1)$ and $p(c+1)$ also have to be neighbors. This is clearly impossible for the hypercube topology, because two nodes that have a common neighbor have to be at a distance of two from each each other. For $\beta = d-1$ (full matrix) the algorithm requires $N/2$ steps, which is again optimal for any column-to-processor assignment. Our algorithm is also clearly of optimal order when $B = O(1)$, which is another practical case. It remains to examine the performance of the algorithm for values of the bandwidth that are between the two extremes. In this section, we will derive a universal lower bound on the minimum number of steps required to transpose a banded matrix, which holds for any possible assignment of matrix columns to hypercube nodes. When the bandwidth B is $\Theta(N^c)$ for some constant $c \in (0, 1]$, the universal bound will turn out to be of the same order of magnitude as the completion time of our algorithm. The case $c = 1/2$ is particularly important since it corresponds to banded matrices obtained by the discretization of partial differential equations (see [9]). We have not proved that our algorithm is of optimal order over all possible embeddings and for all bandwidths B. However, the optimality of the algorithm for the two most extreme cases, combined with the fact that it is of optimal order for a broad range of intermediate bandwidths are indications of the algorithm's efficiency.

In order to prove the lower bound on the completion time of any banded-matrix transposition algorithm let $p(c)$ represent the hypercube node which stores column c. Suppose that there is a way to assign columns to nodes so that for each column c, the $B-1$ columns j that satisfy $|c-j| \le 2^{\beta}$, or $|c-j| \ge N-2^{\beta}$ are stored at $B-1$ hypercube nodes which are closest to node $p(c)$. Such an assignment, although not always possible, would result in a minimum number of packet transmissions, and therefore it can provide a lower bound on the minimum number of steps required to transpose a banded matrix for any column-to-processor assignment.

Let r be the distance from node $p(c)$ to the farthest of its $B-1$ closest nodes. Then

$$\sum_{i=0}^{r-1} \binom{d}{i} \le B \le \sum_{i=1}^{r} \binom{d}{i}.$$

This relation gives

$$\frac{B}{N} \le \sum_{i=0}^{r} \binom{d}{i} \left(\tfrac{1}{2}\right)^{d} = \sum_{i=d-r}^{d} \binom{d}{i} \left(\tfrac{1}{2}\right)^{d}, \tag{4}$$

or

$$\frac{B}{N} \le \mathscr{B}(d-r, 1/2, d),$$

where $\mathscr{B}(m, p, K)$ is the probability that we get more than m successes in K independent Bernoulli trials with p being the probability of success for each trial. The Chernof bound gives (see e.g. [14]) that

$$\frac{B}{N} \le \left(\frac{d}{2(d-r)}\right)^{d-r} \left(\frac{d}{2r}\right)^{r}, \quad \text{for } r < d/2,$$

or

$$B \le \frac{d^d}{(d-r)^{d-r} r^r}, \quad \text{for } r < d/2.$$

Letting $r = \lambda d$, the previous relation is transformed to

$$B \le \frac{1}{(1-\lambda)^{d-r} \lambda^r} = \left(\frac{1}{(1-\lambda)^{1-\lambda} \lambda^{\lambda}}\right)^{d}, \quad \text{for } \lambda < 1/2,$$

which yields

$$\frac{\log_2 B}{d} \le H(\lambda),$$

with

$$H(\lambda) = -\lambda \log_2\lambda - (1-\lambda)\log_2(1-\lambda)$$

being the entropy (base 2) function (see Fig. 7). If we restrict $H(\lambda)$ to $\lambda < 1/2$, then H^{-1} is well defined and monotonically increasing. Therefore, we can write

$$r = \lambda d \ge \min\left(dH^{-1}\left(\frac{\log_2 B}{d}\right), d/2\right).$$

In the case where $B = N^c$ for some $c \in (0, 1]$, we get that

$$r \le \min(dH^{-1}(c), d/2) = \Theta(d), \quad c \in (0, 1]. \tag{5}$$

The following lemma gives a lower bound on the number of transmissions required to transpose a banded matrix.

Lemma 2. *Let W be the total number of transmissions required for the packets that are sent (or received) by the node $p(c)$ to arrive at their destinations. Then*

$$W = \Omega(Br), \quad \text{for } r > 1.$$

Proof. The mean number of transmissions required by the packets sent by processor $p(c)$ satisfies

$$\frac{W}{B} \ge \frac{\sum_{j=1}^{r-1} j\frac{d!}{j!(d-j)!}}{\sum_{j=0}^{r-1}\binom{d}{j}} = \frac{d\sum_{j=1}^{r-1}\binom{d-1}{j-1}}{\sum_{j=0}^{r-1}\binom{d}{j}}$$

$$= \frac{d\sum_{j=0}^{r-2}\binom{d-1}{j}}{\sum_{j=0}^{r-1}\binom{d}{j}} = \frac{d}{2}\frac{\sum_{j=0}^{r-2}\binom{d-1}{j}\left(\frac{1}{2}\right)^{d-1}}{\sum_{j=0}^{r-1}\binom{d}{j}\left(\frac{1}{2}\right)^{d}}.$$

Let $X^{(d)}$ be the sum of d independent Bernoulli random variables with mean 0.5, and let $X^{(d-1)}$ be the sum of the first $d-1$ of them. Then

$$\frac{W}{B} \ge \frac{d}{2}\frac{\Pr(X^{(d-1)} \le r-2)}{\Pr(X^{(d)} \le r-1)} \geqslant \frac{d}{2}\Pr(X^{(d-1)} \le r-2 \mid X^{(d)} \le r-1).$$

The conditional probability in the preceding equation is always greater than or equal to $(r-1)/d$. This is because if $X^{(d)} < r-1$ then $X^{(d-1)} \le r-2$ always, while if $X^{(d)} = r-1$ then $X^{(d-1)} \le r-2$ with probability $(r-1)/d$ [given that we

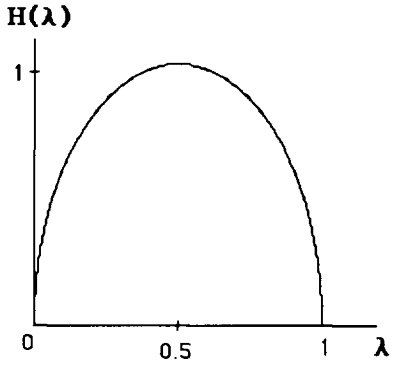

Fig. 7. The entropy function $H(\lambda)$.

had exactly $r-1$ successes in d independent trials, the probability that the last trial was a success is $(r-1)/d]$. Thus,

$$W \geq \frac{B(r-1)}{2} = \Omega(Br), \quad \text{for } r > 1. \quad \square$$

Since the packets sent by a node require a total of $\Omega(Br)$ transmissions, and each node has d links, a lower bound on the minimum time $\mathcal{T}$ required to transpose the banded matrix is

$$\mathcal{T} = \Omega\left(\frac{Br}{d}\right). \tag{6}$$

Eq. (6) holds when $r > 1$, or else $B > \log N$. Combining Eqs. (5) and (6) we obtain the following theorem.

Theorem 7. *When* $B = \Theta(N^c)$ *for some* $c \in (0, 1]$, *then*

$$\mathcal{T} = \Omega(B) = \Omega(N^c).$$

The algorithm of Section 3 executes the transposition task in $(B-1)/2 = O(B)$ steps. Theorem 7 shows that when $B = N^c$ the algorithm is of the optimal order over all possible assignments of columns to processors. In particular, the ratio of the lower bound to the complexity of the algorithm is roughly min(0.5, $H^{-1}(c)$). This, combined with the optimality of the algorithm in the cases $B = 1$ and $B = N$, suggest that the algorithm is efficient.

5. Several simultaneous banded matrix transpositions

In this section we present a way to store $\lfloor d/\beta \rfloor$ matrices each of bandwidth $B = 2^{\beta+1} + 1$ in a hypercube, and transpose them simultaneously in $B = 2^{\beta+1} + 1$

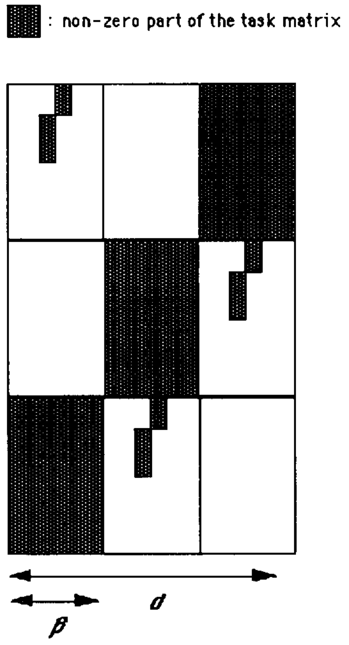

Fig. 8. A typical task matrix at a node for the case $d/\beta = 3$.

steps. This means that when $\lfloor d/\beta \rfloor > 2$ (or, equivalently $B < N^{1/2}$), performing more than one banded matrix transpositions simultaneously increases the link utilization and the efficiency of the algorithm. This does not contradict Theorem 7, since in the case $B = N^c$ we have $d/\beta \approx c^{-1}$, and the improvement in efficiency is a constant factor. If B is of strictly smaller order of magnitude [e.g. if B is $\Theta(1)$, or $\Theta(d)$] then $d/\beta \to \infty$ as $d \to \infty$, and the improvement is even more significant.

The main idea of the section can be summarized as follows. The transposition algorithm of Section 3 uses mainly β hypercube dimensions, say dimensions $d-1$, $d-2, \ldots, d-\beta$. Thus, a second banded matrix can be stored in the hypercube so that its transposition uses mainly the dimensions $d-\beta-1$, $d-\beta-2, \ldots, d-2\beta$, and this can be extended to a total of $\lfloor d/\beta \rfloor$ matrices. When doing so, it may no longer be possible to pipeline the isotropic with the non-isotropic part of the task. This results in an increase of the completion time roughly by a factor of two, which is offset by the improvement in efficiency if $\lfloor d/\beta \rfloor > 2$.

Let $L_j(\cdot)$ be the operator that when applied to a binary string, shifts it cyclically j positions to the left; for example $L_1(0110) = 1100$. Let also c_1, c_2, and G_{c_2} be defined as in Section 3. The matrices are stored in the hypercube in the following way. We assign column c of the jth matrix to the hypercube node $p_j(c)$ where

$$p_j(c) = L_{j\beta}(p(c)) = L_{j\beta}(c_1 G_{c_2}), \quad j = 0, 1, \ldots \left\lfloor \frac{d}{\beta} \right\rfloor - 1, \quad c = 0, 1, \ldots, N-1.$$

The transposition is viewed as a single task and it is done simultaneously for all matrices. A typical initial task matrix is shown in Fig. 8. The algorithm consists of two phases, which cannot in general overlap. In the first phase, the isotropic part of the task matrices is cleared; this requires 2^{β} steps. This is possible because the clearance of the isotropic part of the task matrix that corresponds to the jth matrix, $j = 0, 1, \ldots \lfloor d/\beta \rfloor - 1$, requires the use of dimensions $d - j\beta - 1$, $d - j\beta - 2, \ldots, d - (j+1)\beta$ of the hypercube, and does not interfere with the transposition of the other banded matrices. In the second phase, the non-isotropic part is cleared, which requires at most $2^{\beta} + 1$ additional steps. Thus, the total time $\mathscr{T}$ required to transpose $\lfloor d/\beta \rfloor$ banded matrices, each of bandwidth $B = 2^{\beta+1} + 1$, is

$$\mathscr{T} = 2^{\beta+1} + 1 = B, \quad \text{for } \left\lfloor \frac{d}{\beta} \right\rfloor \text{ concurrent banded matrix transpositions.}$$

The results can be easily extended to the case where we have banded matrices with different bandwidths $B_k = 2^{\beta_k} + 1$, $k = 0, 1, \ldots, p$, which satisfy the relation $\sum_{k=1}^{p} \beta_k \leq d$.

References

[1] D.P. Bertsekas, *Linear Network Optimization: Algorithms and Codes* (M.I.T. Press, Cambridge, MA, 1991).

[2] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[3] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng and J.N. Tsitsiklis, Optimal communication algorithms for hypercubes, *J. Parallel Distrib. Comput.* 11 (1991) 263–275.

[4] A. Edelman, Optimal matrix transposition and bit reversal on hypercubes: All-to-All personalized communication, *J. Parallel Distrib. Comput.* 11 (1991) 328–331.

[5] A. Edelman, S. Heller and S.L. Johnsson, Index transformation algorithms in a linear algebra framework, preprint.

[6] C.T. Ho and M.T. Raghunath, Efficient communication primitives on hypercubes, *Concurrency* 4 (Sep. 1992) 427–457.

[7] S.L. Johnsson and C.T. Ho, Algorithms for matrix transposition on Boolean N-cube configured ensemble architectures, *SIAM J. Matrix Analysis* 9 (3) (July 1988).

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

[9] O.A. McBryan and E.F. Van de Velde, Hypercube algorithms and their implementations, *SIAM J. Sci. Stat. Comput.* 8 (1987) 227–287.

[10] H.J. Ryser, *Combinatorial Mathematics* (Mathematical Association of America, Rahway, NJ, 1965).

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

[12] Y. Saad and M.H. Schultz, Data communication in hypercubes, *J. Parallel Distr. Comput.* (1989) 115–135.

[13] Y. Saad and M.H. Schultz, Data communication in parallel architectures, *Parallel Comput.* 11 (1989) 131–150.

[14] L.G. Valiant and G.J. Brebner, Universal schemes for parallel communication, in *Proc. 13th Annual Symp. on Theory of Computing* (1981) 263–277.

[15] E.A. Varvarigos and D.P. Bertsekas, Communication algorithms for isotropic tasks in hypercubes and wraparound meshes, *Parallel Comput*, 18 (1992) 1233–1257.

[16] E.A. Varvarigos, Optimal communication algorithms for multiprocessor computers, MS. Thesis, Report CICS-TH-192, Center of Intelligent Control Systems, MIT, 1990.