Algorithms for Matrix Transposition on Boolean n-cube Configured Ensemble Architectures

Ching-Tien Ho and S. Lennart Johnsson

YALEU/DCS/TR-577
April 1987

This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564. Approved for public release: distribution is unlimited.

† To appear in the Proceedings of International Conference on Parallel Processing, 1987.

Algorithms for Matrix Transposition on Boolean n-cube Configured Ensemble Architectures

Ching-Tien Ho and S. Lennart Johnsson
Departments of Computer Science
Yale University
New Haven, CT 06520

Abstract In a multiprocessor with distributed storage the data structures have a significant impact on the communication complexity. In this paper we present a few algorithms for performing matrix transposition on a Boolean n-cube. One algorithm performs the transpose in a time proportional to the lower bound both with respect to communication start-ups and element transfer times. We present algorithms for transposing a matrix embedded in the cube by a binary encoding, a ***binary-reflected*** Gray code encoding of rows and columns, or combinations of these two encodings. The transposition of a matrix when several matrix elements are identified to a node by ***consecutive*** or *cyclic* partitioning is also considered and lower bound algorithms given. Finally, experimental data are provided for the Intel iPSC and the Connection Machine.

1 Introduction

Matrix transposition is one of the basic operations frequently performed in linear algebra. It is useful in the solution of systems of linear equations by a variety of techniques. For instance, it may be beneficial with respect to performance for the solution of tridiagonal systems of equations on Boolean n-cube configured architectures to move all equations of a system to one processor and solve it locally instead of using a parallel algorithm such as cyclic reduction [9]. Tridiagonal systems occur in the Alternating Direction Implicit (ADI) method and in the solution of Poisson's problem by the Fourier Analysis Cyclic Reduction (FACR) method.

In this paper we focus on matrix transposition on Boolean n-cube architectures. The transpose can be formed recursively as described in [13,1,6,11]. Stone describes a mapping on to shuffle-exchange networks for the case with one matrix element per node. We consider the case with multiple matrix elements per node and focus on the pipelining of communication operations and the optimal use of the communication bandwidth of the Boolean n-cube. In [6,7] we described and analyzed the complexity of a transpose algorithm for a two-dimensional mesh and presented a few algorithms for the transposition of matrices embedded in the cube by binary or Gray code encoding of the row and column indices. In this paper we present a transpose algorithm that is of lower complexity in the case of concurrent communication on multiple ports, and present experimental data for the Intel iPSC and the Connection Machine [2].

We first introduce the notation and data structures used in this study, then present algorithms for the transpose operation for one-dimensional and two-dimensional partitionings. Implementation issues particular to the actual machines used, but important for the interpretation of the experimental results presented, are addressed after the description of the algorithms. A summary and conclusion follows.

2 Notation and Definition

In the case of an $N = 2^n$ processors Boolean n-cube and a $P \times Q$ matrix such that $P = 2^p$, $Q = 2^q$ and $p + q = n$, matrix elements can be assigned to distinct processors without any waste. One obvious assignment is to embed the matrix by encoding the row and column indices of matrix elements in binary code. Such an embedding does not preserve proximity. A ***binary-reflected Gray code*** [12] encoding of row and column indices preserves adjacency. This code is referred to as Gray code in the following and the encoding of i is $G(i)$. The conversion from one kind of encoding to the other can be accomplished in $\log N - 1$[1] routing steps and the paths made edge-disjoint [6].

In the case of $P \times Q > N$ multiple matrix elements must be assigned to the same node in the Boolean cube. For $N < \max(P, Q)$ there is a choice between one-dimensional partitioning (strip) and two-dimensional partitioning (block). For either kind of partitioning the matrix elements are assigned either ***cyclicly*** or ***consecutively*** [6,7]. In a one-dimensional ***cyclic*** partitioning column (or row) j is assigned to partition $j \bmod N$ and in a one-dimensional ***consecutive*** partitioning column j is assigned to partition $\lfloor \frac{j}{\lceil \frac{Q}{N} \rceil} \rfloor$ with partitions labeled $0, 1, \ldots, N-1$. In a cube with $N = 2^n$ nodes the n lowest order bits of the binary encoded column (row) index determines the partition to which a column (row) is assigned in the cyclic partitioning, and analogously, the n highest order bits determines the partition assignment in the consecutive partitioning, Figure 1. The partitions are assigned to cube nodes through encoding in binary or Gray code.

In the two-dimensional partitioning we let N_r denote the number of partitions in the row direction and N_c the number of partitions in the column direction. The total number of partitions is $N_r \times N_c = N$. In the cyclic partitioning matrix element (i, j) is assigned to partition $(i \bmod N_r, j \bmod N_c)$ and in the consecutive partitioning it is assigned to partition $(\lfloor \frac{i}{\lceil \frac{P}{N_r} \rceil} \rfloor, \lfloor \frac{j}{\lceil \frac{Q}{N_c} \rceil} \rfloor)$, Figure 2. For a $P \times Q$ matrix partitioned by the consecutive strategy the highest order $\log N_r$ bits of the matrix row index determines the partition row index. Analogously, the $\log N_c$ highest order bits of the matrix column index determines the partition column index. We assume for simplicity that $N_r = 2^r$ and $N_c = 2^c$. In cyclic storage, the last several bits of the matrix row and column indices determine the assignment to a partition. The partitions are assigned to nodes in the cube by encoding the row and column indices of a partition in binary code or Gray code.

For the architecture we assume that it has packet oriented communication with a communications overhead τ, a transmission time per element t_c, and a maximum packet size of B_m elements. This model applies for the Intel iPSC. For a bit-serial architecture, such as the Connection Machine, the overhead can be made additive by pipelining. With the operating system for the Intel iPSC on which our experiments were carried out $\tau \approx 5ms$, $t_c \approx 1\mu\text{sec/byte}$ and $B_m = 1k$ bytes. For the algorithm description and analysis we consider two cases with respect to communication capabilities: communication restricted to

[1]Throughout this paper $\log N = \log_2 N$

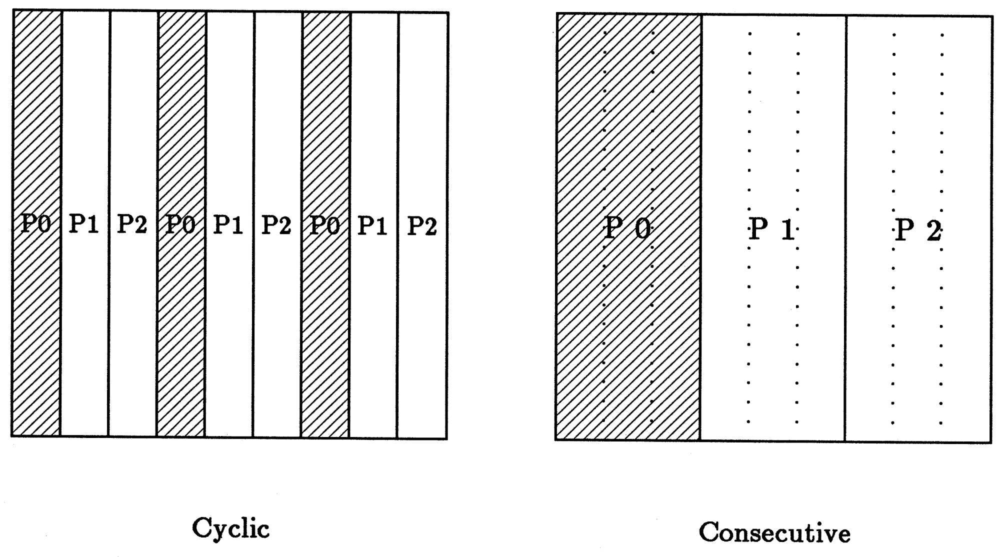


Figure 1: Cyclic and Consecutive one-dimensional partitioning.

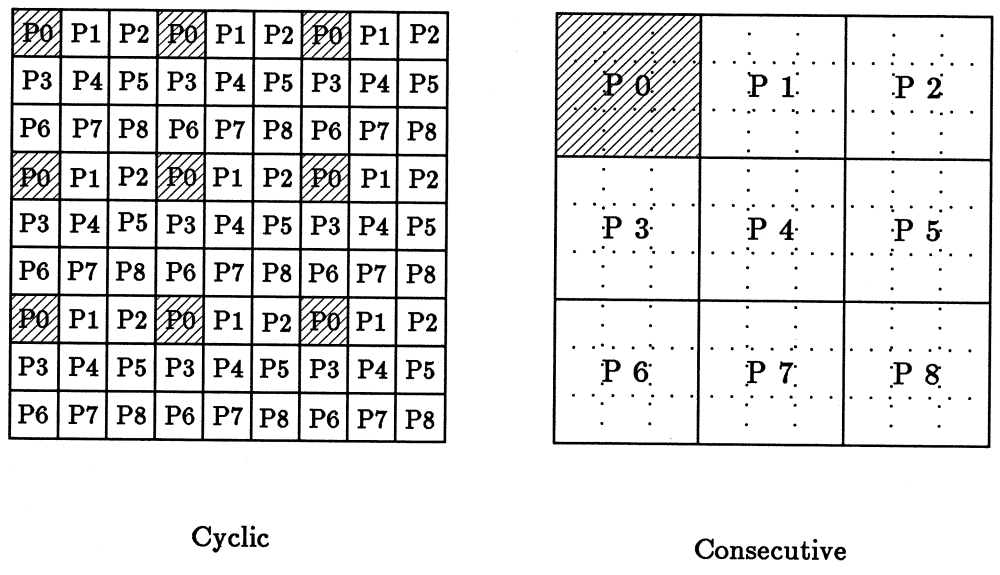

P0 P1 P2 P0 P1 P2 P0 P1 P2
P3 P4 P5 P3 P4 P5 P3 P4 P5
P6 P7 P8 P6 P7 P8 P6 P7 P8
P0 P1 P2 P0 P1 P2 P0 P1 P2
P3 P4 P5 P3 P4 P5 P3 P4 P5
P6 P7 P8 P6 P7 P8 P6 P7 P8
P0 P1 P2 P0 P1 P2 P0 P1 P2
P3 P4 P5 P3 P4 P5 P3 P4 P5
P6 P7 P8 P6 P7 P8 P6 P7 P8
P 0
P 1
P 2
P 3
P 4
P 5
P 6
P 7
P 8
Cyclic
Consecutive

Figure 2: Cyclic and Consecutive two-dimensional partitioning.

one port at a time, ***one-port*** communication, and concurrent communication on all ports, ***n-port*** communication. The former is a good approximation of the capabilities of the Intel iPSC.

We assume that $n = r + c$ and write the processor address as $(ra_{r-1}ra_{r-2}\ldots ra_0ca_{c-1}ca_{c-2}\ldots ca_0)$ or $(ra||ca)$, where $ra = (ra_{r-1}\ ra_{r-2}...ra_0)$, $ca = (ca_{c-1}ca_{c-2}...ca_0)$ and '$||$' is the concatenation operator of two binary numbers.

3 One-Dimensional Matrix Partitioning

3.1 Consecutive and Cyclic Storage

The transposition of a matrix from consecutive row to consecutive column partitioning has the same communication pattern as transposition from cyclic row to cyclic column partitioning. If $P, Q \geq N$ each node needs to send data to all other nodes. For $P < N$ or $Q < N$ a subset of the nodes are either recipients or senders of data. Conversion between consecutive row and cyclic row (or column) partitioning also implies that all nodes send unique information to all other nodes, if $Q \geq N^2$ for column partitioning, and $P \geq N^2$ for row partitioning. Consecutive partitioning is made on the $\log N$ highest order bits, cyclic partitioning on the $\log N$ lowest order bits of the row or column indices. Clearly, if less than $2\log N$ bits are required for the encoding of row or column indices data is not sent from each node to every other node in the conversion. The amount of data sent from any node to any other node is $\frac{PQ}{N}$ in either of the two matrix transpose operations, or the cyclic/consecutive partitioning conversion.

Proposition 1 *Conversion between any two of the following six embeddings*

1. *consecutive row storage.*
2. *consecutive column storage.*
3. *cyclic row storage.*
4. *cyclic column storage.*
5. *combination of consecutive row and cyclic row storage.*
6. *combination of consecutive column and cyclic column storage.*

is equivalent to all-to-all personalized communication [10], if $P \geq N^2$ for row partitioning, or $Q \geq N^2$ for column partitioning.

For the transpose of a matrix partitioned consecutively by columns the partition assignment is changed from the $\log N$ highest order bits of the column index encoding to the $\log N$ highest order bits of the row index encoding. For cyclic partitioning it is instead the lowest order $\log N$ bits of the row and column indices that are of interest. For conversions between cyclic and consecutive partitioning only one index (row or column) is involved. It follows that if either the maximum row or column index requires less than $\log N$ bits for its encoding (or both), then the matrix transpose operation corresponds to a few distinct one-to-all personalized communications[10].

3.2 Binary and Gray Code Embeddings

The six partitionings in proposition 1 can be used in connection with either Gray code encoding or binary code encoding. There is a total of 12 matrix embeddings obtainable through the combination of one-dimensional partitioning strategies and encodings in the Boolean cube; namely, consecutive or cyclic partitioning or a combination of the two, row or column partitioning, binary or Gray code encoding. Most conversions between any two of the 12 storage methods are equivalent in terms of the global communication. [2]

3.3 Generic Algorithms

If there are data elements for every processing node both before and after the data rearrangement, then the communication is ***all-to-all personalized communication***. In the case where only a few processing nodes contain data before or after the transformation it is of the form ***many-to-all*** or ***all-to-many personalized communication***. In the extreme case it is ***one-to-all personalized communication***. These forms of communication are studied in detail in [3,10].

3.3.1 One-to-All Personalized Communication

One-to-all personalized communication can be performed in lower bound time by routing according to a *Spanning Binomial Tree* (SBT) with ***one-port*** communication [3]. Before the communication the source node holds all PQ data elements. After the communication, every processor holds $\frac{PQ}{N}$ data elements. The communication time is $T_{min} = (1-\frac{1}{N})PQt_c + \tau(\lceil\frac{PQ}{B_m}\rceil + \min(n, log\lceil\frac{B_mN}{PQ}\rceil) - 1)$, which is minimized for $B_m \geq \frac{PQ}{2}$. $T_{opt} = (1-\frac{1}{N})PQt_c + n\tau$.

With ***n-port*** communication routing according to a SBT is no longer optimal. The lower bound for concurrent communication is $T = \frac{1}{n}(1-\frac{1}{N})PQt_c + n\tau$. One nearly optimal (truly optimal for n prime) routing strategy is to use a *Spanning Balanced n-Tree* (SBnT) [3,10,5]. The routing time is $T \approx \sum_{i=1}^{n}(\frac{1}{n}\binom{n}{i}\frac{PQ}{N}t_c + \lceil\frac{1}{n}\binom{n}{i}\frac{PQ}{B_mN}\rceil\tau) = T \approx \frac{1}{n}(1-\frac{1}{N})PQt_c + \sum_{i=1}^{n}(\lceil\frac{1}{n}\binom{n}{i}\frac{PQ}{B_mN}\rceil\tau$, which has a minimum of $T_{opt} = \frac{1}{n}(1-\frac{1}{N})PQt_c + n\tau$ for $B_m \geq \sqrt{\frac{2}{\pi}\frac{PQ}{n^{3/2}}}$. The speed-up of the transmission time of SBnT routing over SBT routing is a factor of $n/2$. The maximum packet size is reduced approximately by a factor of n.

An alternative routing for ***n-port*** communication, is to divide the data set $(\frac{PQ}{N})$ for each node into n equal parts (if $\frac{PQ}{N}$ mod $n = 0$) and route the parts according to SBT's ***rotated***, or ***reflected***, with respect to each other, or a combination thereof. The minimum time for ***one-to-all personalized communication*** using n distinctly rotated spanning binomial trees is $T = \frac{1}{n}(1-\frac{1}{N})PQt_c + n\tau$ [3], i.e., the same complexity as that of the lower bound. The lower bound is achieved for $B_m \geq \max\binom{n}{i}\frac{1}{n}\frac{PQ}{N} \approx \sqrt{\frac{2}{\pi}\frac{PQ}{n^{3/2}}}$, $1 \leq i \leq n$.

[2]except the conversions between binary code and Gray code encoding in which both use the same partitioning scheme in the same direction.

For $\frac{PQ}{N} = k < n$ the SBnT routing has a lower time complexity for element transfers. For k SBT's the transfer time for optimally rotated spanning binomial trees is $(2^n - 1)\frac{2^{\frac{n}{k}-1}}{2^{\frac{n}{k}}-1}\frac{PQ}{N}t_c$ and for optimally reflected and rotated spanning binomial trees the minimum transfer time with concurrent communication on all ports is $(2^n - 1)\frac{2^{\frac{2n}{k}-1}+1}{2^{\frac{2n}{k}}-1}\frac{PQ}{N}t_c$.

3.3.2 All-to-All Personalized Communication

For ***all-to-all personalized communication*** a simple exchange algorithm scanning through the dimensions of the cube attains the lower bound, $T_{min} = 2n(\frac{PQ}{2N}t_c + \tau)$, for ***one-port*** communication [10]. In each communication $\frac{PQ}{2N}$ elements are transfered. The exchange algorithm routes elements from a node to all other nodes according to a SBT. With ***n-port*** communication pipelining can be employed in the exchange algorithm, but the algorithm so modified is suboptimal. However, routing based on spanning balanced n-trees, or rotated spanning binomial trees, attains the lower bound, $T_{min} = \frac{PQ}{2N}t_c + n\tau$, ignoring lower order terms [10].

For the exchange algorithm [7] presented next it is assumed that the matrix is partitioned consecutively by rows and that processor i initially holds the elements of the i^{th} block row. After the transpose operation it shall hold the elements of the i^{th} block column. Note that the number of rows in a block row is different from the number of columns in a block column, unless $P = Q$. However, the number of elements in a block row and a block column are the same. For the transpose operation the block row of each processor is partitioned by columns into N equally sized blocks. The transpose is formed by processor i exchanging its j^{th} block with the i^{th} block of processor j. The data array in each processor holding the elements of a block row is two-dimensional, unless the number of rows is equal to the number of processors, and the local data array after the transposition is also two-dimensional, unless the number of columns is less than or equal to the number of processors. To complete the transposition after the interprocessor communication is completed, this two-dimensional data array can be transposed further locally, explicitly, or implicitly by indirect addressing.

```
/* An Exchange Algorithm: */
for $j := n - 1$ downto 0 do
        if (bit $j$ of my-addr $= 0$) then
                exchange blocks $\frac{1}{2}N$ to $N - 1$ of my blocked array
                        with my neighbor in dimension $j$
        else
                exchange blocks 0 to $\frac{1}{2}N - 1$ of my blocked array
                        with my neighbor in dimension $j$
        endif;
        shuffle my blocked array;
enddo
```

The loop can also be performed with the loop index running in the opposite order, but then the first operation in the loop shall be an unshuffle operation, which replaces the shuffle operation at the end of the loop.

```
/* A SBnT Algorithm: */
/* Let the format of msg be (source-addr, relative-addr, data). */
for all $j \neq myaddr$ do
    form msg for processor $j = (myaddr, myaddr \oplus j \oplus 00..01_b0..0, data)$
        and append to output-buf [b] where b is the base of $myaddr \oplus j$.
loop n times
    send concurrently for all n output ports.
    receive concurrently for all n input ports.
    for each j do, $0 \leq j < n$
        for each msg of input-buf [j] do
            if relative-addr = 0 then
                put the data into the source-addr^{th} block of the target array
            else
                form relative-addr := relative-addr $\oplus$ $(0..01_p0..0)$ in
                    the msg and append to output-buf [p], where p is
                    the bit position of relative-addr which is the
                    nearest 1-bit to the left of the $j^{th}$ bit cyclically.
                /* Note: $j^{th}$ bit is always 0 here. */
            endif
        enddo
    enddo
endloop
```

For both the SBT and SBnT algorithms presented above it is assumed that the partitions are embedded in the cube by a binary encoding. For Gray code encoding of partitions and binary encoding locally, we can first perform a transformation locally such that block i is moved to block location $G(i)$, then carry out the above algorithms. The two operations can also be combined as described in the next section for two-dimensional partitioning.

4 Two-Dimensional Partitioning

4.1 Relationships Between Different Data Structures and Their Encodings

As with one-dimensional partitioning there is a multitude of cases. If there is no particular reason for identifying a particular processor with a particular partition, then renaming of the processors suffices to realize the transpose. The processors that were assigned to column partitions will be assigned to row partitions after such a relabelling.

With the same number of partitions in the row and column directions, the transpose operation of a matrix partitioned by the consecutive strategy in both dimensions implies the same interprocessor communication as if the partitioning is made cyclicly in both dimensions (or by the same combination in both dimensions). An exchange of data takes place between distinct pairs of partitions. What matrix elements are exchanged depends on the partitioning strategy, as is easily seen if the concatenated bitfields of the matrix row and column indices is partitioned with the first (or last) $\log N_r = \log N_c$ bits of the row and column halves of the index field making up the partition address. In [6,7] we show that transposition of a matrix with partitions embedded in the cube by a binary code or Gray code encoding implies the same communication.

If the number of row and column partitions are different, then the transpose operation is no longer a pure exchange operation between a pair of processors. Some one-to-many and many-to-one communications are necessary. For example, assume that there are two row partitions and four column partitions. Then some partitions exchange data with one other partition, and some partitions with two other partitions. If *virtual* partitions are introduced in the row direction such that there are equally many row partitions, then the transpose operation becomes equivalent to the canonical case having as many row as column partitions. The number of virtual partitions is $2^{|\log N_r - \log N_c|}$. With virtual partitions one dimension is "collapsed" to a certain degree.

Conversion between cyclic storage and consecutive storage in the row or column direction is equivalent to a number (N_c or N_r) of independent one-dimensional conversions. Conversion in both dimensions is equivalent to *all-to-all personalized communication*, if $Q \geq N_c^2$ and $P \geq N_r^2$. In the two-dimensional conversion the assignment of matrix rows to partitions is changed from being determined by the last $\log N_r$ bits of the matrix row index to the first $\log N_r$ bits, or vice versa, and the column assignment is changed similarly according to the last and first $\log N_c$ bits of the matrix column index. The source cube node address is defined by the concatenated last $\log N_r$ bits of the row index and $\log N_c$ bits of the column index, and the destination address by the concatenated first $\log N_r$ bits of the matrix row index and $\log N_c$ bits of the column index. Clearly, by a suitable permutation of the bits in the concatenated row and column index encoding the two-dimensional conversion is equivalent to a one-dimensional conversion on a cube with $\log N_r + \log N_c = \log N$ dimensions. Conversion of storage form and transposition can be combined with no increase in communication complexity [4].

4.2 Algorithms

We consider the transposition operation for binary encoding first. Define $tr(i)$ to be the function which maps the address of partition $i = (ra||ca)$ to the address of the transposed partition, i.e., $tr(i) = (ca||ra)$. Let $D(i) = |ra \oplus ca|$, i.e., the value of $D(i)$ is equal to the number of bits that differ in ra and ca. The distance between i and $tr(i)$ is $2D(i)$. We assume that there are equally many row and column partitions.

The *Single path Recursive Transpose* (SRT) algorithm [7] uses one path from node i to $tr(i)$. Paths for different i are edge-disjoint, and pipelining of communications can be employed to reduce the communication complexity. The *Dual paths Recursive Transpose* (DRT) algorithm is a straightforward improvement of the *SRT* algorithm in that two directed edge-disjoint paths are established from each source node to its corresponding destination node. In the *Multiple paths Recursive Transpose* (MRT) algorithm, we partition all the nodes into sets having equivalent properties with respect to an operator. In [4] we show that the paths of any two nodes in different sets are edge-disjoint, and prove that all the nodes in the same set share the same set of edges, but use them during different cycles.

4.3 The Single Path Recursive Transpose (SRT) Algorithm

The *Single path Recursive Transpose* (SRT) algorithm [7] for a two-dimensional, consecutively partitioned matrix, exchanges data between the upper right $P/2 \times Q/2$ submatrix ($ra_{\frac{n}{2}-1} = 0$, $ca_{\frac{n}{2}-1} = 1$) and the lower left submatrix ($ra_{\frac{n}{2}-1} = 1$, $ca_{\frac{n}{2}-1} = 0$) in two steps. The transpose operation is completed by recursively applying the operation to each of the four submatrices. The implied routing corresponds to directed edge-disjoint paths [6] from each node i to $tr(i)$. This path only goes through the appropriate dimensions of the cube corresponding to the bits of the source node address i that need to be complemented to become the destination node address $tr(i)$. The routing order for the dimensions is the same for all nodes, for instance highest to lowest order for both row and column encoding, i.e., $ra_{\frac{n}{2}-1}, ca_{\frac{n}{2}-1}, ra_{\frac{n}{2}-2}, ca_{\frac{n}{2}-2}, \ldots, ra_0, ca_0$. The length of the path of node i is $2D(i)$. The first packet for each node on the anti-diagonal arrives after n routing steps and additional packets every cycle thereafter. The total number of routing steps is $\lceil \frac{PQ}{B_m N} \rceil + n - 1$. The nodes which are not on the anti-diagonal can either finish the transposition early in a "greedy" manner, or synchronize with the anti-diagonal nodes, i.e., the packet with the same ordinal number of all the nodes uses the same dimension (or idles) during the same step. The total transposition time T is $(\frac{PQ}{B_m N} + n - 1)(B_m t_c + \tau)$. The optimal packet size, B_{opt}, is $\sqrt{\frac{PQ\tau}{N(n-1)t_c}}$ and the minimum time, $T_{min} = (\sqrt{\frac{PQ}{N} t_c} + \sqrt{(n-1)\tau})^2$.

4.4 The Dual Paths Recursive Transpose (DRT) Algorithm

The *SRT* algorithm can be improved by establishing two directed edge-disjoint paths between i and $tr(i)$ for all i. In addition to the path used in the *SRT* algorithm, a second path is defined by permuting row and column dimensions pairwise to yield a routing order selected from $ca_{\frac{n}{2}-1}$, $ra_{\frac{n}{2}-1}$, $ca_{\frac{n}{2}-2}$, $ra_{\frac{n}{2}-2}$, ...,ca_0, ra_0. The two directed paths are edge-disjoint (as observed in [8] for the solution of tridiagonal systems on Boolean cubes). Moreover, the two directed paths for any i are edge-disjoint with respect to all paths for other i. This second path can be used to reduce the time for data transfer by splitting the set of data $\frac{PQ}{N}$ into two equal parts. The path lengths are already minimal in the *SRT* algorithm.

4.5 The Multiple Paths Recursive Transpose (MRT) Algorithm

For the *Multiple paths Recursive Transpose* (MRT) algorithm we define $2D(i)$ paths, labeled $0, 1, \ldots, 2D(i) - 1$, between nodes i and $tr(i)$. The paths differ in the order in which the dimensions are routed. All paths have the same length. Let $\alpha_{D(i)-1}$, $\alpha_{D(i)-2}$,..., α_0, $\beta_{D(i)-1}$, $\beta_{D(i)-2}$,..., β_0 be the sequence of dimensions that need to be routed in descending order. We describe a pair of paths as a sequence of dimensions.

$$path\ p = \begin{cases} \alpha_{(p+D(i)-1)\bmod D(i)}, \beta_{(p+D(i)-1)\bmod D(i)}, \alpha_{(p+D(i)-2)\bmod D(i)}, \beta_{(p+D(i)-2)\bmod D(i)}, \\ \qquad \ldots, \alpha_p, \beta_p. \qquad 0 \le p < D(i); \\ \beta_{(j+D(i)-1)\bmod D(i)}, \alpha_{(j+D(i)-1)\bmod D(i)}, \beta_{(j+D(i)-2)\bmod D(i)}, \alpha_{(j+D(i)-2)\bmod D(i)}, \\ \qquad \ldots, \beta_j, \alpha_j. \qquad j = p - D(i), D(i) \le p < 2D(i). \end{cases}$$

For example, if i = (1001||0100), then ra = 1001, ca = 0100, $D(i)$ = 3 and $tr(i)$ =

$(ca||ra) = (0100||1001)$. The distance between i and $tr(i)$ is 6. The 6 paths are defined as follows.

$$\begin{array}{ll} path\ 0 = 7,3,6,2,4,0. & path\ 3 = 3,7,2,6,0,4. \\ path\ 1 = 4,0,7,3,6,2. & path\ 4 = 0,4,3,7,2,6. \\ path\ 2 = 6,2,4,0,7,3. & path\ 5 = 2,6,0,4,3,7. \end{array}$$

Path 0 starts from the source node (10010100) and goes through nodes (00010100), (00011100), (01011100), (01011000), (01001000) and reaches the destination node (01001001). Path p can be derived by a right rotation of two steps of path $(p-1) \bmod D(i)$, if $0 \le p < D(i)$. For $D(i) \le p < 2D(i)$, path p can be derived by a right rotation of two steps of path $((p-1) \bmod D(i)) + D(i)$ and also by permuting row and column dimensions pairwise of path $(p-1) \bmod D(i)$. Note that path 0 is the same as the path defined in the *SRT* algorithm. Paths 0 and $D(i)$ are the two paths defined for node i in the *DRT* algorithm.

For the routing, the data from node i is split into $4D(i)$ packets of size $\lceil \frac{PQ}{4ND(i)} \rceil$ each. The packets are sent during the first two cycles. The first $2D(i)$ packets will arrive at the destination node, $tr(i)$, after $2D(i)$ cycles, and the second set during the next cycle. The total transpose time is [4].

$$T = \begin{cases} (n+1)\tau + (\frac{n+1}{2n})\frac{PQ}{N}t_c & \text{if } \frac{n}{2} \ge \frac{PQt_c}{8N\tau}; \\ 3\tau + \frac{3}{4}\frac{PQ}{N}t_c & \text{otherwise.} \end{cases}$$

The transpose time decreases as a function of $D(i)$ for $1 \le D(i) \le \sqrt{\frac{PQt_c}{8N\tau}}$ and increases for $\sqrt{\frac{PQt_c}{8N\tau}} \le D(i)$. The transpose time for $D(i) = 1$ and $D(i) = \frac{PQt_c}{8N\tau}$ are the same. The maximal packet size is $\frac{PQ}{4N}$. The maximal packet size can be reduced either without affecting the total transpose time (if $\frac{n}{2} \ge \frac{PQt_c}{8N\tau}$) or the total transpose time reduced by splitting the data into $\lfloor \frac{n}{2D(i)} \rfloor * 4D(i)$ packets. In fact, the data sent from node i can be split into $4kD(i)$ packets instead of $4D(i)$ packets. The whole routing completes in $2kD(i)+1$ cycles. Hence, $T = (2kD(i)+1)(\tau + \frac{PQ}{4kD(i)N}), 1 \le D(i) \le \frac{n}{2}$. The optimal k is $\sqrt{\frac{PQt_c}{2N\tau}} \frac{1}{2D(i)}$ and $T_{min} = (\sqrt{\tau} + \sqrt{\frac{PQt_c}{2N}})^2$. Notice that T_{min} is valid only when $k \ge 1$, which implies $\sqrt{\frac{PQt_c}{2N\tau}} \ge n$.

Theorem 1 *The total matrix transpose time by the MRT algorithm is*

$$T = \begin{cases} (n+1)\tau + \frac{n+1}{2n}\frac{PQ}{N}t_c & \textit{if } n \ge \sqrt{\frac{PQt_c}{N\tau}} \textit{ approximately;} \\ (\frac{n}{2}+3)\tau + \frac{n+6}{2n+8}\frac{PQ}{N}t_c & \textit{if } \sqrt{\frac{PQt_c}{2N\tau}} < n \le \sqrt{\frac{PQt_c}{N\tau}} \textit{ approximately and } \frac{n}{2} \textit{ is even;} \\ (\frac{n}{2}+2)\tau + \frac{n+4}{2n+4}\frac{PQ}{N}t_c & \textit{if } \sqrt{\frac{PQt_c}{2N\tau}} < n \le \sqrt{\frac{PQt_c}{N\tau}} \textit{ approximately and } \frac{n}{2} \textit{ is odd;} \\ (\sqrt{\tau} + \sqrt{\frac{PQt_c}{2N}})^2 & \textit{if } n \le \sqrt{\frac{PQt_c}{2N\tau}}. \end{cases}$$

and the maximal packet size is

$$B = \begin{cases} \lceil \frac{PQ}{N(n+4)} \rceil & \textit{for even } \frac{n}{2} \textit{ and } n > \sqrt{\frac{PQt_c}{2N\tau}}; \\ \lceil \frac{PQ}{N(n+2)} \rceil & \textit{for odd } \frac{n}{2} \textit{ and } n > \sqrt{\frac{PQt_c}{2N\tau}}; \\ \sqrt{\frac{PQ\tau}{2Nt_c}} & \textit{for } n \le \sqrt{\frac{PQt_c}{2N\tau}}. \end{cases}$$

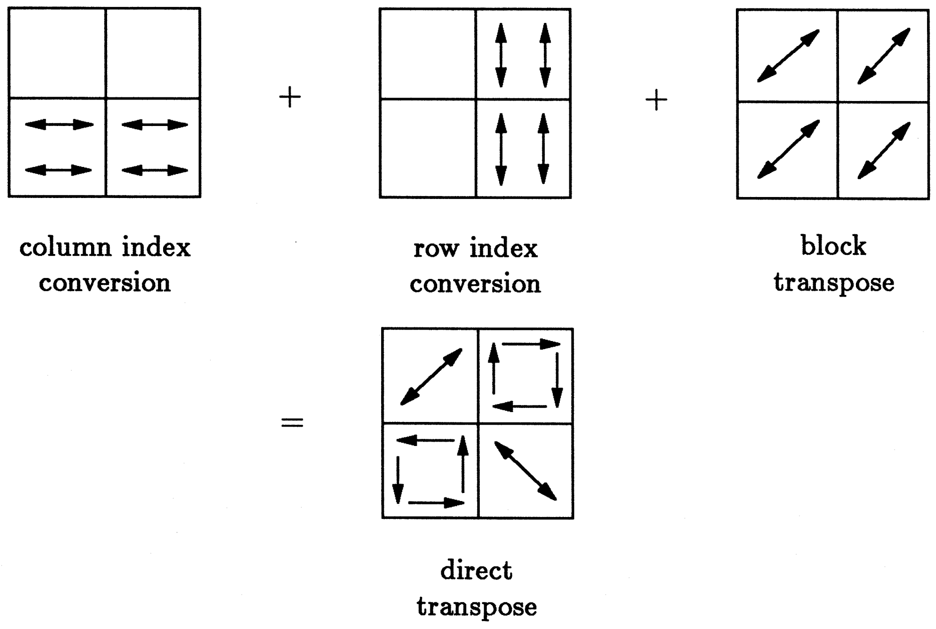


Figure 3: Transpose of a matrix stored by binary code encoding of row index and Gray code encoding of column index.

Proposition 2 *The matrix transposition time $T \geq n\tau + \frac{PQ}{2N}t_c$.*

Proof: The minimum number of communications is determined by the longest path. Nodes on the main anti-diagonal are at distance n. For a lower bound on the required time for data transfer consider the upper right $\sqrt{N}/2 \times \sqrt{N}/2$ submatrix. There are $N/4$ nodes. Each node has to send $\frac{PQ}{N}$ data to some node outside the submatrix. There are two dimensions per node that connects to nodes outside of the submatrix, i.e., a total of $2N/4$ links. Hence, the data transfer requires a time of at least $\frac{PQ}{2N}t_c$. ▮

The transposition algorithm above can be applied also if both row and column indices are encoded in Gray code. In the binary encoded case matrix element (i,j) is stored in processor $i||j$ and matrix element (j,i) in processor $j||i$. The two dimensional transpose algorithms described above defines a permutation between processor $ra||ca$ and processor $ca||ra$, $\forall 0 \leq ra < P, ca < Q$. For Gray code encoding of row and column indices, matrix element (i,j) is stored in processor $G(i)||G(j)$ and matrix element (j,i) is stored in processor $G(j)||G(i)$. It follows that the permutation will transpose the matrix.

If row and column indices are encoded in the same way, the transpose algorithm only depends on the processor addresses, not on the row and column indices of the matrix elements in the processors. For $N < PQ$, the argument applies to matrix blocks instead of matrix elements.

It is also possible to combine matrix transposition with Gray code to binary code conversion without increasing the communication time [4]. The data movement for mixed encoding is illustrated in Figure 3.

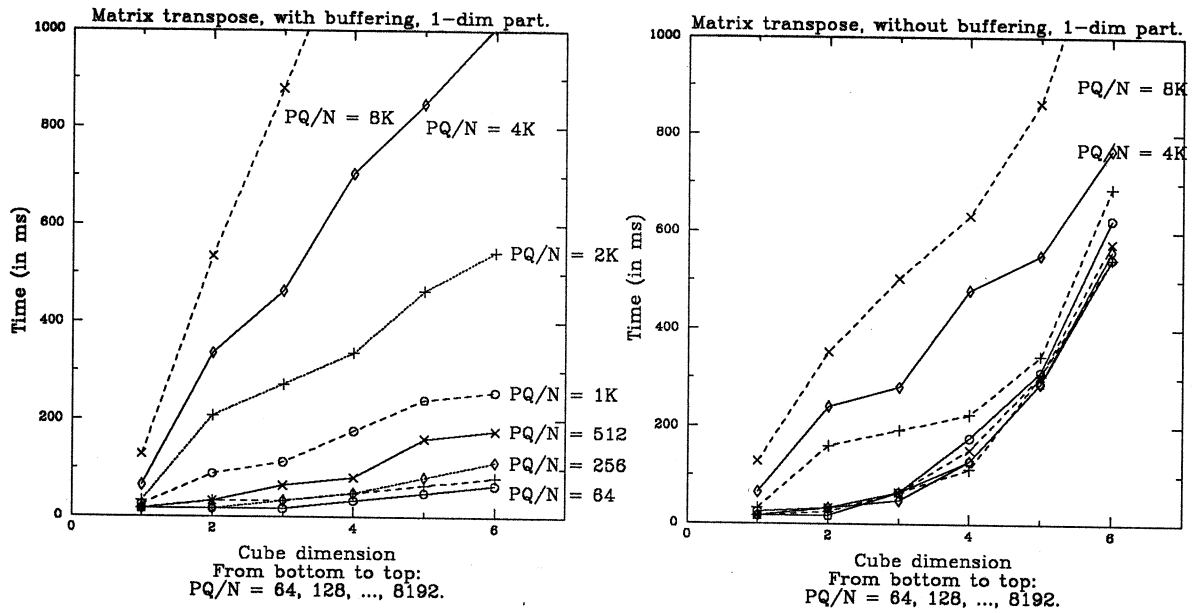


Figure 4: Measured times on the Intel iPSC for the transpose of a matrix, one-dimensional partitioning, encoded in binary code.

5 Experiments

5.1 One-Dimensional Partitioning

The Intel iPSC effectively allows only *one-port* communication. Hence, we implemented the one-dimensional transpose using the exchange algorithm. However, our implementation deviates from the above description in that we do not perform the shuffle operations explicitly, since the copying time on the Intel iPSC is significant. Copying 1024 single precision floating-point numbers (4 kbytes) takes about 37 *msec* according to our measurements. Instead, we logically partition the local array into 2^j same-sized blocks during step j. The odd or even blocks can either be sent directly to minimize the copy time, or copied into a buffer to reduce the number of start-ups. Figure 4 presents the measurements for unbuffered and buffered communication for rearrangement of cyclic to consecutive partitioning (one-dimensional local arrays).

The complexity of the unbuffered communication is easily found to be $T = n\frac{PQ}{N}t_c + (N + \lceil \frac{PQ}{2B_mN} \rceil \min(n, \log_2\lceil \frac{PQ}{B_mN} \rceil) - \frac{PQ}{B_mN})2\tau$. With buffered communication, messages may initially be larger than the buffer size, in which case they are sent directly. Small messages are buffered and the time for communication is $T = n\frac{PQ}{N}t_c + (\min(n, \log\lceil \frac{PQ}{B_mN} \rceil)\lceil \frac{PQ}{2B_mN} \rceil + \min(N, \frac{PQ}{B_{copy}N}) - \min(N, \frac{PQ}{B_mN}) + \lceil \frac{PQ}{2B_mN} \rceil \max(0, n - \log\lceil \frac{PQ}{B_{copy}N} \rceil))2\tau + \frac{PQ}{N}\max(0, n - \log\lceil \frac{PQ}{B_{copy}N} \rceil)t_{copy}$, where B_{copy} is the array size beyond which it is preferable to send without copying into a buffer. The complexity of the unbuffered communication grows linearly in the number of processors, i.e., exponentially in the number of cube dimensions, as shown in Figure 4. The buffered communication grows linearly in the number of cube dimensions. For a low growth rate it is important to have a large buffer, to reduce the number

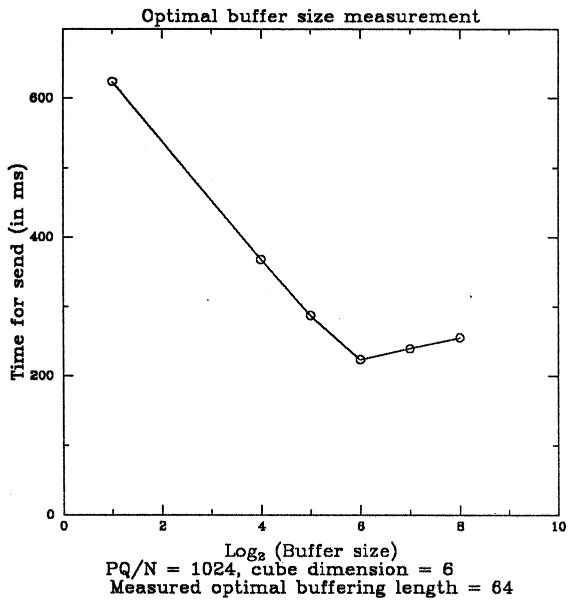


Figure 5: Performance measurements for optimum buffer size on the Intel iPSC.

of start-ups, and fast copy. With the times for copy of floating-point numbers and communication start-ups on the Intel iPSC the copy of 64 single-precision floating-point numbers (256 bytes) takes approximately the same time as one communication start-up, Figure 5. It is beneficial to send blocks of length at least 64 floating-point numbers without buffering. Figure 6 shows the improvement in performance with optimum buffering compared to the unbuffered communication. Note that for sufficiently small cubes (or large data sets) the time required by the two schemes coincide.

5.2 Two-Dimensional Partitioning

5.2.1 The Intel iPSC

We have implemented algorithm *SRT* as a step by step procedure. On the Intel iPSC it is necessary to rearrange two-dimensional arrays into one-dimensional arrays before sending. Since the copy time is significant we arrive at an estimate for the time of a block transpose of $T = (\frac{PQ}{N}t_c + \lceil\frac{PQ}{B_mN}\rceil\tau)n + 2\frac{PQ}{N}t_{copy}$. The growth rate is proportional to the number of matrix elements. There is an exponential decay as well as a linear increase in T as a function of the number of cube dimensions. Figure 7 shows measured values for the copy time, the communication time and the total time for a 2-cube and a 6-cube. As expected, the copy time for the 6-cube is lower than that for the 2-cube. Also, the communication time is essentially determined by the number of start-ups, which for the 6-cube remains the same for $PQ \leq 64$ kbytes.

Figure 8(a) shows the total transpose time as a function of the number of cube dimensions and matrix size. For small matrices the number of communication start-ups dominates and the total time increases with the number of cube dimensions, but as the matrix size increases the transpose time decreases with increased cube size.

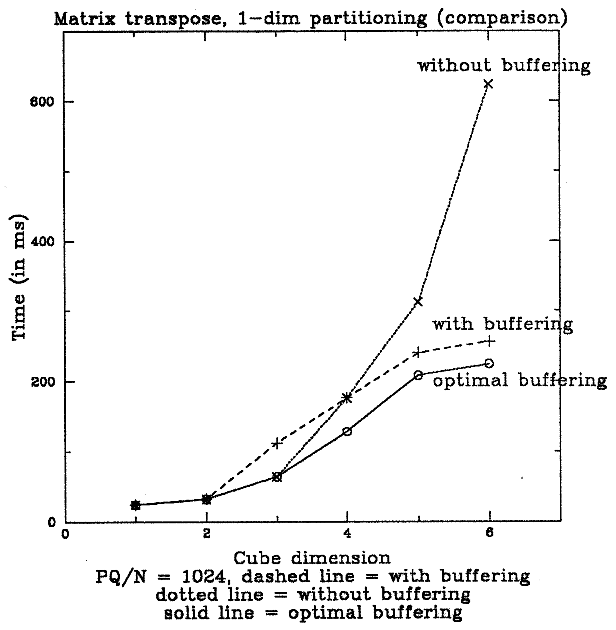
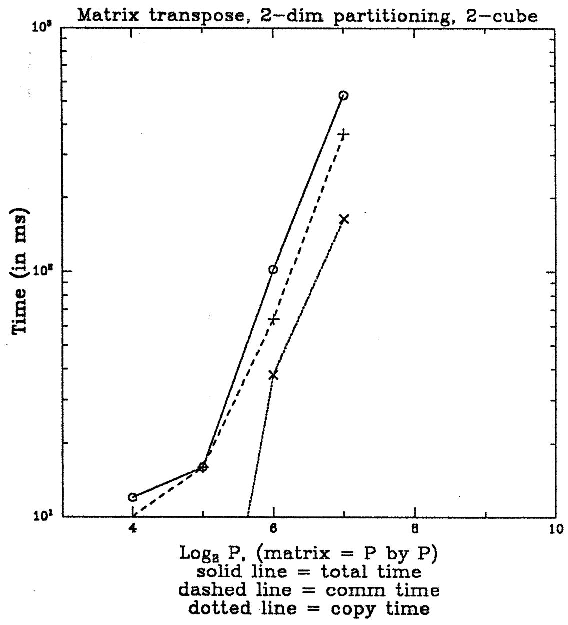


Figure 6: The effect of optimum buffering on performance for matrix transpose on the Intel iPSC.



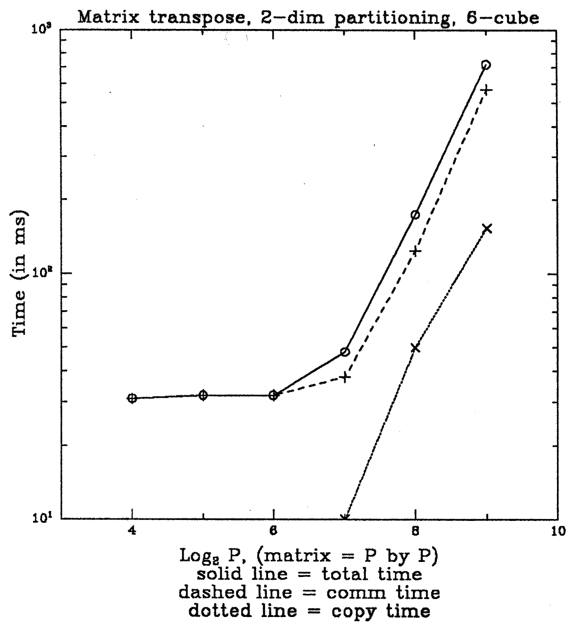
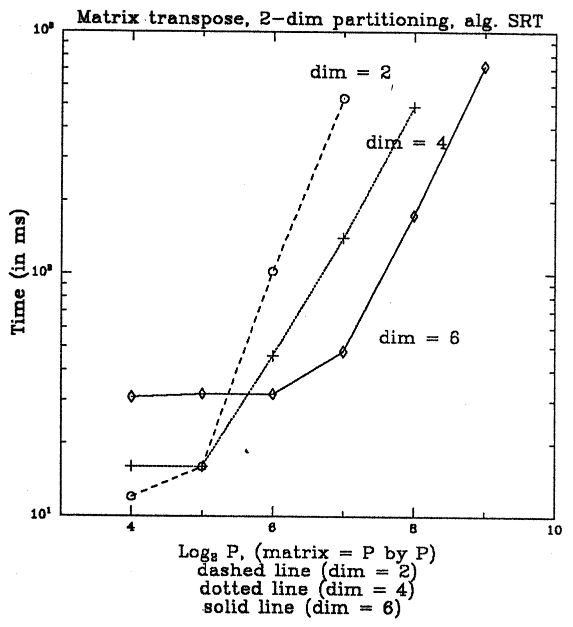


Figure 7: Performance measurements for matrix block transpose on the Intel iPSC.



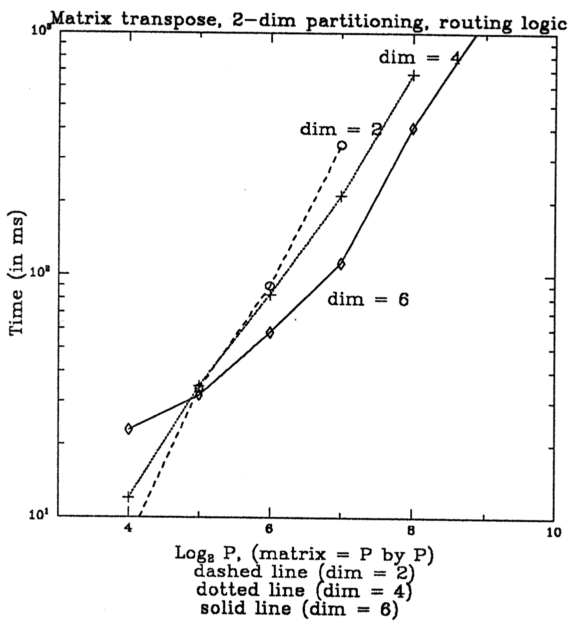


Figure 8: Measured times for block matrix transpose on the Intel iPSC using the *SRT* algorithm without pipelining (a) and using routing logic (b).

On the Intel iPSC it is also possible to carry out the transpose operation by a direct send to the final destination. Figure 8(b) gives the times measured for matrix transpose using the routing logic alone. As the cube size increases the recursive block transpose algorithm yields a significantly better performance than the transpose time offered by the routing logic.

5.2.2 The Connection Machine

We have also implemented matrix transposition on the Connection Machine. It has a bit-serial, pipelined communication system. The recursive algorithm does not exploit this feature, but the routing logic does. Figure 9 shows the transpose time using the routing logic. Each processor holds one matrix element (32-bits). Figure 10 shows the transpose times for various number of matrix elements per processor, and for various number of processors. Figure 11 shows the transpose times for two fixed sized matrices on various sizes of the Connection Machine.

6 Comparison and Conclusion

It is of interest to compare the times for matrix transpose based on a one-dimensional partitioning and a two-dimensional partitioning.

$$T^{2d} = (\frac{PQ}{N}t_c + \lceil \frac{PQ}{B_m N} \rceil \tau)n + 2\frac{PQ}{N}t_{copy}$$

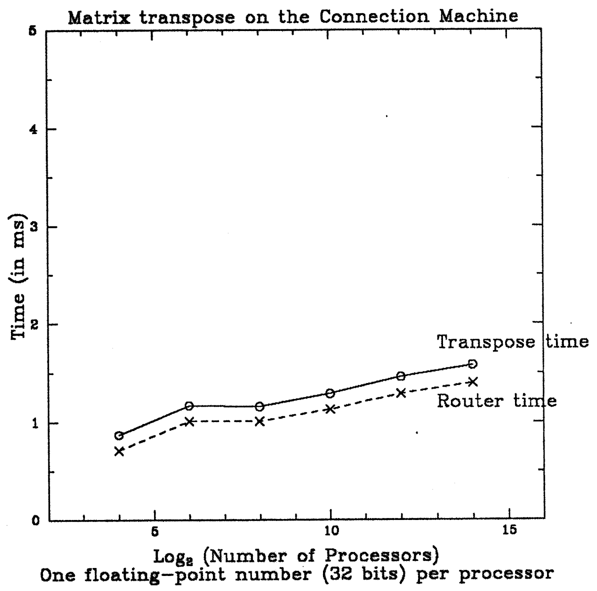


Figure 9: Matrix transpose on the Connection Machine. One element per processor.

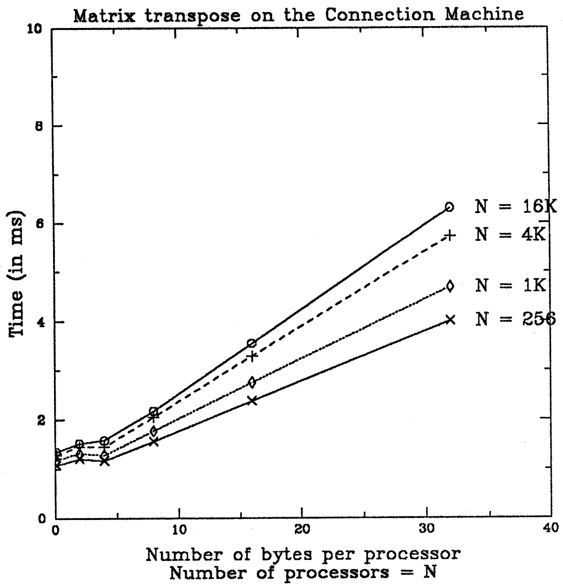


Figure 10: Matrix transpose on the Connection Machine. Multiple elements per processor.

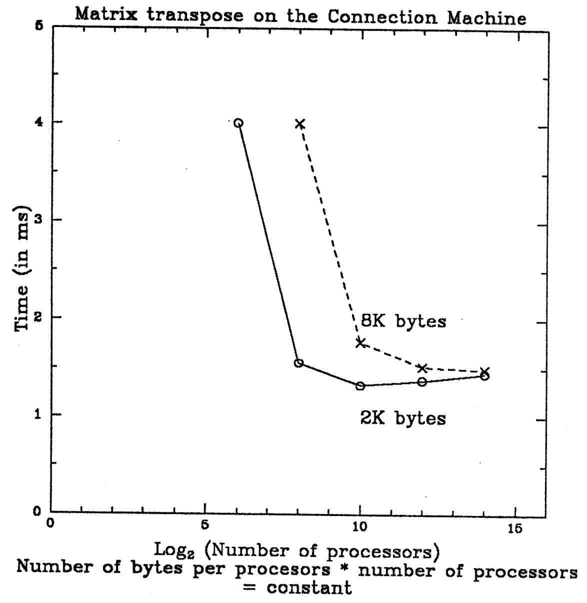


Figure 11: Matrix transpose on the Connection Machine as a function of the machine size.

$$\begin{aligned} T^{1d} \;=\; & (\min(n, \log\lceil \frac{PQ}{B_m N} \rceil)\lceil \frac{PQ}{2B_m N} \rceil + \min(N, \frac{PQ}{B_{copy} N}) - \min(N, \frac{PQ}{B_m N}) \\ & + \lceil \frac{PQ}{2B_m N} \rceil \max(0, n - \log\lceil \frac{PQ}{B_{copy} N} \rceil))2\tau \\ & + n\frac{PQ}{N} t_c + \frac{PQ}{N} \max(0, n - \log\lceil \frac{PQ}{B_{copy} N} \rceil) t_{copy} \end{aligned}$$

For $\log \frac{PQ}{B_m N} \geq n$ and $\log \frac{PQ}{B_{copy} N} \geq n$ there is no buffering of communication for the strip transpose, i.e., no copy and a lower communication complexity than for the block transpose. The block transpose always needs an initial and final copy. If the local data structures for the strip transpose are two-dimensional arrays, the same copy operations may be required for both the strip and the block transpose. But, the conclusion remains that for problems which are large relative to the size of the cube, the one-dimensional partitioning is most efficient with respect to performance.

For a cube with a size approaching the size of the matrix the copy time for the one-dimensional partitioning grows, and so does the number of start-ups. Both eventually may be higher than that for the two-dimensional transpose. We conclude that for sufficiently large cubes the block transpose is preferable for the Intel iPSC. Figure 12 gives the experimental evidence for this conclusion.

Note that if the copy time can be ignored, then the one-dimensional partitioning always yields a better performance than a two-dimensional partitioning for *one-port* communication.

With *n-port* communication the transfer time for the two-dimensional partitioning decreases exponentially in the number of cube dimensions, but for the optimum packet size

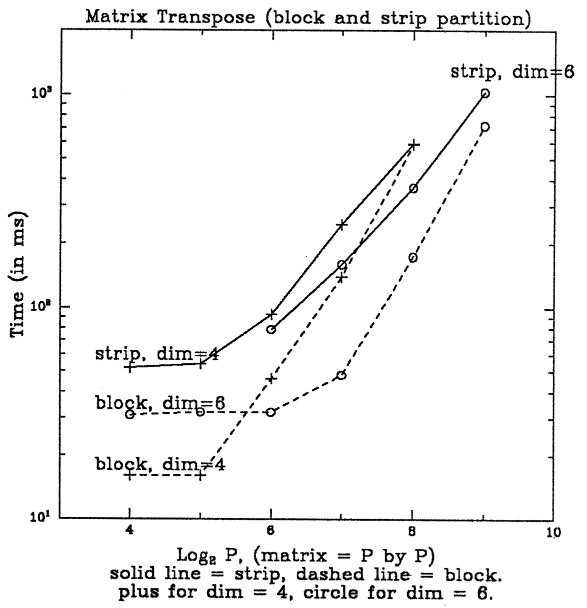


Figure 12: Comparison of matrix transposition of one- and two-dimensionally partitioned matrices on the Intel iPSC.

the number of start-ups is higher than for the one-dimensional partitioning.

$$T^{1d}_{min} = \frac{PQ}{2N} t_c + n\tau$$

and

$$T^{2d}_{min} = \begin{cases} (n+1)\tau + \frac{n+1}{2n}\frac{PQ}{N} t_c & \text{if } n \geq \sqrt{\frac{PQt_c}{N\tau}} \text{ approximately;} \\ (\frac{n}{2}+3)\tau + \frac{n+6}{2n+8}\frac{PQ}{N} t_c & \text{if } \sqrt{\frac{PQt_c}{2N\tau}} < n \leq \sqrt{\frac{PQt_c}{N\tau}} \text{ approximately and } \frac{n}{2} \text{ is even;} \\ (\frac{n}{2}+2)\tau + \frac{n+4}{2n+4}\frac{PQ}{N} t_c & \text{if } \sqrt{\frac{PQt_c}{2N\tau}} < n \leq \sqrt{\frac{PQt_c}{N\tau}} \text{ approximately and } \frac{n}{2} \text{ is odd;} \\ (\sqrt{\tau} + \sqrt{\frac{PQt_c}{2N}})^2 & \text{if } n \leq \sqrt{\frac{PQt_c}{2N\tau}}. \end{cases}$$

and the maximal packet size is

$$B = \begin{cases} \lceil \frac{PQ}{N(n+4)} \rceil & \text{for even } \frac{n}{2} \text{ and } n > \sqrt{\frac{PQt_c}{2N\tau}}; \\ \lceil \frac{PQ}{N(n+2)} \rceil & \text{for odd } \frac{n}{2} \text{ and } n > \sqrt{\frac{PQt_c}{2N\tau}}; \\ \sqrt{\frac{PQ\tau}{2Nt_c}} & \text{for } n \leq \sqrt{\frac{PQt_c}{2N\tau}}. \end{cases}$$

For $n \geq \sqrt{\frac{PQt_c}{N\tau}}$, the one-dimensional partitioning always yields a lower complexity than the two-dimensional partitioning. The difference is about one start-up time unless the cube is very small. For $\sqrt{\frac{PQt_c}{2N\tau}} < n \leq \sqrt{\frac{PQt_c}{N\tau}}$, the break even point (ignoring copy) can be computed to be

$$N \approx c\frac{r}{\log^2 r}$$

where $\frac{1}{2} < c < 1$ and $r = \frac{PQt_c}{\tau}$. For $n \leq \sqrt{\frac{PQt_c}{2N\tau}}$, the one-dimensional partitioning always yields a lower complexity than the two-dimensional partitioning.

In summary, if the copy time is ignored and communication is restricted to one port at a time, then the one-dimensional partitioning always yields a lower complexity than the two-dimensional partitioning. If the copy time is included then the two-dimensional partitioning yields a lower complexity for a sufficiently large cube. With concurrent communication on all ports the *Spanning Balanced n-Tree*(SBnT) routing can be used for the one-dimensional partitioning, and the copy times for one and two-dimensional partitioning should be comparable. The one-dimensional partitioning yields a lower complexity for a cube dimension n satisfying $n \geq \sqrt{\frac{PQt_c}{N\tau}}$ or $n \leq \sqrt{\frac{PQt_c}{2N\tau}}$.

In comparing the Intel iPSC with the Connection Machine we conclude that the latter performs a transpose about two orders of magnitude faster.

Acknowledgement This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564, and the Thinking Machines Corporation.

References

[1] J.O. Eklundh. A fast computer method for matrix transposing. *IEEE Trans. Computers*, C-21(7):801–803, 1972.

[2] W. Daniel Hillis. *The Connection Machine.* MIT Press, 1985.

[3] Ching-Tien Ho and S. Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *1986 Int. Conf. Parallel Processing*, pages 640–648, IEEE Computer Society, 1986. Tech. report YALEU/CSD/RR-483.

[4] Ching-Tien Ho and S. Lennart Johnsson. *Matrix Transposition on Boolean n-cube Configured Ensemble Architectures.* Technical Report YALEU/CSD/RR-494, Yale University, Dept. of Computer Science, September 1986.

[5] Ching-Tien Ho and S. Lennart Johnsson. *Spanning Balanced Trees in Boolean cubes.* Technical Report YALEU/CSD/RR-508, Yale University, Dept. of Computer Science, January 1987.

[6] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, 4(2):133–172, April 1987. (Report YALEU/CSD/RR-361, January 1985).

[7] S. Lennart Johnsson. *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures.* Technical Report YALEU/CSD/RR-367, Yale University, Dept. of Computer Science, February 1985.

[8] S. Lennart Johnsson. *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations.* Technical Report YALE/CSD/RR-339, Department of Computer Science, Yale University, October 1984.

[9] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Stat. Comp.*, 8(3):354–392, May 1987. (Report YALEU/CSD/RR-436, November 1985).

[10] S. Lennart Johnsson and Ching-Tien Ho. *Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes.* Technical Report YALEU/CSD/RR-500, Yale University, Dept. of Computer Science, November 1986. To appear in IEEE Trans. Computers.

[11] Oliver A. McBryan and Eric F. Van de Velde. *Hypercube Algorithms and Implementations.* Technical Report, Courant Institute of Mathematical Sciences, New York University, November 1985.

[12] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice Hall, 1977.

[13] Harold S. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. Computers*, C-20:153–161, 1971.