An Optimal Routing Algorithm for Mesh-Connected Parallel Computers

DAVID NASSIMI AND SARTAJ SAHNI

University of Minnesota, Minneapolis, Minnesota

ABSTRACT. An optimal algorithm to route data in a mesh-connected parallel computer is presented. This algorithm can be used to perform any data routing that can be specified by the permutation and complementing of the bits in a PE address. Matrix transpose, bit reversal, vector reversal, and perfect shuffle are examples of data permutations that can be specified in this way. The algorithm presented uses the minimum number of unit distance routing steps for every data permutation that can be specified as above.

KEY WORDS AND PHRASES: parallel algorithm, mesh-connected computer, ILLIAC IV, permutation, complexity, data routing

CR CATEGORIES: 5.25, 5.31, 6.22

1. *Introduction*

It is well known that the performance of a parallel computer is often limited by the time spent communicating data from one processing element (PE) to another. For example, the sorting algorithms developed by Thompson and Kung [16] and Nassimi and Sahni [9] make use of only $O(\log^2 n)$ comparison steps to sort n^2 elements on an $n \times n$ mesh-connected computer (MCC). These algorithms, however, require $O(n)$ data routing steps. For large n, the time spent doing useful work (i.e., comparisons) is negligible compared to the time spent moving data from one PE to another. As another example, consider matrix multiplication and inversion. It is easy to see that if enough PE's are available, then two $n \times n$ matrices can be multiplied using $O(\log n)$ arithmetic steps. Csanky [4] has shown how to invert a matrix using only $O(\log^2 n)$ arithmetic steps on a parallel computer. However, it is also known (Gentleman [6]) that if an MCC is used, then $O(n)$ data routing steps have to be performed for both multiplication and inversion. Hence, the complexity of these operations is determined not by the time spent performing arithmetic operations but by the time needed to route data from PE to PE.

The problem of communicating data between PE's has received much attention in the literature, and many PE interconnection schemes have been suggested. Lang [7] and Stone [13] consider shuffle-exchange networks, Lawrie [8] considers Ω-networks, and Swanson [14] considers k-apart interconnections.

In this paper we are concerned only with mesh-connected computers (MCC's). The basic MCC model we look at is a q-dimensional generalization of an ILLIAC-IV-like computer

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

This research represents a portion of the first author's Ph.D. dissertation.

This research was supported in part by the National Science Foundation under Grants MCS 76-21024 and MCS 78-15455.

Authors' present addresses: D. Nassimi, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201; S. Sahni, Department of Computer Science, 136 Lind Hall, 207 Church St. S.E., University of Minnesota, Minneapolis, MN 55455.

© 1980 ACM 0004-5411/80/0100-0006 $00.75

[1]. Our model differs from the ILLIAC IV essentially in the absence of end-around connections. Our model is an SIMD machine [5] with $N = 2^p$ identical PE's. The PE's may be thought of as arranged into a q-dimensional $n_{q-1} \times n_{q-2} \times \cdots \times n_0$ array, where each n_i is a power of 2. Two methods to identify a PE are used. One is the standard array element reference $PE(i_{q-1}, i_{q-2}, \ldots, i_0)$ with $0 \leq i_k \leq n_k - 1$, $0 \leq k < q$. In the second indexing method the PE's are numbered 0 to $N - 1$ using the standard row-major representation of a q-dimensional array. The row-major index of $PE(i_{q-1}, i_{q-2}, \ldots, i_0)$ is given by $m = \sum_{j=0}^{q-1} a_j i_j$, where $a_0 = 1$, and $a_j = a_{j-1} * n_{j-1}$ for $1 \leq j < q$. The PE with the row-major address m will be referred to as $PE(m)$. Or, representing the addresses in binary, $m = m_{p-1} m_{p-2} \cdots m_0$, the PE may be referred to as $PE(m_{p-1} m_{p-2} \cdots m_0)$. Note that since each n_j is a power of 2, the least significant $\log n_0$ bits in the row-major address correspond to i_0; the next $\log n_1$ bits correspond to i_1; ...; and the most significant $\log n_{q-1}$ bits correspond to i_{q-1}.

Our MCC model is the same as that used in [9] and [16] and has the following properties.

(1) Each processor $PE(i_{q-1}, \ldots, i_0)$ is connected to its (at most $2q$) nearest neighbors $PE(i_{q-1}, \ldots, i_k \pm 1, \ldots, i_0)$, $0 \leq k < q$, provided they exist.

(2) Each PE has at least three registers R_r, R_s, and R_t. R_r is the routing register and R_s and R_t are storage registers.

(3) The MCC has a routing instruction

$$\text{ROUTE}(k, j),$$

where $0 \leq k < q$ and $|j| < n_k$. The effect of this instruction is to move the data in the routing register of every PE to the routing register of the PE j units away along the kth dimension. That is, the contents of the routing register in $PE(i_{q-1}, \ldots, i_k, \ldots, i_0)$ are moved to $PE(i_{q-1}, \ldots, i_k + j, \ldots, i_0)$ for all $i_{q-1}, i_{q-2}, \ldots, i_0$. This move (or shift) may be regarded as "end-off." When j is positive, we refer to the route as a *positive route* along the kth dimension; otherwise it is referred to as a *negative route*. The time required by the hardware to execute this instruction is $a + b * |j|$, where a and b are constants. Each time a ROUTE(k, j) instruction is used, we shall say that one *long-route* and $|j|$ *unit-routes* have been used. One should note that the route instruction defined allows data to be routed along only one dimension at a time. Furthermore, all data are routed in the same direction. This is in accordance with [9] and [16].

(4) The MCC has a register transfer instruction

$$R' \leftarrow R$$

and a register swap instruction

$$R' \leftrightarrow R,$$

where R and $R' \in \{R_r, R_s, R_t\}$. It is possible to specify that only certain PE's be involved in the transfer or swap. This is done by specifying a PE selection or enable mask. We shall use a "processor address mask" as suggested by Siegel [12]. The mask has $\log N$ positions. Each position may have value 0, 1, or x. Each position in the mask corresponds to a bit in the binary representation of a PE address (using the row-major indexing scheme). The effect of the mask is to require enabling of all PE's whose address bits agree with the mask in all non-x positions. Thus, if $\log N = 3$ then the mask $0\ x\ 1$ enables PE(1) and PE(3). For notational simplicity, we shall specify the processor address mask by specifying a sequence of signed integers, $\pm i$, where $+i$ specifies that bit i is to be 1 and $-i$ specifies that bit i is to be 0. "Don't care" bits are not included in the sequence. For example, if $p = \log N = 3$ then the instruction

$$R_r \leftarrow R_s\ (-0, 2)$$

means that for each PE whose (row-major) address has a 0 in bit 0 and a 1 in bit 2 (i.e., the mask $1\ x\ 0$), the content of R_s is to be moved to R_r. So, PE(4) and PE(6) are the only PE's

for which the transfer is made. Note that the transfer for all enabled PE's is done in parallel.

Let c be the time required by a register transfer or swap instruction, and let a and b be the constants explained above. Then the total routing time of an algorithm is $N_1a + N_2b + N_3c$, where N_1, N_2, and N_3 are, respectively, the number of long-routes, unit-routes, and register transfers or swaps.

The permutation routing problem may be specified by a vector $D(m)$, $0 \le m \le N - 1$. Data are to be routed from PE(m) to PE(D(m)). Thompson [15] suggests solving this routing problem by mapping the Benes network [3] or the Waksman network [17] onto the mesh interconnection scheme of an MCC. This approach has some drawbacks. First, extensive preprocessing is required to determine which pairs of PE's are to exchange data at each stage. The best sequential preprocessing algorithm known [17] takes $O(N \log N)$ time. Second, the PE selectivity needed requires $O(N)$ bits, with bit i a zero iff PE(i) is not selected. This is undesirable (and impractical) when N is large. Siegel's selectivity scheme [12], as described above, requires only log N "bits," with each bit being one of 0, 1, x.

A second approach to this problem is to associate a destination tag $D(m)$ with the data item in PE(m), and then sort these tags (moving the associated data item each time a destination tag is moved). The sorting can be done using the efficient algorithms of [9] and [16]. The drawbacks of this approach are (i) the added complexity of creating and loading destination tags, and (ii) additional routing steps required to move the tags (i.e., in addition to those needed to move the data). However, both of the above approaches work for all permutations.

In this paper we study a subset F of all routings definable by permutations. A permutation is in F iff it can be defined by a vector $A = [A_{p-1}, A_{p-2}, \ldots, A_0]$ (recall that $N = 2^p$), where

(i) $A_i \in \{\pm 0, \pm 1, \ldots, \pm(p-1)\}$, $0 \le i < p$, and
(ii) $[|A_{p-1}|, |A_{p-2}|, \ldots, |A_0|]$ is a permutation of $[0, 1, \ldots, p-1]$.

The destination of the data item in each PE can be computed from this vector as follows. Consider PE($m_{p-1}m_{p-2} \cdots m_0$). The destination for the data in this PE is $d = d_{p-1}d_{p-2} \cdots d_0$ where for $i = 0, 1, \ldots, p-1$ we have

$$d_{|A_i|} = \begin{cases} m_i & \text{if } A_i \ge 0, \\ \bar{m}_i & \text{if } A_i < 0. \end{cases}$$

Note that we distinguish between +0 and −0 (−0 < +0 = 0). Also, note that the total number of permutations specifiable in this way is $2^p p! = N(\log N)!$.

Intuitively then for each permutation $P \in F$ the index of the destination PE for PE(m) is obtained by permuting the bits in the binary representation of m and complementing some bits. The vector A specifies how the bits are to be permuted and also which bits are to be complemented. $|A_i|$ tells us *where* bit i is to go, and the sign of A_i tells us if the ith bit in m is to be complemented.

As an example, consider the case $N = 16$, $p = 4$, and $A = [-0, 3, -1, -2]$. The destination PE for the data in PE($m = m_3m_2m_1m_0$) is $m_2\bar{m}_0\bar{m}_1\bar{m}_3$. So, for example, the data from PE(0) are to be moved to PE(7) and the data from PE(15) are to be moved to PE(8).

While F contains only N(log N)! of the possible N! permutations, it is nonetheless a very rich class. In fact, most of the common permutations arising in practice are included in F. For $N = 2^p$ processors, Stone's perfect shuffle is given by $A = [0, p-1, p-2, \ldots, 1]$; shuffled row-major [16] can be obtained from row-major on an $n \times n$ MCC using $A = [p-1, p/2-1, p-2, p/2-2, \ldots, p/2, 0]$; and a matrix transpose is given by $A = [p/2-1, p/2-2, \ldots, 0, p-1, p-2, \ldots, p/2]$. Table I gives the A vectors defining some of the commonly occurring permutations. (Note that p must be even for the first and the last two permutations in the table.) Bit reversal is encountered in the computation of the fast Fourier transform (FFT). In vector reversal the data in PE(m) are moved to

TABLE I. SOME COMMON PERMUTATIONS

Permutation	Vector representation
Matrix transpose	$[p/2 - 1, \ldots, 0, p - 1, \ldots, p/2]$
Bit reversal	$[0, 1, 2, \ldots, p - 1]$
Vector reversal	$[-(p - 1), -(p - 2), \ldots, -0]$
Perfect shuffle	$[0, p - 1, p - 2, \ldots, 1]$
Unshuffle	$[p - 2, p - 3, \ldots, 0, p - 1]$
Shuffled row-major	$[p - 1, p/2 - 1, p - 2, p/2 - 2, \ldots, p/2, 0]$
Bit shuffle	$[p - 1, p - 3, \ldots, 1, p - 2, p - 4, \ldots, 0]$

PE(N $-$ m $-$ 1). Bit shuffle can be used to unscramble from "shuffled row-major" to row-major. Unshuffle is the inverse of a perfect shuffle and restores row-major order from a perfect shuffle. Circular shifts and p-ordered vectors [11, 14] are examples of permutations not in F.

Orcutt [11] presents algorithms for perfect shuffle and bit reversal on an $n \times n$ MCC. Both of his algorithms use $O(\log n)$ long-routes and $4(n - 1)$ unit-routes. The algorithm we present in this paper can be used for every permutation A $\in$ F. In addition, our algorithm uses the absolute minimum number of unit-routes possible for each permutation in F. It thus performs a perfect shuffle faster than Orcutt's algorithm and uses only $2n$ unit-routes. A bit reversal is performed in $4(n - 1)$ unit-routes (the same as required by Orcutt's algorithm), which is the minimum number of unit-routes in which a bit reversal can be performed. However, while Orcutt has two different algorithms for these two permutations, we use the same algorithm for all A $\in$ F. Our algorithm spends $O(\log^2 N)$ time to determine how the routing is to be carried out.

The organization of this paper is as follows. First, in Section 2 we develop a lower bound, $\beta(A)$, on the number of unit-routes needed to perform the permutation A. This lower bound is used in Sections 3, 4, and 5 to develop the routing algorithm. In Section 3 we present two building blocks for our algorithm. In Section 4 we show how the permutation A is to be factored in order to route in the minimum number of unit-routes. The complete algorithm for a two-dimensional MCC is also given in this section. Section 5 presents the general algorithm for q-dimensional MCC's.

Finally, in Section 6 we determine the efficiency of the q-dimensional routing algorithm in case it is used on MCC's with wraparound. Let $\gamma(A)$ be the lower bound on the number of unit-routes needed to perform A when the MCC has a wraparound connection. It is shown that $\gamma(A)/\beta(A) \geq \frac{1}{2}$ when A contains no bit complementation, and $\gamma(A)/\beta(A) \geq \frac{1}{3}$ otherwise. Hence, the algorithm can be speeded by at most a factor of 3 if wraparound connections exist.

2. *The Lower Bound*

In this section we obtain a lower bound, $\beta(A)$, on the number of unit-routes needed to permute the data in a q-dimensional MCC according to the permutation A. The strategy used to arrive at this bound is based on the observation that moving the data from PE($i_{q-1}, i_{q-2}, \ldots, i_0$) to their destination PE($j_{q-1}, j_{q-2}, \ldots, j_0$) requires $|j_k - i_k|$ unit-routes along the kth dimension, $0 \leq k \leq q - 1$. The bound is derived by treating each dimension separately: for dimension k, we identify a PE for which the "distance" $|j_k - i_k|$ is maximum. The lower bound, $\beta(A)$, will be shown to be twice the sum of the maximum distance for each dimension. Before obtaining $\beta(A)$, we develop some terminology.

Let $m = m_{p-1}m_{p-2} \cdots m_0$ be the (row-major) address of PE($i_{q-1}, \ldots, i_0$) in an $n_{q-1} \times n_{q-2} \times \cdots \times n_0$ q-dimensional MCC. Define the set

$$I = \{p - 1, p - 2, \ldots, 0\}$$

as the set of *bit indices*. As pointed out in Section 1, each i_k corresponds to a subset of the bits in the address m. That is, the rightmost $\log n_0$ bits determine i_0, the next $\log n_1$ bits are equal to i_1, etc. Accordingly, we define I_k to be the set of bits corresponding to the kth

dimension, i_k. Thus $I_0 = \{\log n_0 - 1, \ldots, 0\}$, $I_1 = \{\log(n_1 n_0) - 1, \ldots, \log n_0\}$, etc. Any bit index $j \in I$ may be thought of as having two components $u(j)$ and $l(j)$, where $u(j)$ tells us which I_k the bit j falls in and $l(j)$ gives the position of the bit within I_k. More formally, $u(j) = k$ iff $j \in I_k$. And,

$$l(j) = j - \sum_{r=0}^{u(j)-1} \log n_r.$$

Example 2.1. Let $N = 64$; $q = 2$; $n_0 = 4$; and $n_1 = 16$. Six binary bits are needed to represent the index of any processor. Let $m = m_5 m_4 m_3 m_2 m_1 m_0$ be the index of $PE(i_1, i_0)$. It should be easy to see that $i_1 = m_5 m_4 m_3 m_2$ and $i_0 = m_1 m_0$. So, we have $I_1 = \{5, 4, 3, 2\}$; $I_0 = \{1, 0\}$; $u(5) = u(4) = u(3) = u(2) = 1$; $u(1) = u(0) = 0$; $l(5) = 3$; $l(4) = 2$; $l(3) = 1$; $l(2) = 0$; $l(1) = 1$; $l(0) = 0$.

Next, given any permutation A, we partition the set of bit indices, I, into four disjoint sets L, R, S, and E as below:

(i) L (bits moving *L*eft)

$$L = \{i \in I : u(|A_i|) = u(i) \text{ and } |A_i| > i\}.$$

That is, L is the set of all bits whose destination is a bit position to its left but in the same dimension.

(ii) R (bits moving *R*ight)

$$R = \{i \in I : u(|A_i|) = u(i) \text{ and } |A_i| < i\}.$$

So, R is the set of all bits whose destination is a bit position to its right but in the same dimension.

(iii) S (bits staying in the *S*ame position)

$$S = \{i \in I : |A_i| = i\}.$$

(iv) E (*E*xiting bits)

$$E = \{i \in I : u(|A_i|) \neq u(i)\},$$

i.e., E is the set of all bits with a destination in a different dimension.

For every bit-index set J, J^+ denotes the subset $\{i \in J : A_i \geq 0\}$ and J^- denotes the subset $\{i \in J : A_i < 0\}$. J_k denotes the subset $J \cdot I_k$ ($\cdot$ denotes set intersection). A(J) represents the set of destinations of the index set J. Hence, $A(J) = \{|A_i| : i \in J\}$.

Example 2.2. Consider the two-dimensional MCC of Example 2.1. Let the permutation be defined by $A = \{5, -3, 1, 4, 2, 0\}$. Figure 1 shows the bit movement. $L = \{2\}$; $R = \{4\}$; $S = \{0, 5\}$; and $E = \{1, 3\}$. $L_0 = \emptyset$; $L_1 = \{2\}$; $L^+ = L_1^+ = \{2\}$; $L^- = \emptyset$; $R_1^- = \{4\}$; and $A(\{0, 1, 2\}) = \{0, 2, 4\}$.

Let $m = m_{p-1} m_{p-2} \cdots m_0$ and $d = d_{p-1} d_{p-2} \cdots d_0$ be, respectively, the row-major addresses of $PE(i_{q-1}, \ldots, i_0)$ and $PE(j_{q-1}, \ldots, j_0)$. The distance $DIST_k(m, d)$ from m to d along dimension k is $j_k - i_k$, $0 \leq k < q$. Letting $g(i) = 2^{l(i)}$, we get

$$DIST_k(m, d) = \sum_{i \in I_k} (d_i - m_i) g(i).$$

For a given k, we now want to find an address m and destination address d under the permutation A such that $|DIST_k(m, d)|$ is maximized. Note that the address pair (m, d) will not necessarily have the maximal distance along any other dimension.

Define T_k to be the set of bit positions either in I_k or having a destination in I_k. Denoting set union by "+" and intersection by "$\cdot$", T_k may be expressed as

$$T_k = I_k + \bar{I}_k \cdot A^{-1}(I_k),$$

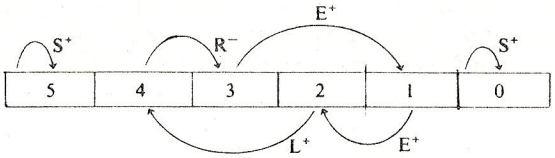

FIG. 1. Bit movement for Example 2.2

where $A^{-1}(I_k)$ denotes the set of bits with destination in I_k. That is,

$$A^{-1}(I_k) = \{i : |A_i| \in I_k\}.$$

Now, $DIST_k(m, d)$ may be rewritten as

$$\begin{aligned} DIST_k(m, d) &= \sum_{i \in I_k} (d_i - m_i) g(i) \\ &= \sum_{\substack{i \in T_k \\ |A_i| \in I_k}} d_{|A_i|} g(|A_i|) - \sum_{i \in I_k} m_i g(i) = \sum_{i \in T_k} \alpha_i, \end{aligned} \tag{2.1}$$

where

$$\alpha_i = \begin{cases} d_{|A_i|} g(|A_i|), & |A_i| \in I_k \text{ and } i \notin I_k, \\ -m_i g(i), & |A_i| \notin I_k \text{ and } i \in I_k, \\ d_{|A_i|} g(|A_i|) - m_i g(i), & |A_i| \in I_k \text{ and } i \in I_k. \end{cases}$$

The value of α_i depends on m_i and the destination of bit i. For example, if $i \in S_k^+$, then $A_i = i \in I_k$ and $d_{|A_i|} = m_i$. Hence, $\alpha_i = 0$. If $i \in S_k^-$, then again $|A_i| = i \in I_k$. Now, $d_{|A_i|} = \bar{m}_i$. So, when $m_i = 0$, $\alpha_i = g(i)$, and when $m_i = 1$, $\alpha_i = -g(i)$. As another example, consider the case $i \in L_k^+$. Now, $A_i \in I_k$ and $d_{|A_i|} = m_i$. So $\alpha_i = m_i(g(|A_i|) - g(i))$. Note that $g(|A_i|) > g(i)$ as $i \in L$. When $m_i = 0$, $\alpha_i = 0$. When $m_i = 1$, $\alpha_i = g(|A_i|) - g(i)$. Table II gives the value of α_i under the different possibilities for i, A_i, and m_i. α_i^0 and α_i^1 are, respectively, the values of α_i when $m_i = 0$ and 1. $h_k(i) = |\alpha_i^0 - \alpha_i^1|$.

From Table II we immediately obtain the following lemma.

LEMMA 2.1. *Let A be a permutation. Let d be the destination PE for PE(m). $DIST_k(m, d)$ is a maximum for the PE(m) with the following bit assignment:*

$$m_i = \begin{cases} 1 & \text{if } i \in L_k^+ + (E^+ \cdot A^{-1}(I_k)), \\ 0 & \text{otherwise.} \end{cases}$$

Let m be a PE such that m_i is as given by Lemma 2.1 for all $i \in T_k$. Let d be the destination PE for PE(m). Let $\bar{m}$ be the complement of m. The destination PE for PE($\bar{m}$) is easily seen to be $\bar{d}$. From Table II it follows that each α_i in $DIST_k(\bar{m}, \bar{d})$ is 0 or less. Consequently, to route from $\bar{m}$ or $\bar{d}$, negative routes are needed, while to route from m to d, positive routes are needed. Since these cannot be overlapped, to move m to d and $\bar{m}$ to $\bar{d}$, $DIST_k(m, d) + |DIST_k(\bar{m}, \bar{d})|$ unit-routes along the kth dimension are needed. This yields our lower bound $\beta_k(A)$ on the number of unit-routes needed along dimension k. We have

$$\begin{aligned} \beta_k(A) &= DIST_k(m, d) + |DIST_k(\bar{m}, \bar{d})| \\ &= DIST_k(m, d) - DIST_k(\bar{m}, \bar{d}) = \sum_{i \in T_k} h_k(i). \end{aligned} \tag{2.2}$$

Note that since $DIST_k(m, d) = -DIST_k(\bar{m}, \bar{d})$, we get

$$DIST_k(m, d) = \tfrac{1}{2}(\beta_k(A)).$$

Since Lemma 2.1 holds for every dimension (not necessarily for the same address m), the bound $\beta_k(A)$ is valid for all k. Observing that routes along one dimension cannot move

TABLE II. α_i VALUES FOR DIFFERENT BIT TYPES

	i type	α_i^0	α_i^1	$h_k(i)$
1	$i \in S_k^+$	0	0	0
2	$i \in S_k^-$	$g(i)$	$-g(i)$	$2g(i)$
3	$i \in L_k^+$	0	$g(\lvert A_i \rvert) - g(i)$	$g(\lvert A_i \rvert) - g(i)$
4	$i \in L_k^-$	$g(\lvert A_i \rvert)$	$-g(i)$	$g(\lvert A_i \rvert) + g(i)$
5	$i \in R_k^+$	0	$-(g(i) - g(\lvert A_i \rvert))$	$g(i) - g(\lvert A_i \rvert)$
6	$i \in R_k^-$	$g(\lvert A_i \rvert)$	$-g(i)$	$g(i) + g(\lvert A_i \rvert)$
7	$i \in E_k$	0	$-g(i)$	$g(i)$
8	$i \in E^+ \cdot A^{-1}(I_k)$	0	$g(\lvert A_i \rvert)$	$g(\lvert A_i \rvert)$
9	$i \in E^- \cdot A^{-1}(I_k)$	$g(\lvert A_i \rvert)$	0	$g(\lvert A_i \rvert)$

items along any other dimension, the lower bound $\beta(A)$ is obtained by summing the unit-routes needed along all dimensions. Thus,

$$\beta(A) = \sum_{k=0}^{q-1} \beta_k(A) = \sum_{k=0}^{q-1} \sum_{i \in T_k} h_k(i).$$

We wish to rewrite this into the more compact form $\sum_{i \in I} h(i)$ for an appropriate $h(\ )$. To obtain $h(\ )$ we consider the contribution of each bit i. We consider two cases:

Case 1. $i \notin E$. In this case $i \in I_j$ and $|A_i| \in I_j$ for some j. This bit contributes only to $\beta_j(A)$ and

$$h(i) = h_j(i) = \begin{cases} |g(|A_i|) - g(i)| & \text{if } i \in (L_j^+ + R_j^+ + S_j^+), \\ g(|A_i|) + g(i) & \text{if } i \in (L_j^- + R_j^- + S_j^-). \end{cases}$$

Case 2. $i \in E$. Now, $i \in I_j$ and $|A_i| \in I_r$, $j \neq r$. So i now contributes to both $\beta_j(A)$ and $\beta_r(A)$. From Table II it is clear that $h_j(i) = g(i)$ and $h_r(i) = g(|A_i|)$. Hence,

$$h(i) = g(|A_i|) + g(i) \qquad \text{for } i \in E.$$

Combining these two cases together, we get the following theorem.

THEOREM 2.1. *To perform the permutation A on an MCC, at least $\beta(A)$ unit-routes are needed. $\beta(A)$ is given by the equation*

$$\beta(A) = \sum_{i \in I} h(i), \tag{2.3}$$

where

$$h(i) = \begin{cases} |g(|A_i|) - g(i)| & \text{if } i \in (L^+ + R^+ + S^+), \\ g(|A_i|) + g(i) & \text{otherwise.} \end{cases}$$

While (2.3) is an elegant form for β, it is not very convenient for hand computation of β. We now present two equivalent forms for β. The first of these is used in Sections 3 through 5 to obtain the routing algorithm. The second form is convenient for hand calculation of β and is used in the examples at the end of this section.

The form of β used to arrive at the routing algorithm is obtained from (2.3) by separating out the contributions of bits that get complemented (i.e., $A_i < 0$). Table III gives the values of $h(i)$ for each bit type when complemented (i.e., $A_i < 0$) and when not complemented (i.e., $A_i \geq 0$). If we look at the column for uncomplemented bits, we see that their contribution to β is given by $f(i, |A_i|)$, where $f(i, j)$ is as defined below:

$$f(i, j) = \begin{cases} |g(i) - g(j)| & \text{if } u(i) = u(j), \\ g(i) + g(j) & \text{otherwise.} \end{cases}$$

The last column in the table gives the additional contribution of a complemented bit. From this column and the definition of f, we immediately obtain

$$\beta(A) = \sum_{i \in I} f(i, |A_i|) + 2\left(\sum_{i \in (S^- + L^-)} g(i) + \sum_{i \in R^-} g(|A_i|) \right). \tag{2.4}$$

TABLE III. CONTRIBUTION OF BIT i TO $\beta(A)$

Bit type	$A_i \geq 0$	$A_i < 0$	Difference
E	$g(A_i) + g(i)$	$g(\|A_i\|) + g(i)$	0
S	0	$2g(i)$	$2g(i)$
L	$g(A_i) - g(i)$	$g(\|A_i\|) + g(i)$	$2g(i)$
R	$g(i) - g(A_i)$	$g(i) + g(\|A_i\|)$	$2g(\|A_i\|)$

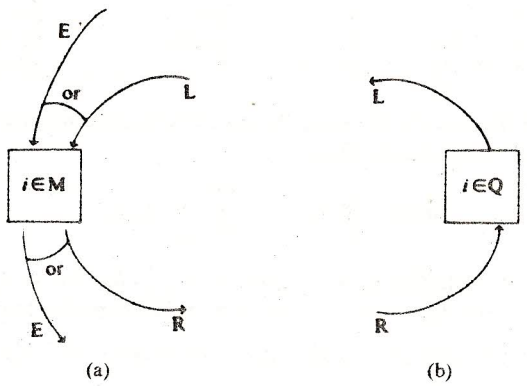

FIG. 2. Illustration of sets M and Q: (a) a bit in M; (b) a bit in Q

The form of β suitable for hand calculation is easily obtained from (2.4). This form uses the following subsets of I (see Figure 2):

(i) $M = (R + E)\cdot A(L + E)$,
(ii) $Q = L\cdot A(R)$.

In words, M is the set of all bit positions of type R or E that will be replaced by bits from positions of type L or E. Q is the set of all bit positions of type L for which the incoming bit is from a position of type R. In terms of M, Q, S^-, L^-, and R^-, β may be written as

$$\beta(A) = 2\left(\sum_{i\in M} g(i) - \sum_{i\in Q} g(i)\right) + 2\left(\sum_{i\in(S^-+L^-)} g(i) + \sum_{i\in R^-} g(|A_i|)\right). \tag{2.5}$$

The correctness of (2.5) is proved in the following lemma.

LEMMA 2.2. *Equation* (2.5) *is correct.*

PROOF. We simply have to show that

$$\sum_{i\in I} f(i, |A_i|) = 2\left(\sum_{i\in M} g(i) - \sum_{i\in Q} g(i)\right).$$

Define $M' = (R + E)\cdot A(R)$ and $Q' = L\cdot A(L + E)$. It is easy to see that $M + M' = R + E$ and $Q + Q' = L$. Hence, $I = M + M' + Q + Q' + S$. From the definition of f and the fact that A is a permutation of the bit indices, it follows that $\sum_{i\in I} f(i, |A_i|)$ will contain exactly two occurrences of $g(i)$ for each i. When $i \in Q$ these two terms come from $f(i, |A_i|)$ and $f(j, |A_j|)$ where $i = |A_j|$. Since $i \in L$, $f(i, |A_i|) = g(|A_i|) - g(i)$. Also, $j \in R$ (as $i \in Q = L\cdot A(R)$). Hence, $f(j, |A_j|) = g(j) - g(|A_j|) = g(j) - g(i)$. So both occurrences of $g(i)$ are negative. Using a similar argument, one may show that when $i \in M$, both occurrences of $g(i)$ are positive; and when $i \in (S + M' + Q')$, one occurrence of $g(i)$ is positive while the other is negative. This completes the proof. □

Example 2.3. Consider an $n \times n$ MCC. Let $n^2 = 2^p$. For the perfect shuffle permutation, $M = \{p - 1, p/2 - 1\}$, $Q = S^- = L^- = R^- = \emptyset$. $l(p - 1) = l(p/2 - 1) = p/2 - 1$ and $g(p - 1) = g(p/2 - 1) = 2^{(p/2-1)}$. From (2.5) we obtain

$$\beta(A) = 2(g(p - 1) + g(p/2 - 1)) = 2 * 2^{p/2} = 2n.$$

TABLE IV. THE LOWER BOUND FOR SOME COMMON PERMUTATIONS ON AN $n \times n$ MCC

Permutation	M, Q, S^-, L^-, R^-	β(A)
Matrix transpose	$M = \{p-1, p-2, \ldots, 0\}$	$4(n-1)$
Bit reversal	$M = \{p-1, p-2, \ldots, 0\}$	$4(n-1)$
Vector reversal	$S^- = \{p-1, p-2, \ldots, 0\}$	$4(n-1)$
Perfect shuffle	$M = \{p-1, p/2-1\}$	$2n$
Bit shuffle	$M_0 = E_0 = \{p/2-1, p/2-2, \ldots, \lceil p/4 \rceil\}$ $M_1 = A(E_0) = \{p-2, p-4, \ldots, 2\lceil p/4 \rceil\}$	$\frac{8n - 6*2^{\lceil p/4 \rceil} - 2*2^{2\lceil p/4 \rceil - p/2}}{3}$

For vector reversal, $S^- = \{p-1, p-2, \ldots, 0\}$ and $M = Q = L^- = R^- = \varnothing$. Hence,

$$\beta(A) = 2\sum_{i=0}^{p-1} g(i) = 4\sum_{i=0}^{p/2-1} 2^i = 4(n-1).$$

Table IV gives the values of the sets M, Q, S^-, L^-, and R^- for many of the permutations of Table I. Only nonempty sets are explicitly specified. The last column gives the value of β. In the entry for bit shuffle, M is partitioned into $M_0 = M \cdot I_0$ and $M_1 = M \cdot I_1$.

Before concluding this section, we derive an upper bound on $\beta(A)$ independent of A.

THEOREM 2.2. $\beta(A) \leq 2\sum_{k=0}^{q-1}(n_k - 1)$.

PROOF. From (2.3) it follows that $\beta(A)$ is maximized when $L^+ + R^+ + S^+ = \varnothing$. So,

$$\beta(A) \leq \sum_{i=0}^{p-1} (g(|A_i|) + g(i)) = 2\sum_{i=0}^{p-1} g(i) = 2\sum_{k=0}^{q-1}(n_k - 1).$$

For an $n \times n$ mesh, this implies $\beta(A) \leq 4(n-1)$. □

It should be easy to see that the lower bound obtained in this section depends only on the restriction that data can be routed in only one direction during each unit-route. Relaxing the model to allow arbitrary PE selectivity during a route or arbitrary arithmetic capability in a PE does not affect the bound. However, permitting multidirectional movement of data during a route will lower the bound.

3. *Preliminaries to the Algorithm*

In this section two routing algorithms which are building blocks of our overall permutation routing algorithm are developed. The first of these handles bit complementation and the other handles bit interchanges.

3.1 BIT COMPLEMENTATION. Let $C_i = [p-1, p-2, \ldots, i+1, -i, i-1, \ldots, 0]$ be a permutation which differs from the identity permutation only in that the ith bit is complemented. The permutation C_i is easily performed by first routing the data in all PE's whose address has a 1 in bit position i to the corresponding PE with a 0 in bit position i. This requires a negative route of length $g(i)$ along dimenson $u(i)$. Next, data from PE's with the ith bit equal to 0 are routed to their corresponding PE's using a positive route of length $g(i)$ along dimension $u(i)$. This is stated more formally in procedure COMPLEMENT. It should be evident that the number of unit-routes $R(C_i)$ used by COMPLEMENT is $R(C_i) = \beta(C_i) = 2g(i)$.

```
procedure COMPLEMENT(i)
  //Perform the permutation C_i. Data to be permuted//
  //is in the R_s registers//
  R_r ← R_s (i) //transfer to routing registers for PE's with bit i = 1//
  ROUTE(u(i), −g(i)) //move to corresponding PE's with bit i = 0//
  R_r ↔ R_s (−i) //only PE's with bit i = 0 are enabled//
  ROUTE(u(i), g(i))
  R_s ← R_r (i)
end COMPLEMENT
```

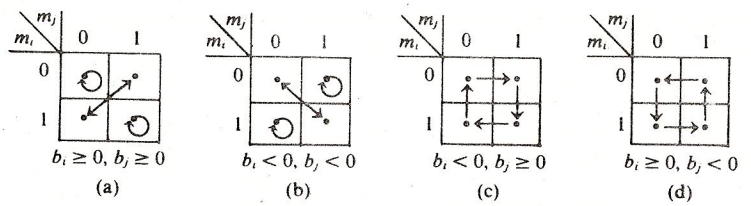

FIG. 3. Data movement for bit interchange B_{ij}. Each quadrant represents one-fourth of the PE's

3.2 BIT INTERCHANGE. Let B_{ij} denote the permutation $[b_{p-1}, b_{p-2}, \ldots, b_0]$ in which $b_i = \pm j$, $b_j = \pm i$, and $b_r = r$ for $r \notin \{i, j\}$. In addition, if $u(i) = u(j)$, then $b_i \geq 0$ and $b_j \geq 0$. In words, B_{ij} is a permutation in which only bits i and j get interchanged (with possible bit complementation). Bits may be complemented only when i and j are in different dimensions. Figure 3 shows the data exchanges corresponding to this permutation for all sign combinations of b_i and b_j.

The permutation B_{ij} can be performed in a very straightforward manner using only $\beta(B_{ij})$ unit-routes. Our algorithm (procedure INTERCHANGE) considers the following cases:

(i) $u(i) = u(j)$. In this case the data movement is as in Figure 3(a). Data in a PE with bit $i = 0$ and bit $j = 1$ get interchanged with those in the corresponding PE with bit $i = 1$ and bit $j = 0$. Note that PE's with bits i and j the same are not affected. This interchange is carried out by first moving the data from the PE's with bit $i = 0$ and bit $j = 1$ (i.e., PE selectivity $(-i, j)$) to the PE's with bit $i = 1$ and bit $j = 0$ (i.e., PE selectivity $(i, -j)$) (lines 3 and 4). This requires $|g(i) - g(j)|$ unit-routes along dimension $u(i)$. Then, using an additional $|g(i) - g(j)|$ unit-routes along dimension $u(i)$, data from PE's $(i, -j)$ are moved to PE's $(-i, j)$ (lines 6 and 7).

(ii-a) $u(i) \neq u(j)$ and (b_i and b_j positive). (See Figure 3(a).) Once again there is no bit complementation, and the interchange is between PE's $(-i, j)$ and $(i, -j)$. In lines 12 and 13 data from PE's $(-i, j)$ are moved to the PE's $(i, -j)$ by using $-g(j)$ unit-routes along dimension $u(j)$ and then $g(i)$ unit-routes along dimension $u(i)$. In lines 14 and 15 the move from $(i, -j)$ to $(-i, j)$ is accomplished. The total number of unit-routes is $2(g(i) + g(j))$.

(ii-b) $u(i) \neq u(j)$ and (b_i and b_j negative). (See Figure 3(b).) In this case data in PE's $(-i, -j)$ are to be interchanged with data in PE's (i, j). Note that data in PE's $(-i, j)$ and $(i, -j)$ are not affected by the permutation. The permutation B_{ij} is carried out in a manner similar to that of (ii-a). The code is given in lines 17–21. The total number of unit-routes is $2(g(i) + g(j))$.

(ii-c) $u(i) \neq u(j)$ and ($b_i < 0$ but $b_j \geq 0$). (See Figure 3(c).) Now, the data movement may be described by the cycle: Move data from PE's $(-i, -j)$ to PE's $(-i, j)$; from $(-i, j)$ to (i, j); from (i, j) to $(i, -j)$; and from $(i, -j)$ to $(-i, -j)$. The code for this routing is given in lines 23–31. The total number of unit-routes is again $2(g(i) + g(j))$.

(ii-d) $u(i) \neq u(j)$ and ($b_i \geq 0$ and $b_j < 0$). (See Figure 3(d).) The data movement here follows a cycle similar to that for case (ii-c). Data from PE's $(-i, -j)$ are to be moved to PE's $(i, -j)$; from $(i, -j)$ to (i, j); from (i, j) to $(-i, j)$; and from $(-i, j)$ to $(-i, -j)$. The code for this is similar to that of lines 23–31 and is omitted. The number of unit-routes needed for this case is also $2(g(i) + g(j))$.

The length of procedure INTERCHANGE can be reduced somewhat by combining cases (ii-b), (ii-c), and (ii-d). We have chosen not to do this in our presentation as the longer version more clearly describes the data movement. As remarked in the case discussion, the number of unit-routes, $R(B_{ij})$, used to perform B_{ij} equals $2f(i, j) = \beta(B_{ij})$ (recall that $L^- = R^- = S^- = \emptyset$ for B_{ij}).

```
line  procedure INTERCHANGE(i, j, b_i, b_j)
      //Perform the permutation B_ij//
 1    case
 2      :u(i) = u(j): //This implies b_i ≥ 0 and b_j ≥ 0//
 3        R_r ← R_s (−i, j)
 4        ROUTE(u(i), g(i) − g(j))
 5        R_r ↔ R_s (i, −j)
 6        ROUTE(u(i), g(j) − g(i))
 7        R_s ← R_r (−i, j)
 8      :u(i) ≠ u(j):
 9        case
10          :b_i ≥ 0 and b_j ≥ 0: //Figure 3(a)//
11            R_r ← R_s (−i, j)
12            ROUTE(u(j), −g(j)); ROUTE(u(i), g(i))
13            R_r ↔ R_s (i, −j)
14            ROUTE(u(i), −g(i)); ROUTE(u(j), g(j))
15            R_s ← R_r (−i, j)
16          :b_i < 0 and b_j < 0: //Figure 3(b)//
17            R_r ← R_s (−i, −j)
18            ROUTE(u(i), g(i)); ROUTE(u(j), g(j))
19            R_r ↔ R_s (i, j)
20            ROUTE(u(i), −g(i)); ROUTE(u(j), −g(j))
21            R_s ← R_r (−i, −j)
22          :b_i < 0 and b_j ≥ 0: //Figure 3(c)//
23            R_r ← R_s (−i, −j)
24            ROUTE(u(j), g(j))
25            R_r ↔ R_s (−i, j)
26            ROUTE(u(i), g(i))
27            R_r ↔ R_s (i, j)
28            ROUTE(u(j), −g(j))
29            R_r ↔ R_s (i, −j)
30            ROUTE(u(i), −g(i))
31            R_s ← R_r (−i, −j)
32          :else: //Figure 3(d). The code for this is similar to that for b_i < 0 and b_j ≥ 0//
33        endcase
34    endcase
35  end INTERCHANGE
```

4. *Two-Dimensional MCC's*

Any permutation A for a two-dimensional MCC can be performed in $\beta(A)$ unit-routes using procedures COMPLEMENT and INTERCHANGE of Section 3. First, however, the permutation A must be factored so that it is the product of permutations of type C_i and B_{ij}. (If X, Y, and Z are three permutations, then $X = Y \cdot Z$ iff permutation X is equivalent to first performing permutation Y and then Z. Y and Z are *factors* of X; X is the *product* of Y and Z.) The factorization of A into factors of type C_i and B_{ij} is to be done in such a way that the sum of the unit-routes needed for each factor does not exceed $\beta(A)$.

We first show how to factor out bit complementation for bits of type S^-, L^-, and R^-. Let i be a complemented bit. Then A may be factored as $A = C_iA'$ where A' differs from A only in that $A'_i = |A_i|$. Such a factorization will be called *precomplementation*. The permutation A may be performed using COMPLEMENT for C_i and then performing A'. The number of unit-routes R(A) used is $R(C_i) + R(A') = 2g(i) + R(A')$. From Table III one readily obtains the following relationship between $\beta(A)$ and $\beta(A')$:

$$\beta(A') = \begin{cases} \beta(A) - 2g(i) & \text{if } i \in S^- + L^-, \\ \beta(A) - 2g(|A_i|) & \text{if } i \in R^-, \\ \beta(A) & \text{if } i \in E^-. \end{cases} \tag{4.1}$$

From (4.1) and the knowledge that $\beta(C_i) = 2g(i)$, it follows that if precomplementation is used for $i \in (E^- + R^-)$, then $\beta(C_i) + \beta(A') > \beta(A)$; and if it is used for $i \in (S^- + L^-)$, then $\beta(C_i) + \beta(A') = \beta(A)$. Since we are attempting to obtain an algorithm using only $\beta(A)$ unit-routes, precomplementation cannot be used for $i \in (E^- + R^-)$.

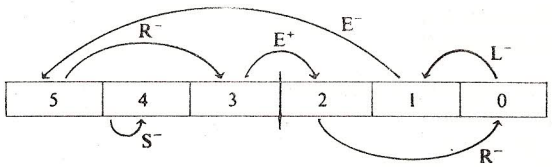

FIG. 4. Bit movement for Example 4.1

$i \in R^-$ can be handled using postcomplementation. Note that every permutation A in which bit i is complemented can also be factored as $A = A'C_{|A_i|}$ where A' is as before. Such a factorization is called *postcomplementation*. Equation (4.1) still holds for A' and we see that $\beta(A') + \beta(C_{|A_i|}) = \beta(A)$ for $i \in R^-$.

$i \in E^-$ cannot be factored out using either post- or precomplementation and yet meet the requirement on β. This does not pose a problem at this point as complementation of such bits is permitted in B_{ij}-type permutations. By repeatedly applying pre- and postcomplementation for bits in $S^- + L^- + R^-$, A may be factored into the product form below:

$$A = \prod_{i \in (S^- + L^-)} C_i \cdot A'' \cdot \prod_{i \in R^-} C_{|A_i|}.$$

This factorization has the property that

$$\beta(A) = \sum_{i \in (S^- + L^-)} \beta(C_i) + \beta(A'') + \sum_{i \in R^-} \beta(C_{|A_i|}).$$

Example 4.1. Let $A = [-3, -4, +2, -0, -5, -1]$. For an 8×8 MCC we get $S^- = \{4\}$, $L^- = \{0\}$, $R^- = \{5, 2\}$, $E^- = \{1\}$, and $E^+ = \{3\}$ (see Figure 4). Thus, A is factored into the product

$$A = C_4C_0[3, 4, 2, 0, -5, 1]C_3C_0.$$

Our algorithm will perform A by first performing C_4 and C_0 using COMPLEMENT; then the permutation $[3, 4, 2, 0, -5, 1]$ will be performed (in a manner yet to be specified); and finally, C_3 and C_0 will be performed using COMPLEMENT.

Once complemented bits in $(S^- + L^- + R^-)$ have been factored out, we are left with a permutation in which all complemented bits are from E^-. We now discuss how a permutation A with this property can be performed in $\beta(A)$ unit-routes. We shall factor A into factors of type B_{ij}. This factorization is a little more involved than that used earlier. In order to develop the factorization, we need to introduce some terminology. This is done below in Definitions 4.1 and 4.2.

Definition 4.1. Let A be any permutation. Bits i and j are *interchangeable* in A iff there exists a permutation B_{ij} (as defined in Section 3) such that if $A = B_{ij}A'$ then $\beta(A) = \beta(B_{ij}) + \beta(A')$.

Definition 4.2. For any i, j, $i \neq j$, the $\langle i, j \rangle$ *path* denoted $P(i, j)$ is the sequence of bit indices defined as follows:

$$P(i, j) = \begin{cases} (i, i-1, \ldots, j) & \text{if } u(i) = u(j) \text{ and } i > j, \\ (i, i+1, \ldots, j) & \text{if } u(i) = u(j) \text{ and } i < j, \\ (i, i-1, \ldots, i-l(i), j-l(j), \ldots, j-1, j) & \text{if } u(i) \neq u(j). \end{cases}$$

Note that when $u(i) \neq u(j)$, $P(i, j)$ consists of a descending bit sequence in $I_{u(i)}$ followed by an ascending bit sequence in $I_{u(j)}$ (see Figure 5).

LEMMA 4.1. *Let i, j, and k be distinct bit indices. $f(i, j) = f(i, k) + f(k, j)$ iff k is on the $\langle i, j \rangle$ path.*

PROOF. We prove only the "if" part. The "only if" part is not used in the later development and is left to the reader.

Case 1. $u(i) = u(j)$. k must be in $I_{u(i)}$ and either $i < k < j$ or $i > k > j$. Without

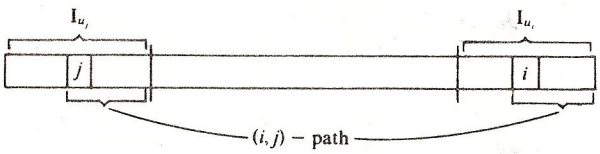

FIG. 5. The (i, j)-path when $u_i \neq u_j$

loss of generality, we may assume $i > k > j$. Hence, $f(i, j) = g(i) - g(j)$, $f(i, k) = g(i) - g(k)$, and $f(k, j) = g(k) - g(j)$.

Case 2. $u(i) \neq u(j)$. Now, either ($u(i) = u(k)$ and $i > k$) or ($u(j) = u(k)$ and $j > k$). If the latter is the case, then $f(i, j) = g(i) + g(j)$; $f(i, k) = g(i) + g(k)$; and $f(k, j) = g(j) - g(k)$. If the former is the case, then $f(i, j) = g(i) + g(j)$; $f(i, k) = g(i) - g(k)$; and $f(k, j) = g(k) + g(j)$. □

THEOREM 4.1 *Let A be any permutation in which all complemented bits are of type E^-. Bits i and j are interchangeable iff i is on the $\langle j, |A_j| \rangle$ path and j is on the $\langle i, |A_i| \rangle$ path.*

PROOF. Once again we prove only the "if" part. The "only if" part will not be used by us and its proof is left to the reader.

Define the interchange permutation $B_{ij} = [b_{p-1}, \ldots, b_0]$ with $b_r = r$ for $r \notin \{i, j\}$ and b_i and b_j as below:

$$b_i = \begin{cases} j & \text{if } A_i \geq 0 \text{ or } u(i) = u(j), \\ -j & \text{otherwise;} \end{cases}$$

$$b_j = \begin{cases} i & \text{if } A_j \geq 0 \text{ or } u(i) = u(j), \\ -i & \text{otherwise.} \end{cases}$$

Thus, bits i or j are complemented by B_{ij} only when i and j are in different partitions $I_{u(i)}$ and $I_{u(j)}$. It is easy to see that A may be written as $A = B_{ij}A'$ where $A'_r = A_r$ for $r \notin \{i, j\}$ and

$$A'_i = \begin{cases} A_j & \text{if } b_j = i, \\ |A_j| & \text{if } b_j = -i; \end{cases}$$

$$A'_j = \begin{cases} A_i & \text{if } b_i = j, \\ |A_i| & \text{if } b_i = -j. \end{cases}$$

From the definition of A, B_{ij}, and A′, it follows that all bits in A′ that are complemented are in E. Hence, from (2.4) we obtain

$$\beta(A') = \sum_{r \in I} f(r, |A'_r|). \tag{4.2}$$

Using (2.4), Lemma 4.1, $\beta(B_{ij}) = 2f(i, j)$, and (4.2), we obtain

$$\begin{aligned} \beta(A) &= \sum_{r \in I} f(r, |A_r|) \\ &= f(i, |A_i|) + f(j, |A_j|) + \sum_{r \in I - \{i,j\}} f(r, |A_r|) \\ &= f(i, j) + f(j, |A_i|) + f(j, i) + f(i, |A_j|) + \sum_{r \in I - \{i,j\}} f(r, |A'_r|) \\ &= 2f(i, j) + f(j, |A'_j|) + f(i, |A'_i|) + \sum_{r \in I - \{i,j\}} f(r, |A'_r|) \\ &= \beta(B_{ij}) + \beta(A'). \end{aligned}$$

Hence, i and j are interchangeable in A. □

The preceding theorem tells us how a permutation A should be factored into products of the type B_{ij}. The next theorem shows that every permutation A for a two-dimensional

TABLE V

$A_7 \cdots A_0$	*lmax*	*i*	$A'_7 \cdots A'_0$
32107654	3	4	32170654
32170654	4	5	32710654
32710654	5	6	37210654
37210654	6	7	73210654
73210654	2	3	73216054
73216054	3	4	73261054
73261054	4	5	73621054
73621054	5	6	76321054
⋮	⋮	⋮	⋮

MCC can be factored as the product

$$A = \prod_{i \in (S^- + L^-)} C_i \cdot \prod B_{ij} \cdot \prod_{i \in R^-} C_{|A_i|}, \tag{4.3}$$

where each B_{ij} is an interchange on an interchangeable pair (i, j).

THEOREM 4.2. *Let A be a permutation for a two-dimensional MCC. If there exists an r such that $|A_r| \neq r$ then A contains an interchangeable pair of bits.*

PROOF. We consider two cases:

(i) $L \neq \emptyset$. In this case $L_k \neq \emptyset$ for some $k \in \{0, 1\}$. Let $lmax = \max\{L_k\}$ and $i = \min\{j : j \in (E_k + R_k)$ and $j > lmax\}$. Note that since $L_k \neq \emptyset$, there must exist at least one j with the desired property. So, i exists. One may verify that bits $lmax$ and i are interchangeable.

(ii) $L = \emptyset$. Since $|A_r| \neq r$ for some r, the sets E_0 and E_1 cannot be empty. Let $e_0 = \min(E_0)$ and $e_1 = \min(E_1)$. Since $L = \emptyset$, e_1 must be on the $\langle e_0, |A_{e_0}| \rangle$ path, and e_0 on the $\langle e_1, |A_{e_1}| \rangle$ path. Hence, e_0 and e_1 are interchangeable. □

The preceding theorem together with our earlier discussion on pre- and postcomplementation guarantees that every permutation for a two-dimensional MCC can be factored as in (4.3). Since procedures COMPLEMENT and INTERCHANGE perform their permutations in $\beta(C_i)$ and $\beta(B_{ij})$ unit-routes, respectively, it follows that A can be performed in $\beta(A)$ unit-routes.

Example 4.2. Consider the permutation $A = [p/2 - 1, \ldots, 0, p - 1, \ldots, p/2]$ for a $1 \times n$ MCC. This defines the transpose of a square matrix stored in row-major order on a *one*-dimensional MCC. $S = E = \emptyset$. $L_0 = \{p/2 - 1, \ldots, 0\}$ and $R_0 = \{p - 1, \ldots, p/2\}$. All bit exchanges fall into case (i) of the above theorem. Let us work out the bit exchange sequence with $p = 8$. We have $A_7A_6 \cdots A_0 = 32107654$. $lmax = 3$ and $i = 4$. Following the interchange, $A'_7A'_6 \cdots A'_0 = 32170654$. Now, $lmax = 4$ and $i = 5$. $A'_7A'_6 \cdots A'_0$ is now 32710654. The sequence of $lmax$, i, and A' values is given in Table V. The bit exchange sequence is (3, 4); (4, 5); (5, 6); (6, 7); (2, 3); (3, 4); (4, 5); (5, 6); .... It should be clear that for general p, $O(p^2)$ bit exchanges will be needed. Hence, performing the permutation A will require $O(p^2)$ long-routes, $O(p^2)$ register transfers, and $\beta(A)$ unit-routes.

While all permutations can be performed with $\beta(A)$ unit-routes using pre- and postcomplementation and the bit interchange factorization as suggested by Theorem 4.2, this process may require $O(p^2)$ bit interchanges (Example 4.2). This in turn would require $O(p^2)$ long-routes and register interchanges. The number of bit interchanges can be reduced to $O(p)$ by factoring A according to the cycles contained in A. Define $D^0(i) = i$, $D^1(i) = |A_i|$, and $D^r(i) = |A_{D^{r-1}(i)}|$. i and j are in the same *cycle* of A iff $i = D^r(j)$ for some r, $r \leq p - 1$.

Example 4.3. Let $A = [5, 0, -4, 1, -2, -3]$. $D^0(4) = 4$; $D^1(4) = 0$; $D^2(4) = D^1(0) = 3$; $D^3(4) = D^2(0) = D^1(3) = 4$. Thus, bits (4, 0, 3) form a cycle in A. The other cycles are (5) and (1, 2).

It is easy to see that the cycles of any permutation A are disjoint.

THEOREM 4.3. *If A is a permutation composed of k disjoint cycles then performing A requires at least $p - k$ bit interchanges.*

PROOF. Trivial. □

To perform A using the fewest number (i.e., $p - k$) of bit interchanges, we require that bit interchanges be performed only between interchangeable bits in the *same* cycle. Following along the lines of Theorem 4.2 one can show that every cycle containing at least two bits contains an interchangeable pair.

Procedure PERMUTE(A, p) uses COMPLEMENT and INTERCHANGE to perform the permutation A on a two-dimensional MCC with $N = 2^p$ processors. Lines 1–3 eliminate all bits in $S^- + L^-$. In lines 4–24 permutation cycles are followed until an interchangeable pair of bits is found. The loop of lines 5–23 forces the algorithm to stay in the cycle for j until this cycle contains only one bit. If a cycle contains no bit in L then the interchangeable pair is found exactly as in Theorem 4.2. This is done in lines 7–12. If the cycle contains a bit in L then the first interchangeable pair found (line 17) is used. Note that $u(i) = u(lmax)$. The reader may verify that the *signs* (but not the values) of the third and fourth parameters in the calls to INTERCHANGE (lines 11 and 18) adhere to that given by Theorem 4.1 for b_i and b_j. The values of these parameters are ignored by the procedure INTERCHANGE.

It is easily seen that PERMUTE takes $O(p^2) = O(\log^2 N)$ time to determine the routing sequence. The number of unit-routes is $\beta(A)$, and the number of long-routes and register transfers are both $O(p)$. For an $n \times n$ MCC, it follows from Theorem 2.2 that PERMUTE uses at most $4(n - 1)$ unit-routes for any permutation.

ALGORITHM 4.1

```
line  procedure PERMUTE(A, p)
        //Perform A on a two-dimensional MCC//
  1     for j ← 0 to p − 1 do //precomplementation//
  2       if j ∈ (S⁻ + L⁻) then [COMPLEMENT(j); A_j ← −A_j]
  3     end
  4     for j ← 0 to p − 1 do //bit interchanges along cycles//
  5       while |A_j| ≠ j do
  6         i ← j; e_0 ← e_1 ← p; success ← false
  7         if i ∉ L then [repeat
  8                          e_u(i) ← min(i, e_u(i))
  9                          i ← |A_i|;
 10                        until i ∈ L or i = j];
 11         if i ∉ L then [INTERCHANGE(e_0, e_1, A_e0, A_e1);
 12                        t ← |A_e0|; A_e0 ← |A_e1|; A_e1 ← t]
 13                  else
 14                [repeat
 15                  if i ∈ L then lmax ← i
 16                           else
 17                    [if |A_i| ≤ lmax or i ∈ E then
 18                       [INTERCHANGE(i, lmax, |A_i|, |A_lmax|)
 19                        t ← A_i; A_i ← A_lmax; A_lmax ← t;
 20                        success ← true]];
 21                  i ← |A_i|
 22                until success]
 23       end
 24     end
 25     for j ← 0 to p − 1 do //postcomplementation//
 26       if A_j = −j then COMPLEMENT(j)
 27     end
 28   end PERMUTE
```

Example 4.4. Let us apply the algorithm to the perfect shuffle permutation, A = $[0, p - 1, \ldots, 1]$, performed on an $n \times n$ array. The algorithm yields the following sequence

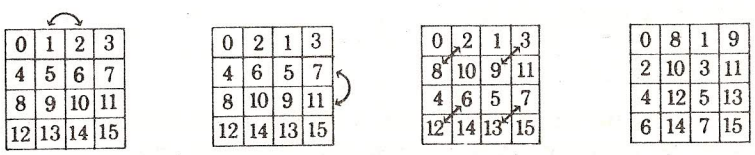

FIG. 6. Perfect shuffle

of INTERCHANGE's:

$$(p/2 - 1, p/2 - 2), (p/2 - 2, p/2 - 3), \ldots, (1, 0),$$
$$(p - 1, p - 2), (p - 2, p - 3), \ldots, (p/2 + 1, p/2), (p/2, 0).$$

Let $N = 4 \times 4$; thus $p = 4$. The sequence of bit interchanges becomes

$$(1, 0); (3, 2); (2, 0).$$

The PE-exchanges corresponding to this are shown in Figure 6.

Example 4.5. Consider the bit shuffle permutation on a 16×16 array. Then $A = [7, 5, 3, 1, 6, 4, 2, 0]$. The algorithm gives the following sequence of bit interchanges:

$$(2, 1); (1, 4); (3, 5); (6, 5).$$

5. *q-Dimensional MCC's* ($q > 2$)

When $q > 2$, some permutations may not contain interchangeable pairs. As an example, consider the case $q = 3$ and $A = [A_{p-1}, \ldots, A_0]$ where $A_r = r$, $r \notin \{i, j, k\}$, $A_i = j$, $A_j = k$, and $A_k = i$ for some $i \in I_0$, $j \in I_1$, and $k \in I_2$. $\beta(A) = 2(g(i) + g(j) + g(k))$. The factorization $A = B_{ij}B_{ik}$ has $\beta(B_{ij}) + \beta(B_{ik}) = 4g(i) + 2(g(j) + g(k)) > \beta(A)$. All other factorizations also have a β value in excess of $\beta(A)$.

In order to perform permutations in $\beta(A)$ unit-routes, it is necessary to handle bit interchanges in a different manner when $u(i) \neq u(j)$. The interchange of bits i and j is carried out in three stages: FOLD(i); EXCHANGE(i, j), and UNFOLD(i):

FOLD(i). During FOLD(i) the data in all PE's with ith bit 1 are moved to the corresponding PE's with ith bit 0 and transferred to register R_t. Procedure FOLD is a formal description.

EXCHANGE(i, j). When EXCHANGE is invoked, all relevent data are lying in the R_s and R_t registers of PE's with bit $i = 0$. Let us denote by e_0, e_1, e_2, and e_3 the data originally in PE's $(-i, -j)$, $(-i, j)$, $(i, -j)$, and (i, j), respectively (see Figure 7(a)). Thus, e_0, e_1, e_2, and e_3 data are in registers R_s $(-i, -j)$, R_s $(-i, j)$, R_t $(-i, -j)$, and R_t $(-i, j)$, respectively, when EXCHANGE is invoked (Figure 7 (b)).

EXCHANGE interchanges the data so that if, following the exchange, data in R_t $(-i)$ are moved to R_s (i), then the bit permutation B_{ij} will have been performed. There are four cases to be considered:

(i) $A_i \geq 0$ and $A_j \geq 0$. In this case we want to end up with e_0, e_1, e_2, and e_3 data in R_s $(-i, -j)$, R_t $(-i, -j)$, R_s $(-i, j)$, and R_t $(-i, j)$, respectively (Figure 7 (c)). This may be accomplished by moving data from R_s (j) to R_t $(-j)$ and from R_t $(-j)$ to R_s (j). This is done in lines 2–6 of procedure EXCHANGE.

(ii) $A_i < 0$ and $A_j < 0$. The final data arrangement required is shown in Figure 7(d). This may be accomplished by first interchanging the data in R_s and R_t to get the configuration of Figure 7(e). Next, e_0 and e_3 may be exchanged using the same steps as in case (i). Finally an interchange between R_s and R_t is needed. One may verify that lines 1–7 accomplish the exchange.

(iii) $A_i < 0$ and $A_j \geq 0$. Figure 7(f) shows the final data arrangement needed (see also Figure 3(c)). One may verify that EXCHANGE obtains this arrangement.

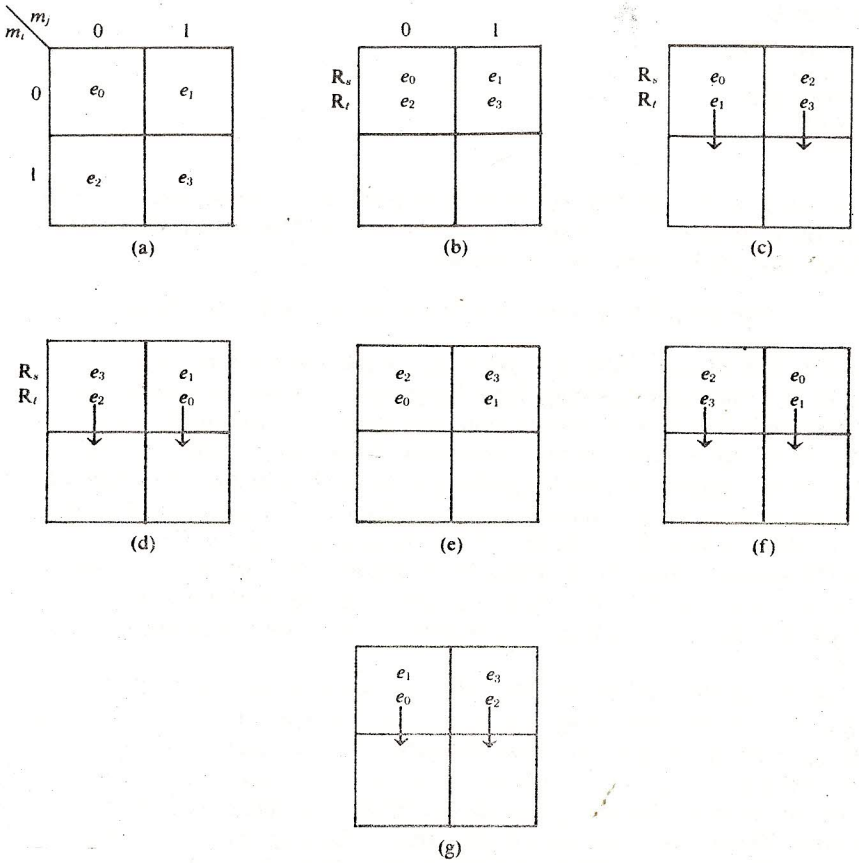

FIG. 7. Bit movement for EXCHANGE

(iv) $A_i \geq 0$ and $A_j < 0$. This time the data arrangement needed is as in Figure 7(g) (see also Figure 3(d)). One may verify that EXCHANGE obtains this arrangement.

UNFOLD(i). This procedure moves data from $R_t(-i)$ to $R_s(i)$. Hence, it accomplishes the move indicated by the arrows in Figure 7(c), (d), (f), and (g).

ALGORITHM 5.1

```
procedure FOLD(i)
  R_r ← R_s(i)
  ROUTE(u(i), −g(i))
  R_t ← R_r(−i)
end FOLD
```

ALGORITHM 5.2

```
line   procedure EXCHANGE(i, j)
 1       if A_i < 0 then R_s ↔ R_t(−i)
 2       R_r ← R_s(j)
 3       ROUTE(u(j), −g(j))
 4       R_r ↔ R_t(−j)
 5       ROUTE(u(j), g(j))
 6       R_s ← R_r(j)
 7       if A_j < 0 then R_s ↔ R_t(−i)
 8       T ← |A_i|; A_i ← |A_j|; A_j ← T
       end EXCHANGE
```

ALGORITHM 5.3

Procedure UNFOLD(i)
  $R_r \leftarrow R_t\ (-i)$
  ROUTE($u(i)$, $g(i)$)
  $R_s \leftarrow R_r\ (i)$
end UNFOLD

Let us see how FOLD, EXCHANGE, and UNFOLD can be used to perform the permutation A defined at the beginning of this section. The factorization to be used is $A = B_{ij}B_{ik}$. This can be carried out as follows:

$$\text{FOLD}(i);\ \text{EXCHANGE}(i, j);\ \text{EXCHANGE}(i, k);\ \text{UNFOLD}(i).$$

The total number of unit-routes is $2(g(i) + g(j) + g(k))$, which is equal to $\beta(A)$.

Now, let us see how this three-stage interchange may be used to perform any permutation A in $\beta(A)$ unit-routes. The permutation algorithm is conceptually quite similar to PERMUTE. We follow cycles in the permutation. If the current cycle contains a bit $i \in L$, then it must also contain an interchangeable pair as required by line 17 of PERMUTE. In this case the bit interchange proceeds as before. The two-dimensional and q-dimensional algorithms differ only in their handling of a cycle containing no bit of type L. In this case PERMUTE enters the **then** clause of line 11. At this time the cycle including bit i contains only bits of type E and R. Such a cycle is an *RE-cycle*. We shall show how to remove all bits of type E from an RE-cycle leaving behind a set of cycles containing only bits of type R and L.

Let i be the smallest bit on an RE-cycle. Bit i must be of type E. Follow the cycle starting from bit i (see Figure 8(a)). Let the sequence of type E bits be $i, i_1, i_2, \ldots, i_r$. Note that $u(|A_{i_r}|) = u(i)$ as the portion of the RE-cycle from i_r to i includes only bits of type R. If the bit exchanges $(i, i_1), (i, i_2), \ldots, (i, i_r)$ are carried out (in that order), then the RE-cycle is broken into a set of disjoint cycles which include only bits of type R, S, and L (see Figure 8(b)). These cycles have the property that performing the permutations described by them completes the permutation required by the original RE-cycle.

Procedure RE-CYCLE follows an RE-cycle carrying out the bit interchange sequence described above. It is assumed that i is the smallest bit on the cycle and so $i \in E$. The

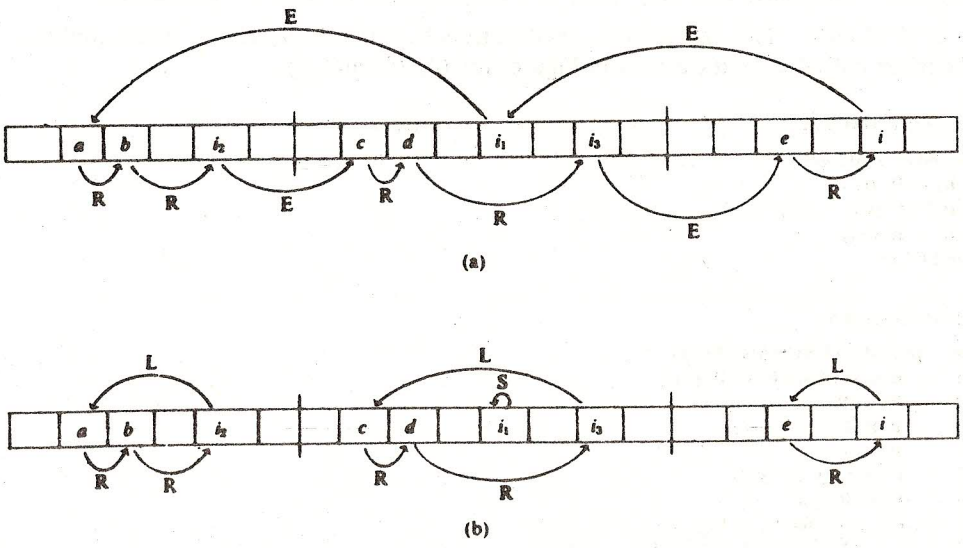

FIG. 8. Handling an RE-cycle: (a) an RE-cycle in A; (b) cycle structure following exchanges of E-type bits

number of unit-routes used is $2(g(i) + g(i_1) + g(i_2) + \cdots + g(i_r))$ where $i_1, i_2, \ldots, i_r$ are the remaining bits of type E on the cycle.

```
procedure RE-CYCLE(i)
  FOLD(i)
  loop
    j ← |A_i|
    while j ∉ E and j ≠ i do j ← |A_j| end //skip R type bits//
    if j = i then [UNFOLD(i); return]//end of cycle//
    EXCHANGE(i, j)
  forever
end RE-CYCLE
```

If we look at the cycles created following an application of RE-CYCLE we see that every bit $i, i_1, \ldots, i_r$ (i.e., that was previously of type E) is now of type L^+ or S^+. Let $\beta(C)$ be the sum of the β values for all the created cycles. Using (2.3) and the fact that all bits previously of type E get changed to bits of type L^+ or S^+, one can easily see that $\beta(A) = \beta(C) + 2(g(i) + g(i_1) + \cdots + g(i_r))$. Hence using RE-CYCLE in place of INTERCHANGE in line 11 of PERMUTE enables us to perform any q-dimensional permutation A in $\beta(A)$ unit-routes. The changes to PERMUTE are

(i) delete $e_0 \leftarrow e_1 \leftarrow p$ from line 6,
(ii) delete line 8,
(iii) replace the then clause of lines 11 and 12 by RE-CYCLE(i).

Since in the then clause of line 11, $i = j$ and $|A_r| = r$ for $r < j$ and $|A_i| \neq i$, it follows that i is of type E and is the smallest index on the cycle. So, it satisfies the requirements for RE-CYCLE. Clearly, the number of long-routes and register transfers used by PERMUTE remains $O(p)$ and the number of unit-routes is $\beta(A)$.

6. *MCC's with Wraparound*

In this section we determine the amount by which PERMUTE can be speeded if the MCC has a wraparound connection. Two kinds of wraparound connections are considered:

(i) Orthogonal Wraparound (OW). Here, each $PE(i_{q-1}, \ldots, i_k, \ldots, i_0)$ is connected to $PE(i_{q-1}, \ldots, (i_k \pm 1) \bmod n_k, \ldots, i_0)$, $0 \leq k < q$.

(ii) Propagating Wraparound (PW). Each $PE(i_{q-1}, \ldots, i_k, \ldots, i_0)$ is connected to $PE((i_{q-1}, \ldots, i_k \pm 1, \ldots, i_0) \bmod N)$, $0 \leq k < q$. For $q = 2$, this yields the ILLIAC IV wraparound scheme.

In the following discussion we shall restrict ourselves to OW MCC's. Later we will show the connection between the results obtained for OW MCC's and corresponding results for PW MCC's.

A permutation A contains a *wraparound distance* (or simply I_k *distance*) δ along the kth dimension if it requires data to be moved from PE(m) to PE(d) such that $DIST_k(m, d) = \sum_{i \in I_k} (d_i - m_i)g(i)$ equals either δ or $n_k - \delta$. Note that if any permutation A maps PE(m) to PE(d), then it also maps PE($\bar{m}$) to PE($\bar{d}$) and $DIST_k(m, d) = -DIST_k(\bar{m}, \bar{d})$.

THEOREM 6.1. *Let A be a permutation containing the I_k distances $\delta_1, \delta_2, \ldots, \delta_j$ such that $0 \leq \delta_1 < \delta_2 < \cdots < \delta_j \leq n_k/2$. Let $MAXGAP = max_{1 \leq i \leq j}\{\delta_i - \delta_{i-1}\}$ where $\delta_0 = 0$. A lower bound γ_k on the number of unit-routes along dimension k needed to perform A on an OW MCC is given by*

$$\gamma_k = min\{2\delta_j, n_k - MAXGAP\}.$$

PROOF. The expression for γ_k may be rewritten as

$$\gamma_k = \min_{1 \leq i \leq j} \{2\delta_j, (n_k - \delta_i) + \delta_{i-1}\}.$$

TABLE VI. β AND γ VALUES FOR SOME PERMUTATIONS ON AN $n \times n$ MCC

Permutation	β	γ
Matrix transpose	$4(n-1)$	$2(n-1)$
Bit reversal	$4(n-1)$	$2(n-1)$
Vector reversal	$4(n-1)$	$2(n-2)$
Perfect shuffle	$2n$	$2(n-1)$
Bit shuffle ($p/2$ even)	$\frac{8n-2}{3} - 2\sqrt{n}$	$\frac{5n-2}{3} - \sqrt{n}$

Let Z^+ and Z^-, respectively, be the number of unit positive and negative routes used by any algorithm to perform A on an OW MCC. Without loss of generality, we may assume $Z^+ \geq Z^-$. We wish to show that $Z^+ + Z^- \geq \gamma_k$. If $Z^- \geq \delta_j$, then $Z^+ + Z^- \geq 2\delta_j \geq \gamma_k$. So we may assume $Z^- < \delta_j$. Let i, $i \leq j$, be such that $\delta_{i-1} \leq Z^- < \delta_i$. As pointed out earlier, every permutation containing the distance δ_i must also contain the distance $-\delta_i$. Since $Z^- < \delta_i$, it follows that the routing for the distance $-\delta_i$ must be carried out using positive routes and the wraparound connection on this dimension. Hence $Z^+ \geq n_k - \delta_i$. So $Z^+ + Z^- \geq (n_k - \delta_i) + \delta_{i-1} \geq \gamma_k$. □

$\gamma = \sum_{k=0}^{q-1} \gamma_k$ is a lower bound on the number of unit-routes needed to perform a permutation A on an OW MCC.

Example 6.1. For an $n \times n$ OW MCC, the matrix transpose permutation $A = [p/2 - 1, \ldots, 0, p - 1, \ldots, p/2]$ contains all distances $1, 2, \ldots, n/2$ for both I_0 and I_1. To see this, observe that data from $PE(m_{p-1} \cdots m_{p/2}\ 000 \cdots 0)$ are to be routed to $PE(000 \cdots 0m_{p-1} \cdots m_{p/2})$. $DIST_1(m, d) = -\sum_{i=p/2}^{p-1} m_i g(i)$, and $DIST_0(m, d) = -DIST_1(m, d)$. By assigning all possible 0, 1 values to $m_{p-1} \cdots m_{p/2}$, all distances for I_0 are obtained. The distances for I_1 are negative but we know that $-DIST_1(m, d)$ also exists along dimension 1. Using only the distances in the range $[0, n/2]$, we have MAXGAP = 1 and $\delta_j = n/2$. Hence, $\gamma_0 = \gamma_1 = n - 1$ and $\gamma = 2(n - 1)$. Table VI gives the γ values obtained for some other permutations. The γ values for bit reversal and perfect shuffle are obtained by showing that all distances $1, 2, \ldots, n/2$ exist. For vector reversal, it can be shown that only the odd distances $1, 3, 5, \ldots, n/2 - 1$ exist. The lower bound for bit shuffle is derived later.

From the proof of Theorem 6.1 it is easy to conclude that any permutation can be performed in γ unit-routes on an OW MCC with "enough" registers and the ability to route many pieces of data from one PE to another in one step. To see this, let $0 \leq \delta_1 < \delta_2 < \cdots < \delta_j \leq n_k/2$ be the only I_k distances contained in A in the range $[0, n_k/2]$. (Note that a distance $\delta > n_k/2$ is equivalent to the distance $-(n_k - \delta)$, and $\delta < -n_k/2$ is equivalent to the distance $n_k + \delta$. So effectively all distances in any A can be thought of as in the range $[-n_k/2, n_k/2]$.) Now, if $2\delta_j \leq n_k - \text{MAXGAP}$, then $\gamma_k = 2\delta_j$. All routings in dimension k can be performed using δ_j positive routes followed by δ_j negative routes. If $2\delta_j > n_k - \text{MAXGAP}$, then $\gamma_k = (n_k - \delta_i) + \delta_{i-1}$ for some i, $i \in [1, j]$. Now the routing for dimension k can be carried out using $n_k - \delta_i$ positive routes and δ_{i-1} negative routes.

The preceding scheme, of course, cannot be used on OW MCC's with a fixed number (independent of the n_k's) of registers and an ability to send only one word of data from one PE to another in one unit-route.

Let α be the minimum number of unit-routes needed to perform permutation A on a PW MCC. The following theorem establishes a relationship between γ and α.

THEOREM 6.2. *For every permutation A,* $\alpha \geq \gamma - 2(q - 1)$.

PROOF. First, we observe that in a PW MCC data in $PE(i_{q-1}, \ldots, i_k, \ldots, i_0)$ can be moved to $PE(j_{q-1}, \ldots, j_k, i_{k-1}, \ldots, i_0)$ by routing along dimensions $k, k - 1, \ldots, 0$. However, routes along dimension r cannot be used to change a dimension s for $s < r$.

From Theorem 6.1 we know that $\gamma_k \leq n_k - 1$, $0 \leq k < q$. Since routings along

dimensions k, $k > 0$, do not affect distances along dimension 0 in a PW MCC, it follows that $\alpha_0 \geq \gamma_0$.

First, we shall establish that $\alpha_j \geq \gamma_j - 2$ when all α_r are restricted to be no more than n_r, $0 \leq r < q$. Under this restriction, we see that for any dimension j, $j > 0$, the I_j distance can be reduced by at most 1 because of routes in other dimensions. Now, using the same argument as used in Theorem 6.1 we see that $\alpha_j \geq \gamma_j - 2$ (for any distance sequence $\delta_1, \ldots, \delta_j$, distance δ_i can decrease by at most 1, and n_k − MAXGAP can decrease by at most 1). So under the restriction $\alpha_r \leq n_r$, $0 \leq r < q$, it is the case that $\alpha = \sum \alpha_j \geq \gamma - 2(q - 1)$.

When α_r is allowed to be more than n_r, the theorem follows from the observation that the number of additional lower dimension routes needed to decrease a higher dimension distance exceeds or equals the gains to be obtained by this decrease. For example, in an $n_1 \times n_0$ PW MCC suppose $\alpha_0 > n_0$ routes are used in dimension 0. The maximum decrease in any I_1 distance is $\lceil \alpha_0/n_0 \rceil$. So $\alpha_1 \geq \gamma_1 - 2\lceil \alpha_0/n_0 \rceil$. Hence

$$\alpha_1 + \alpha_0 \geq \gamma_0 + \gamma_1 + (\alpha_0 - \gamma_0 - 2\lceil \alpha_0/n_0 \rceil) \geq \gamma_0 + \gamma_1 + (\alpha_0 - n_0 + 1 - 2\lceil \alpha_0/n_0 \rceil).$$

Let $\alpha_0 = an_0 + b$ for some $a > 0$ and $0 \leq b < n_0$. We see that $\alpha_0 - n_0 + 1 - 2\lceil \alpha_0/n_0 \rceil \geq -2$. So $\alpha \geq \gamma - 2$. □

The remainder of this section is devoted to obtaining a lower bound on the ratio γ/β. The reasons for doing this are twofold. (i) This bound will give us the maximum improvement one can expect in permutation routing algorithms when going from an MCC without wraparound connections to one with such connections. (ii) This bound tells us how inefficient it may be to use PERMUTE on an MCC with wraparound connections and ignore these connections while performing a permutation A. We shall show that $\gamma/\beta \geq \frac{1}{2}$ when A has no complemented bits of type S, L, or R; and $\gamma/\beta \geq \frac{1}{3}$ when such bits are allowed.

Before obtaining the above bounds on the ratio γ/β, we shall obtain a relationship for γ_k in terms of the MAXGAP in an *unordered sequence* of I_k distances $a_1, a_2, \ldots, a_j$ where $a_i \in [-n_k, n_k]$.

THEOREM 6.3. *Let $a = a_1, a_2, \ldots, a_j$ be a sequence of I_k distances contained in the permutation A. Then,*

$$\gamma_k \geq \mu(a) = min\{2a_j, n_k - MAXGAP(a)\},$$

where $MAXGAP(a) = max_{1 \leq i \leq j}\{a_i - a_{i-1}\}$ and $a_0 = 0$.

PROOF. We shall show that the sequence $a_1, \ldots, a_j$ can be transformed into a sequence of I_k distances $\delta_1, \ldots, \delta_r$ with the property $0 \leq \delta_1 < \delta_2 < \cdots < \delta_r \leq n_k/2$ and $\mu(\delta) \geq \mu(a)$. Since $\gamma_k \geq \mu(\delta)$ (Theorem 6.1), it will follow that $\gamma_k \geq \mu(a)$.

First, consider the sequence of I_k distances $b = (b_1, b_2, \ldots, b_j)$ obtained by sorting a into nondecreasing order. Clearly, b is a sequence of I_k distances in A and $b_j \geq a_j$ and MAXGAP$(b) \leq$ MAXGAP(a). So $\mu(b) \geq \mu(a)$.

Next, eliminate from b multiple copies of any distance so as to get a sorted sequence of distinct distances ($-x$ and $+x$ are distinct) $c = c_1, c_2, \ldots, c_s$. $\mu(c) = \mu(b)$.

Since every permutation containing an I_k distance x also contains an I_k distance $-x$ (DIST$_k(m, d) = -$DIST$_k(\bar{m}, \bar{d})$), we can transform c into an ordered sequence of nonnegative distinct distances $f = f_1, f_2, \ldots, f_t$. This is obtained from c by replacing each negative c_i by $-c_i$ and inserting $-c_i$ into the correct spot in the ordered sequence. Clearly, $f_t \geq c_s$ and MAXGAP$(f) \leq$ MAXGAP(c). So, $\mu(f) \geq \mu(c)$.

Now, we need to eliminate from f all distances greater than $n_k/2$. Assume $f_t > n_k/2$ (if $f_t \leq n_k/2$, then all $f_i \leq n_k/2$ and $\gamma_k \geq \mu(f)$ and we are done). If A contains an I_k distance x, then it contains the distance $-x$ which is equivalent to the distance $n_k - x$. Hence we may replace each $f_i > n_k/2$ by the distance $n_k - f_i$ to get the ordered distance sequence h

$= h_1, h_2, \ldots, h_v$. Each h_i is distinct. To get h from f, some rearrangement and elimination may be necessary. It should be clear that every distance in h is an I_k distance in A and $\text{MAXGAP}(h) \leq \text{MAXGAP}(f)$. However, $h_v \leq n_k/2 < f_t$.

Since $2f_t > n_k > n_k - \text{MAXGAP}(f)$, $\mu(f) = n_k - \text{MAXGAP}(f)$. We wish to show that $\mu(h) = \min\{2h_v, n_k - \text{MAXGAP}(h)\} \geq \mu(f) = n_k - \text{MAXGAP}(f)$. If $\mu(h) = n_k - \text{MAXGAP}(h)$, then since $\text{MAXGAP}(h) \leq \text{MAXGAP}(f)$, $\mu(h) \geq \mu(f)$. So assume $\mu(h) = 2h_v$.

Let r be largest index such that $f_r \leq n_k/2$. If $n_k - f_{r+1} \leq f_r$ (f_{r+1} exists as $f_t > n_k/2$), then $h_v = f_r$ and $2h_v = 2f_r \geq (n_k - f_{r+1}) + f_r \geq n_k - \text{MAXGAP}(f) = \mu(f)$. If $n_k - f_{r+1} > f_r$, then $h_v = n_k - f_{r+1}$ and $2h_v = 2(n_k - f_{r+1}) > (n_k - f_{r+1}) + f_r \geq n_k - \text{MAXGAP}(f) = \mu(f)$. □

THEOREM 6.4. *$\gamma_k/\beta_k \geq \frac{1}{2}$ when $S_k^- + L_k^- + R_k^- = \emptyset$.*

PROOF. We shall show that whenever $S_k^- + L_k^- + R_k^- = \emptyset$, A contains a sequence δ of I_k distances for which $\mu(\delta)/\beta_k \geq \frac{1}{2}$. Hence, $\gamma_k/\beta_k \geq \frac{1}{2}$.

Let $X_k = E_k + R_k^+ + S_k^+$ and $Y_k = L_k^+ + (E \cdot A^{-1}(I_k)) = \{i_{t-1}, i_{t-2}, \ldots, i_0\}$. We may assume that $i_{t-1}, \ldots, i_0$ are ordered such that

$$|A_{i_{t-1}}| > |A_{i_{t-2}}| > \cdots > |A_{i_0}|.$$

Let m be the address of any PE and let SEL_k be the set of T_k bits (see Section 2) in m having value as given in the statement of Lemma 2.1. From Table II it is easy to see that if $X_k \subseteq SEL_k$, then $DIST_k(m, d) = \sum_{i \in SEL_k - X_k} h_k(i)$. Let $G(s) = X_k + \{i_r : \text{bit } r \text{ is one in the binary representation of } s\}$. By choosing $SEL_k = G(s)$, $0 \leq s \leq 2^t - 1$, we see that A contains the following distance sequence δ:

$$\begin{aligned}
&\delta_0 = 0 && //SEL_k = X_k// \\
&\delta_1 = h_k(i_0) && //SEL_k = X_k + \{i_0\}// \\
&\delta_2 = h_k(i_1) && //SEL_k = X_k + \{i_1\}// \\
&\delta_3 = h_k(i_1) + h_k(i_0) && //SEL_k = X_k + \{i_1, i_0\}// \\
&\delta_4 = h_k(i_2) \\
&\delta_5 = h_k(i_2) + h_k(i_0) \\
&\delta_6 = h_k(i_2) + h_k(i_1) \\
&\delta_7 = h_k(i_2) + h_k(i_1) + h_k(i_0) \\
&\vdots \\
&\delta_j = \textstyle\sum_{i \in Y_k} h_k(i) = \beta_k/2 && //SEL_k = X_k + Y_k = T_k//
\end{aligned}$$

For δ, we see that

$$\text{MAXGAP}(\delta) = \max_{1 \leq r \leq j} (\delta_r - \delta_{r-1}) = h_k(i_a) - \sum_{r=0}^{a-1} h_k(i_r)$$

for some a, $0 \leq a \leq t - 1$. Recall that for $i \in Y_k$, $|A_i| \in I_k$ and $h_k(i) \leq g(|A_i|)$. Using this and the fact that n_k is a power of 2, we obtain

$$\begin{aligned}
n_k &= \sum_{i \in I_k} g(i) + 1 = \sum_{\substack{i \in I_k \\ i > |A_{i_a}|}} g(i) + 2g(|A_{i_a}|) \\
&\geq \sum_{r=a+1}^{t-1} g(|A_{i_r}|) + 2g(|A_{i_a}|) \geq \sum_{r=a+1}^{t-1} h_k(i_r) + 2h_k(i_a).
\end{aligned}$$

Hence,

$$n_k - \text{MAXGAP}(\delta) \geq \sum_{r=0}^{t-1} h_k(i_r) = \sum_{i \in Y_k} h_k(i) = \beta_k/2.$$

From this and the observation that $2\delta_j = \beta_k > \beta_k/2$, it follows that $\mu(\delta) \geq \beta_k/2$. □

Theorem 6.4 can be used to arrive at the γ value for the bit shuffle permutation given in Table VI. For the case of an MCC without wraparound, we can show that (assuming $p/2$ is even)

$$\beta_0 = 2(n - \sqrt{n}) \quad \text{and} \quad \beta_1 = 2(n-1)/3.$$

Hence, $\gamma_0 \geq (n - \sqrt{n})$ and $\gamma_1 \geq (n-1)/3$. We can get a better bound on γ_1 using Theorem 6.3 and the distance $\delta = \beta_1/2 = (n-1)/3$ (recall that $\text{DIST}_k(m, d) = \beta_k/2$ for m chosen as in Lemma 2.1). We get $\gamma_1 > \mu(\delta) = \min\{2(n-1)/3, n - (n-1)/3\} = 2(n-1)/3$. So $\gamma = \gamma_0 + \gamma_1 \geq (5n-2)/3 - \sqrt{n}$.

The bound $\gamma_k/\beta_k \geq \frac{1}{2}$ does not hold for all A with $S_k^- + L_k^- + R_k^- \neq \emptyset$. To see this, consider A = [−0, −1] and a 1×4 MCC. From Table II and eq. (2.3) we obtain $\beta = \beta_0 = 2(g(1) + g(0)) = 6$. The only I_0 distance, δ, on a 1×4 OW MCC is 1 corresponding to the exchanges between PE(0) and PE(3). Hence, $\gamma = \min\{2, 4 - \delta\} = 2$. So $\gamma/\beta = \frac{1}{3}$. We shall next show that $\gamma/\beta \geq \frac{1}{3}$ for all permutations A.

THEOREM 6.5. *$\gamma_k/\beta_k \geq \frac{1}{3}$ whenever $\beta_k > 0$.*

PROOF. This theorem is proved by considering five cases. Each case establishes the theorem for a range of β_k values. Note that $\beta_k \in [0, 2(n_k - 1)]$. The details of the proof may be found in [10]. □

7. *Conclusions*

We have developed an algorithm to perform a rich class of permutations on an MCC. If the MCC has $N = 2^p$ PE's, then our algorithm uses $O(\log^2 N)$ units of control unit time to decide the routing sequence. The number of unit-routes used is $\beta(A)$, where A is the permutation being performed. Hence it is optimal as far as unit-routes are concerned. The number of long-routes is $O(\log N)$. In Section 6 we showed that if the algorithm is used on MCC's with wraparound, then it will use no more than three times the optimal number of unit-routes.

REFERENCES

(Note. Reference [2] is not cited in the text.)

1. BARNES, G.H., ET AL. The ILLIAC IV computer. *IEEE Trans. Comptrs. C-17*, 8 (1968), 746–757.
2. BATCHER, K.E. Sorting networks and their applications. Proc. AFIPS 1968 SJCC, Vol. 32, AFIPS Press, Montvale, N.J., pp. 307–314.
3. BENES, V.E. *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic Press, New York, 1965.
4. CSANKY, L. Fast parallel matrix inversion algorithms. 16th Annual IEEE Symp. on Foundations of Computer Science, Berkeley, Calif., 1975, pp. 11–12.
5. FLYNN, M.J. Very high-speed computing systems. *Proc. IEEE 54* (1966), 1901–1909.
6. GENTLEMAN, W.M., Some complexity results for matrix computations on parallel processors. *J. ACM. 25*, 1 (Jan. 1978), 112–115.
7. LANG, T. Interconnections between processors and memory modules using the shuffle-exchange network. *IEEE Trans. Comptrs. C-25*, 5 (1976), 496–503.
8. LAWRIE, D.E. Memory-processor connection networks. Ph.D. Diss., U. of Illinois at Urbana-Champaign, Urbana, Ill., 1973.
9. NASSIMI, D., AND SAHNI, S. Bitonic sort on a mesh-connected parallel computer. *IEEE Trans. Comptrs. C-28*, 1 (1979), 2–7.
10. NASSIMI, D., AND SAHNI, S. An optimal routing algorithm for mesh-connected parallel computers. Tech. Rep. No. 78-19, U. of Minnesota, Minneapolis, Minn., 1978.
11. ORCUTT, S.E. Implementation of permutation functions in Illiac IV-type computers. *IEEE Trans. Comptrs. C-25*, 9 (1976), 929–936.
12. SIEGEL, H.J. Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks. *IEEE Trans. Comptrs. C-26*, 2 (1977), 153–161.
13. STONE, H.S. Parallel processing with the perfect shuffle. *IEEE Trans. Comptrs. C-20*, 2 (1971), 153–161.
14. SWANSON, R.C. Interconnections for parallel memories to unscramble *p*-ordered vectors. *IEEE Trans. Comptrs. C-23*, 11 (1974), 1105–1115.

15. THOMPSON, C.D. Generalized connection networks for parallel processor intercommunication. *IEEE Trans. Comptrs. C-27,* 12 (1978), 1119–1125.
16. THOMPSON, C.D., AND KUNG, H.T. Sorting on a mesh-connected parallel computer. *Comm. ACM 20,* 4 (1977), 263–271.
17. WAKSMAN, A. A permutation network. *J. ACM 15,* 1 (Jan. 1968), 159–163.

RECEIVED AUGUST 1978; REVISED FEBRUARY 1979