STAN-CS-74-429

# Efficient Data Routing Schemes for ILLIAC IV-Type Computers

by

Samuel E. Orcutt

Technical Report No. 70

April 1974

Digital Systems Laboratory

Stanford Electronics Laboratories

Stanford, California

This work was supported by Bell Laboratories and by the National Science Foundation under grant GJ-41093.

| BIBLIOGRAPHIC DATA SHEET | 1. Report No. Technical Report No. 70 | 2. | 3. Recipient's Accession No. |
|---|---|---|---|

4. Title and Subtitle

Efficient Data Routing Schemes for ILLIAC IV-Type Computers

5. Report Date: April 1974

6.

7. Author(s): Samuel E. Orcutt

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Stanford University
Digital Systems Laboratory
Stanford, California 94305

10. Project/Task/Work Unit No.

11. Contract/Grant No.: NSF GJ-41093

12. Sponsoring Organization Name and Address

National Science Foundation
1800 G Street NW
Washington, D.C. 20550

13. Type of Report & Period Covered: technical

14.

15. Supplementary Notes

STAN-CS-74-429

16. Abstracts

Much research has recently been done on processor interconnection schemes for parallel computers. These interconnection schemes allow certain permutations to be performed in less than linear time, typically $O(\log N)$ or $O(\sqrt{N})$ for a vector of N elements and N processors. In this paper we show that many permutations can also be performed in less than linear time on a machine with an ILLIAC IV-type interconnection scheme, that is connections to processors at distances of $\pm 1$ and $\pm\sqrt{N}$. These results show that recently developed interconnection schemes yield less speedup over the ILLIAC IV-type interconnections than was thought. These results are of current interest due to the present ILLIAC IV programming effort.

17. Key Words and Document Analysis. 17a. Descriptors

ILLIAC IV, parallel computation, permutations, perfect shuffle, bit reversal, bitonic sorting, data routing.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field Group



| 18. Availability Statement | 19. Security Class (This Report) | 21. No. of Pages |
|---|---|---|
| Approved for public release; distribution unlimited | UNCLASSIFIED | 37 |
| | 20. Security Class (This Page) UNCLASSIFIED | 22. Price $5.00-1.45 |

## INDEX TERMS

ILLIAC IV, parallel computation, permutations, perfect shuffle, bit reversal, bitonic sorting, data routing.

FIGURE CAPTIONS

# TABLE CAPTIONS

# EFFICIENT DATA ROUTING SCHEMES FOR ILLIAC IV-TYPE COMPUTERS

by Samuel E. Orcutt

## Abstract

Much research has recently been done on processor interconnection schemes for parallel computers. These interconnection schemes allow certain permutations to be performed in less than linear time, typically $O(\log N)$ or $O(\sqrt{N})$ for a vector of N elements and N processors. In this paper we show that many permutations can also be performed in less than linear time on a machine with an ILLIAC IV-type interconnection scheme, that is connections to processors at distances of $\pm 1$ and $\pm\sqrt{N}$. These results show that recently developed interconnection schemes yield less speedup over the ILLIAC IV-type interconnections than was thought. These results are of current interest due to the present ILLIAC IV programming effort.

## I. INTRODUCTION

One of the factors involved in determining the usefulness of a parallel computer is the facility available for communicating results between processors. Much research has recently been done on these interconnection schemes. Representative of the interconnection schemes proposed are sorting networks [Rohrbacher, 1966], the shuffle-exchange [Stone, 1971; Lang, 1973], p-ordered vectors [Swanson, 1973], and $\Omega$-networks [Lawrie, 1973]. These interconnection schemes allow certain permutations to be performed in less than linear time, typically $O(\log N)$ or $O(\sqrt{N})$ for a vector of N elements and N processors. In this paper we show that these permutations can also be performed in less than linear time on a machine with ILLIAC IV-type interconnections.

In the next section we describe the interconnection scheme that we consider, and we give the notation used in the rest of the report. Sections III and IV present specific algorithms for the perfect shuffle and bit reversal. In Section V we present an algorithm capable of handling a wide variety of permutations in $O(\sqrt{N})$ time. Section VI describes the application of this algorithm to a specific problem, namely the p-ordered vectors of Swanson [1973]. In Section VII an algorithm is presented that is capable of doing any permutation in $O(\sqrt{N} \log N)$ time. This shows that recently developed interconnection schemes yield a speedup of at most $O(\sqrt{N})$ as compared with the ILLIAC IV-type interconnections.

## II. BASIC DEFINITIONS

To analyze algorithms for parallel computers it is necessary to make certain assumptions regarding the structure of the computer on which the algorithms are to execute. So these analyses will be of practical use, the assumptions made in this paper are consistent with the structure of the ILLIAC IV.

The basic organization of the machine we consider is Single Instruction stream - Multiple Data stream (SIMD [Flynn, 1966]). In this class of machine all Processing Elements (PE's) obtain their instructions simultaneously from a single instruction stream. Each PE will, in general, execute this instruction stream with different data. A PE is either enabled or disabled from executing instructions according to its own enable bit.

We consider a parallel computer with $N = 2^n$ PE's, for some even integer n, each labelled with a distinct index between 0 and N-1. A PE is connected to PE's whose indices differ by $\pm 1$ and $\pm\sqrt{N}$. In particular, PE i is connected to PE's $(i-\sqrt{N}) \bmod N$, $(i-1) \bmod N$, $(i+1) \bmod N$, and $(i+\sqrt{N}) \bmod N$. The ILLIAC IV interconnection scheme is precisely this scheme for $N = 64$.

Communication of data between PE's is accomplished by cyclically shifting data between the PE's. This data transfer is referred to as a route. The physical interconnections that are present allow routes of $\pm 1$ and $\pm\sqrt{N}$ to be done in a single step. These four routes are referred to as unit routes. We assume that a single route instruction can be used to route any distance k, $1 \leq k \leq N-1$. The execution of a route instruction

is done by performing a sequence of unit routes in such a manner as to move data the required distance. For example, a route of distance 5 can be done as five routes of +1. The length of a route, $\ell(k)$, is the minimum number of unit routes required to route a distance k. For example, $\ell(\sqrt{N}-1) = 2$ since a distance of $\sqrt{N}-1$ can be obtained by a route of $+\sqrt{N}$ followed by a route of -1. It is easily shown that

$$1 \leq \ell(k) \leq \sqrt{N}-1$$

The time required to route a distance k is given by

$$T(k) = a + b \cdot \ell(k)$$

where a and b are machine dependent constants.

The algorithms in this paper are given in an ALGOL-like language previously used by Stone [1973] and Kogge [1972]. The main difference from ALGOL is an extension to describe the parallel features of algorithms as follows:

1) An inequality of the form $(r \leq i \leq s)$ following a statement indicates that the statement is to be executed simultaneously for all values of the index in the specified range.
2) If M is a logical vector, a vector whose elements are either true or false, of appropriate dimension then an expression of the form (M(i)) following a statement indicates that the statement is to be executed simultaneously for all values of the index corresponding to true elements in M. When used in this manner the vector M is called a control vector.

Additionally, we make frequent use of the function

$$\text{bit}(i,j)$$

which yields the value of the $j^{th}$ bit in the binary representation of i where the least significant bit is bit 0.

In the analyses of algorithms that occur in subsequent sections, we consider only the time used by route instructions when estimating execution times. We let the execution time of an algorithm be given by

$$T = a \cdot R + b \cdot L$$

where R is the total number of route instructions and L is the total number of unit routes required.

## III. PERFECT SHUFFLE

An intercommunication pattern frequently encountered is the perfect shuffle. A shuffle of the elements of a vector is equivalent to viewing the vector as a card deck, and shuffling them so that after a shuffle the elements from the two halves of the vector alternate. Under this pattern the $i^{th}$ element is shuffled into position i' where i' is obtained by shifting the bits in the binary representation of i left one bit cyclically. In this section we consider performing the perfect shuffle with a parallel computer having $\pm 1$ and $\pm\sqrt{N}$ interconnections.

The algorithm for performing the perfect shuffle is based on the construction of a size 2m perfect shuffler from two size m perfect shufflers and a size 2m adjustment network. Starting with size 2 perfect shufflers, which perform no interchanges, a size $N = 2^n$ perfect shuffler can be constructed in $\log_2 (N/2) = n-1$ stages using only adjustment networks. A typical stage of this process is shown in Figure 3.1. The complete network for N = 16 is given in Figure 3.2. For this method to be useful, the adjustment networks must be easily implementable using the available interconnections. At stage k in the construction we are implementing adjustment networks of size $2^{k+1}$. A network of this size performs interchanges based upon bits 0 and k of the processor index as described in Table 3.1.

The algorithm for the perfect shuffle is easily derived from the description in Table 3.1. In this algorithm the array X contains the data

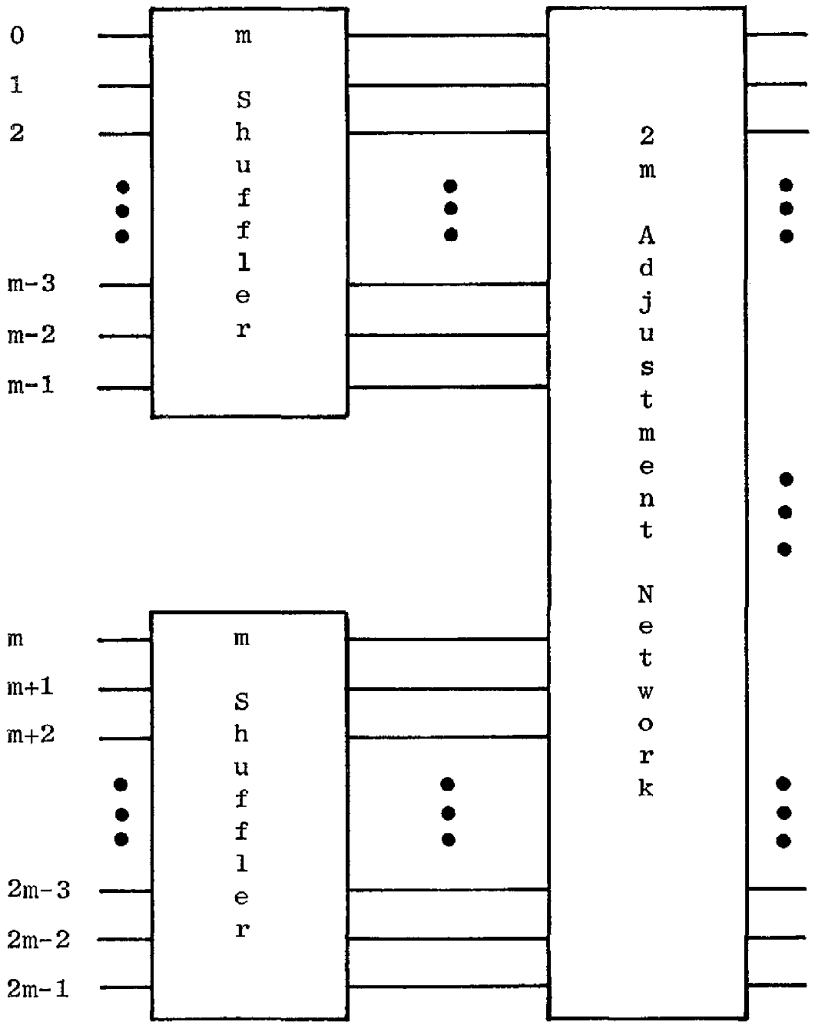


Figure 3.1

Construction of a size 2m perfect shuffler

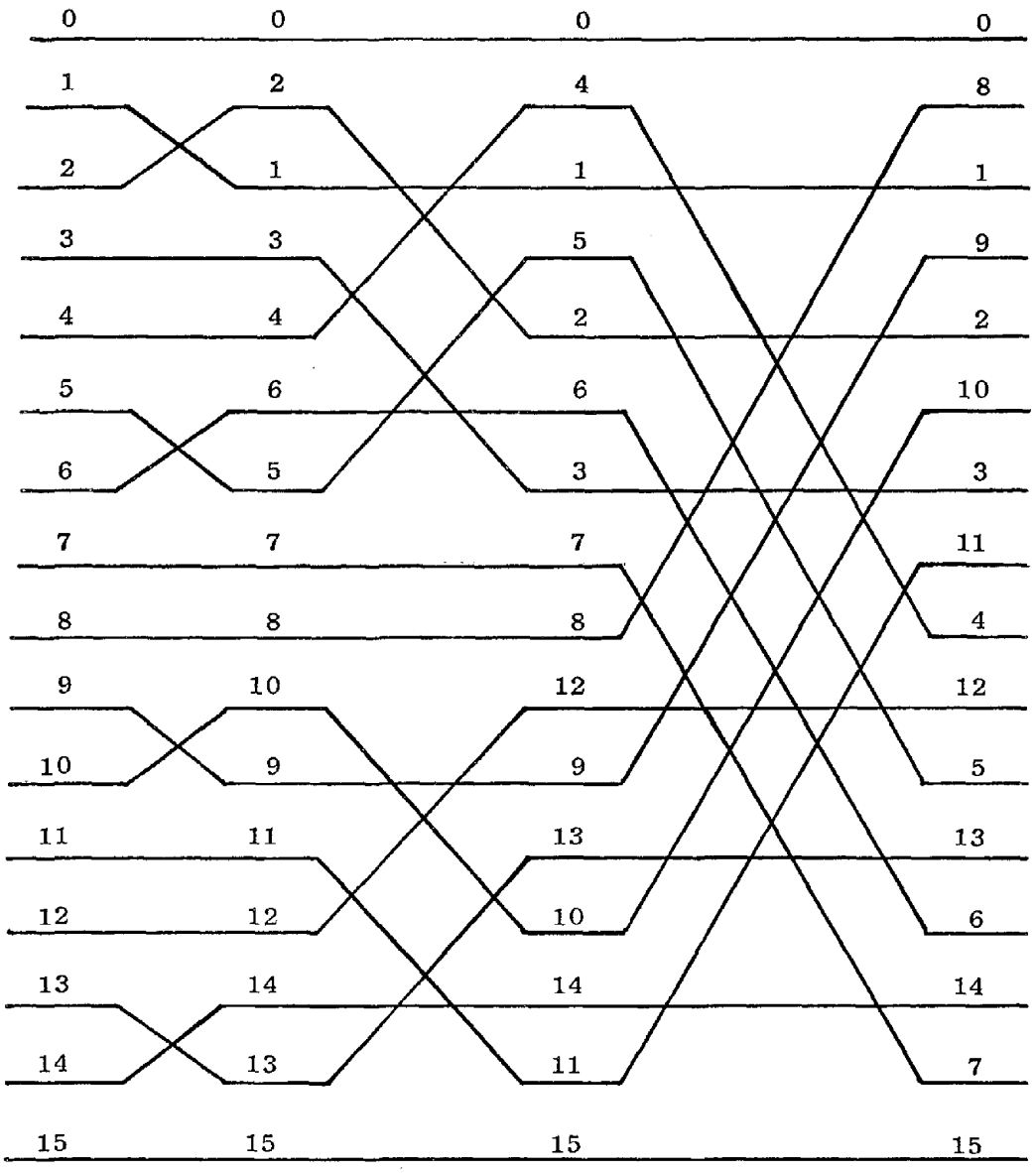


Figure 3.2

Size 16 perfect shuffler

Table 3.1

| bit k | bit 0 | action |
|---|---|---|
| 0 | 0 | do nothing |
| 0 | 1 | route $2^k-1$ |
| 1 | 0 | route $-(2^k-1)$ |
| 1 | 1 | do nothing |

Decision table for perfect shuffle.

to be shuffled, the array TEMP is a temporary storage area and the arrays MASK1 and MASK2 are control vectors.

```
for k := 1 step 1 until n-1 do begin
    MASK1(i) := ¬bit(i,k)∧bit(i,0), (0 ≤ i ≤ N-1);
    MASK2(i) := MASK1(i-(2^k-1)),    (2^k-1 ≤ i ≤ N-1);
    TEMP(i)  := X(i+2^k-1), (MASK1(i));
    TEMP(i)  := X(i-(2^k-1)), (MASK2(i));
    X(i)     := TEMP(i), (MASK1(i)∨MASK2(i));
end;
```

This algorithm works in a very straightforward manner. MASK1 and MASK2 determine which pairs of elements are to be exchanged according to Table 3.1. The new values are stored in the vector TEMP while interchanges are being made, and then copied back into X when the interchanges are completed.

To complete the presentation of this algorithm we now determine its execution time. Since there are n-1 stages and each stage requires three route instructions, the total number of routes is

$$R = 3n-3 = 3\cdot\log_2 N - 3 = O(\log N)$$

The evaluation of L is somewhat more difficult. L is given by

$$L = \sum_{k=1}^{n-1} 3\cdot\ell(2^k-1)$$

This is quite straightforward except that the behavior of the function $\ell(x)$ is quite peculiar. In particular if

$$x = a_1 \cdot \sqrt{N} + b_1 \qquad a_1 \geq 0, \quad 0 \leq b_1 \leq \sqrt{N} - 1$$

$$N-x = a_2 \cdot \sqrt{N} + b_2 \qquad a_2 \geq 0, \quad 0 \leq b_2 \leq \sqrt{N} - 1$$

then

$$\ell(x) = \min(a_1 + b_1,\ a_2 + b_2,\ a_1 + 1 + \sqrt{N} - b_1,\ a_2 + 1 + \sqrt{N} - b_2)$$

In this algorithm we know that $1 \leq x \leq N/2-1$. For x in this range it is seen that

$$\min(a_1 + b_1,\ a_1 + 1 + \sqrt{N} - b_1) \leq \min(a_2 + b_2,\ a_2 + 1 + \sqrt{N} - b_2)$$

From this we conclude that

$$\ell(x) = \min(a_1 + b_1,\ a_1 + 1 + \sqrt{N} - b_1) \quad 1 \leq x \leq N/2-1$$

By considering the form of the values of x required by this algorithm, it is easily shown that

$$\ell(2^k-1) = \begin{cases} 2^k-1 & 1 \leq k \leq n/2-1 \\ 2^{k-n/2}+1 & n/2 \leq k \leq n-1 \end{cases}$$

Using these values for $\ell(x)$ we obtain

$$L = 3 \cdot \sum_{i=1}^{n/2-1} (2^i-1) + 3 \cdot \sum_{i=n/2}^{n-1} (2^{i-n/2} + 1)$$

$$= 6(\sqrt{N}-1) = O(\sqrt{N})$$

As compared with the straightforward permutation method which requires

$$R = N-1 = O(N) \text{ and } L = N-1 = O(N)$$

this method yields a speedup of $O(\sqrt{N})$.

This algorithm allows us to adapt parallel algorithms that utilize perfect shuffle interconnections for execution on a machine of the type we consider. Although the implementation of the perfect shuffle is efficient, the algorithms developed by using this method to simulate the perfect shuffle are not necessarily the best possible. For example, this method can be used to adapt sorting and the Fast Fourier Transform, both of which have the perfect shuffle as a natural intercommunication pattern, for execution on a machine of this type, but these algorithms are not as efficient as those developed directly for this type machine.

## IV. BIT REVERSAL

Another problem of interest for computation on a parallel computer is the Fast Fourier Transform [Pease, 1968]. An idiosyncracy of this algorithm is that the results of the transform appear in bit reverse order, that is, if $i_{n-1}\ i_{n-2} \cdots i_1\ i_0$ is the binary representation of i then the $i^{th}$ component of the result is in position $i_0\ i_1 \cdots i_{n-2}\ i_{n-1}$ of the result vector. This unscrambling has recently been considered by Polge et al. [1974] for the serial computer case. In this section we present an algorithm for efficient bit reversal on an SIMD computer with interconnections similar to those of the ILLIAC IV. The algorithm given here, although developed independently, is a parallel equivalent of their algorithm. This algorithm illustrates the use of control vectors to replace loop structure when converting algorithms from serial to parallel.

The unscrambling of bit reversed vectors is done in stages as for the perfect shuffle. At stage k, $0 \le k \le n/2-1$, interchanges are performed based upon bits k and n-1-k of the processor index according to Table 4.1. Figure 4.1 shows a complete example of this procedure for N = 16. The interchanges are performed in such a way that after k steps all items are in the correct PE with respect to the high and low order k bits of their index.

We now present the actual unscrambling algorithm. The arrays X, TEMP, MASK1, and MASK2 are defined as in the previous section.

Table 4.1

| bit k | bit n-1-k | action |
|---|---|---|
| 0 | 0 | do nothing |
| 0 | 1 | route $-(2^{n-1-k}-2^{k})$ |
| 1 | 0 | route $(2^{n-1-k}-2^{k})$ |
| 1 | 1 | do nothing |

Decision table for bit reversal.

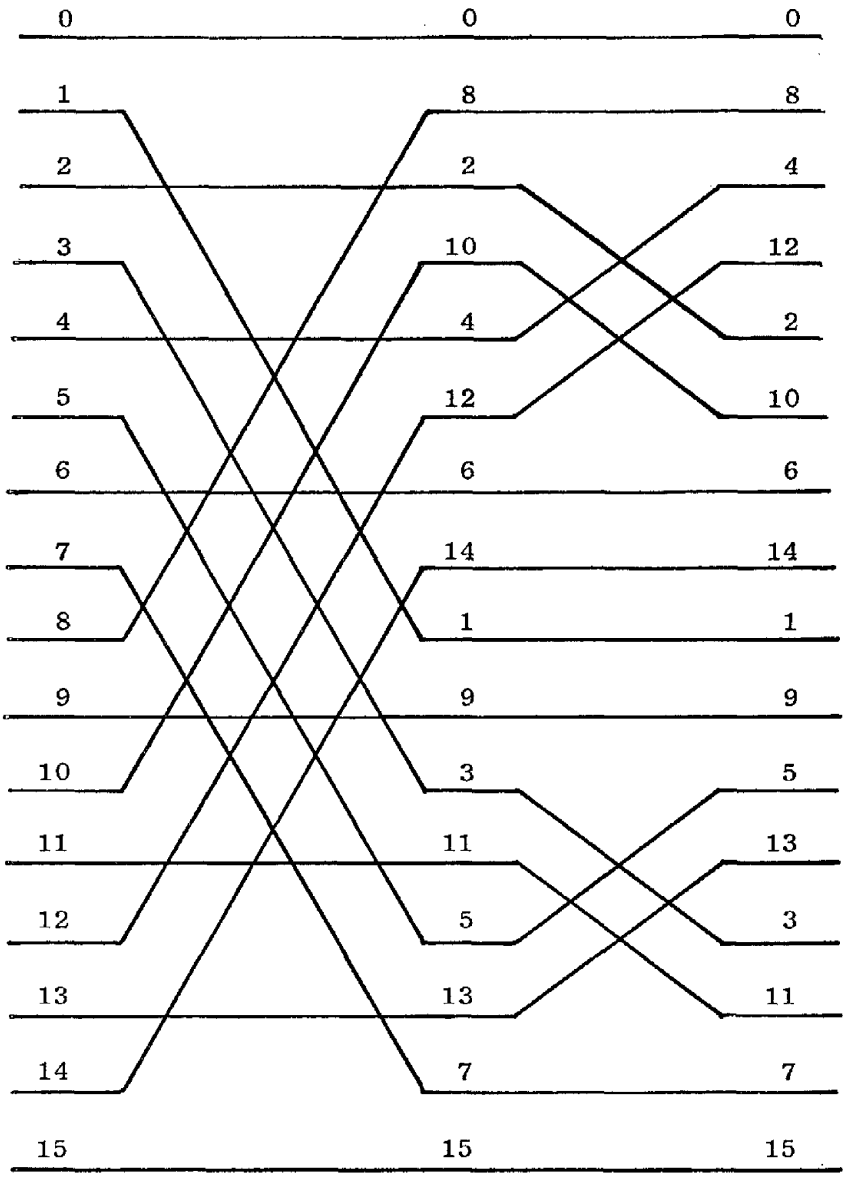


Figure 4.1

Size 16 bit reversal network

```
for k := 0 step 1 until n/2-1 do begin
    MASK1(i) := bit(i,k)∧¬bit(i,n-1-k),   (0 ≤ i ≤ N-1);
    MASK2(i) := MASK1(i-(2^(n-1-k)-2^k)),   (2^(n-1-k)-2^k ≤ i ≤ N-1);
    TEMP(i)  := X(i+2^(n-1-k)-2^k),   (MASK1(i));
    TEMP(i)  := X(i-(2^(n-1-k)-2^k)), (MASK2(i));
    X(i)     := TEMP(i),   (MASK1(i)∨MASK2(i));
end;
```

This algorithm functions similarly to the previous one. MASK1 and MASK2 determine the pairs of items that are to be exchanged. The exchanges are made into the array TEMP and then copied back into X when completed.

We now determine the execution time for this algorithm. n/2 steps are used each requiring 3 routes. This gives

$$R = 3 \cdot n/2 = (3/2) \cdot \log_2 N = O(\log N)$$

$$L = \sum_{k=0}^{n/2-1} 3 \cdot \ell(2^{n-1-k} - 2^k)$$

Proceeding similarly to the analysis for the perfect shuffle, we obtain

$$\ell(2^{n-1-k} - 2^k) = \begin{cases} 2^{n/2-1-k} + 2^k & 0 \le k \le n/2-2 \\ 2^{n/2}/2 & k = n/2-1 \end{cases}$$

This yields

$$L = (3/2) \cdot 2^{n/2} + \sum_{k=0}^{n/2-2} 3 \cdot (2^{n/2-1-k} + 2^k)$$

$$= (11/2) \cdot \sqrt{N} - 9 = O(\sqrt{N})$$

This again yields a speedup of $O(\sqrt{N})$ as compared to the straightforward algorithm.

Although the computation of the control vectors is shown as a part of the main computation loop in both of the previous algorithms, it is easily seen that the control vectors need only be computed once. Precomputation of the control vectors provides for some savings in time if the permutation is performed more than once.

## V. GENERAL ALGORITHM

In the previous two sections we developed algorithms for performing several particular permutations on a machine similar to the ILLIAC IV. These results are of interest, but it is desirable to develop a more general algorithm for performing permutations. In this and subsequent sections we consider performing permutations that are specified as a set of pairs $(i,d(i))$ indicating that element i is to be mapped into element $d(i)$.

To simplify the discussion that follows we introduce some additional notation. The particular notation that we use is due to Lang [1973]. Consider a permutation P: $i \rightarrow d(i)$ where $0 \leq i, d(i) \leq N-1$. $d(i)$ is the destination of i, that is the datum starting in PE i is to end up in PE $d(i)$. We consider permuting only the $d(i)$ since the associated data is moved in exactly the same way. We define $(i,d(i))^k$ to be the location of $d(i)$ after k steps of the algorithm. Clearly $(i,d(i))^0 = i$ and $(i,d(i))^M = d(i)$ where M is the required number of steps in the algorithm. The $k^{th}$ step of the algorithm performs the permutation $P_k$: $(i,d(i))^{k-1} \rightarrow (i,d(i))^k$. The technique that we use is to develop a decomposition of P into $P_M P_{M-1} P_{M-2} \cdots P_2 P_1$ where each $P_k$ can be efficiently realized using the available interconnections and M is "small enough". To describe the particular decomposition that we use, let the binary representation of i be $i_{n-1}i_{n-2}\ldots i_1 i_0$ and the binary representation of $d(i)$ be $d(i)_{n-1}d(i)_{n-2}\ldots d(i)_1 d(i)_0$. At step k we consider pairs of words i' and i" where i' and i" are identical except bit k-1 of i' is 0 and bit k-1 of i" is 1.

Let d(j') be the tag currently in PE i' and d(j'') be the tag currently in PE i''. We perform interchanges based upon $d(j')_{k-1}$ and $d(j'')_{k-1}$ simultaneously for all N/2 pairs of i' and i'' according to Table 5.1 From this it follows that

$$(i,d(i))^k = d(i) \bmod 2^k + 2^k \cdot \lfloor i/2^k \rfloor$$

where $\lfloor x \rfloor$ denotes the integer part of x. This is shown schematically in Figure 5.1. We perform these interchanges for k = 1,2,3,...,n. After n steps all the d(i) are in the correct PE. This is easily seen from the definition of $(i,d(i))^k$.

In the table describing the execution of this algorithm, Table 5.1, certain entries are marked as not allowed. We now present a theorem that describes the set of permutations that satisfy this requirement.

Theorem: A permutation P: i→d(i) can be performed using the previous algorithm if and only if $Q_k$: $i \rightarrow (i,d(i))^k$ is a permutation for $1 \le k \le n$.

Proof: See Lawrie [1973].

We now give the actual algorithm for performing the permutation. The array X contains the d(i) currently in each PE and the arrays TEMP, MASK1, and MASK2 are temporary storage and control vectors respectively.

```
for k := 0 step 1 until n-1 do begin
     MASK1(i) := ¬bit(i,k)∧bit(X(i),k)∧¬bit(X(i+2^k),k),
                      (0 ≤ i ≤ N-1-2^k);
     MASK2(i) := MASK1(i-2^k), (2^k ≤ i ≤ N-1);
     TEMP(i)  := X(i+2^k), (MASK1(i));
     TEMP(i)  := X(i-2^k), (MASK2(i));
     X(i)     := TEMP(i), (MASK1(i)∨MASK2(i));
end;
```

Table 5.1

| $d(j')_{k-1}$ | $d(j'')_{k-1}$ | action |
|---|---|---|
| 0 | 0 | not allowed |
| 0 | 1 | do nothing |
| 1 | 0 | exchange d(j') and d(j'') |
| 1 | 1 | not allowed |

Decision table for general algorithm.

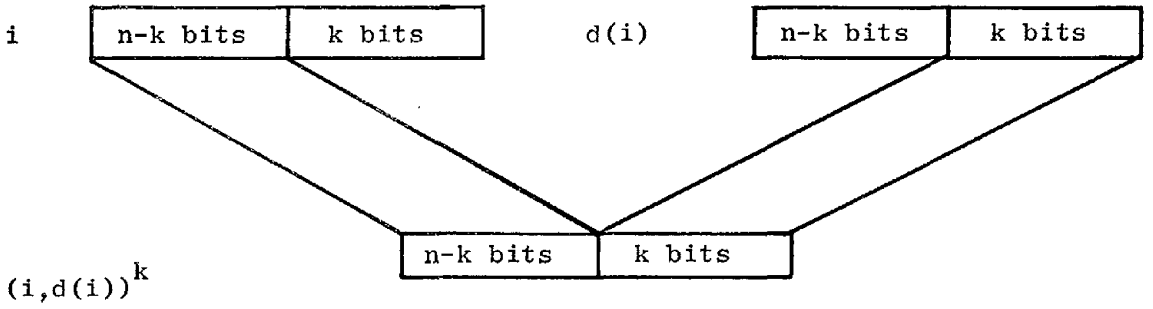


Figure 5.1

Determination of $(i,d(i))^k$

This algorithm functions the same as the last two except that different pairs of elements are exchanged at every step.

We now evaluate the execution time of the algorithm. The algorithm takes n steps each requiring 4 routes. This yields

$$R = 4n = 4 \cdot \log_2 N = O(\log N)$$

$$L = \sum_{j=0}^{n-1} 4 \cdot \ell(2^j)$$

Similarly to previous cases we obtain

$$\ell(2^j) = \begin{cases} 2^j & 0 \le j \le n/2-1 \\ 2^{j-n/2} & n/2 \le j \le n-1 \end{cases}$$

Substituting gives

$$L = 4 \cdot \sum_{j=0}^{n/2-1} 2^j + 4 \cdot \sum_{j=n/2}^{n-1} 2^{j-n/2}$$

$$= 8 \cdot (\sqrt{N}-1) = O(\sqrt{N})$$

## VI. P-ORDERED VECTORS

One of the main considerations in programming the ILLIAC IV is the choice of data structure. From the standpoint of minimizing wasted space the packed storage scheme is very attractive. A characteristic of this storage scheme is that when accessing columns of an array stored packed by rows the data comes out in permuted order. These particular permutations have been called p-ordered vectors by Swanson [1973]. In this section we consider an algorithm for unscrambling p-ordered vectors into normal order.

For this unscrambling we use the previous algorithm. To use this algorithm we must show that no conflicts can arise. For p-ordered vectors with p relatively prime to N the $i^{th}$ element of a vector is in PE $(p \cdot i) \bmod N$. We choose q such that $(p \cdot q) \bmod N = 1$. That such a q must exist is easily shown. With this q we can unscramble the vector by

$$d(i) = (q \cdot i) \bmod N$$

To show that this can be done without conflicts we must show that $Q_k: i \rightarrow (i, d(i))^k$ is a permutation for all k. We do this by showing that

$$(i_1, d(i_1))^k = (i_2, d(_2))^k \Rightarrow i_1 = i_2$$

We proceed as follows

i) $d(i_1) \bmod 2^k + 2^k \cdot \lfloor i_1/2^k \rfloor = d(i_2) \bmod 2^k + 2^k \cdot \lfloor i_2/2^k \rfloor$

ii) $i) \Rightarrow d(i_1) \bmod 2^k = d(i_2) \bmod 2^k$

iii) $d(i) \bmod 2^k = (q \cdot i) \bmod 2^k$

iv) $ii) \Rightarrow (q \cdot i_1) \bmod 2^k = (q \cdot i_2) \bmod 2^k$

v) $iv) \Rightarrow q \cdot (i_1 \bmod 2^k) = q \cdot i_2 \bmod 2^k)$

vi) $i) \Rightarrow 2^k \cdot \lfloor i_1/2^k \rfloor = 2^k \cdot \lfloor i_2/2^k \rfloor$

vii) $vi) \Rightarrow i_1 - i_1 \bmod 2^k = i_2 - i_2 \bmod 2^k$

viii) $vii) \Rightarrow q \cdot i_1 - q \cdot (i_1 \bmod 2^k) = q \cdot i_2 - q \cdot (i_2 \bmod 2^k)$

ix) $v, viii) \Rightarrow q \cdot i_1 = q \cdot i_2$

x) $ix) \Rightarrow i_1 = i_2$

Since this shows that $Q_k$: $i \rightarrow (i, d(i))^k$ is a permutation for all k we can use the algorithm of the previous section and know that it will function properly. Thus the unscrambling of p-ordered vectors requires

$$R = O(\log N) \quad \text{and}$$

$$L = O(\sqrt{N})$$

## VII. UNIVERSAL ALGORITHM

The main drawback to the general algorithm previously described is the requirement that $Q_k$: $i \rightarrow (i,d(i))^k$ be a permutation for all k. It is fairly easy to devise permutations that are important but fail to satisfy these requirements. In fact, both the perfect shuffle and bit reversal are permutations of this class. The restriction on $Q_k$ was removed in Lang [1973] by providing queues for the conflicts at each PE. It is easily shown that for our interconnection scheme we have

$$R = O(\sqrt{N}) \text{ and } L = O(N)$$

if we attempt to use this method. This is not a suitable solution. We choose instead to make use of an entirely different approach.

Stone [1971] describes implementing the bitonic sorting algorithm of Batcher [1968] on a parallel computer. This method was previously proposed for interprocessor communications in Rohrbacher [1966]. In this section we show that this algorithm can be utilized to perform permutations efficiently on a computer with the interconnection scheme under consideration.

To perform permutations we use the Batcher algorithm to sort the d(i). Since all d(i) are distinct and $0 \leq d(i) \leq N-1$, it is easily seen that this method will yield the correct permutation. As before, the algorithm presented permutes only the d(i).

The Batcher algorithm consists of doing comparison-exchange operations on items in PEs whose indices differ in only one bit. These comparison-exchanges are done on bits in the order

$$i_0, i_1, i_0, i_2, i_1, i_0, \ldots, i_{n-1}, i_{n-2}, \ldots i_1, i_0$$

The algorithm to implement this is very similar to the previous algorithm. The additional complexity is due to a peculiarity of the Batcher algorithm. A comparison-exchange operation may order its outputs in either ascending or descending order. A set of mask bits must be calculated to determine the appropriate mode for each comparison-exchange operation. These mask bits are stored in the array MASK. The method used for computing these mask bits is not that originally used by Stone, but is due to Knuth [1973, p. 237].

The algorithm we now present is derived directly from the above description.

```
for j := 0 step 1 until n-1 do begin
    MASK(i) := bit(i,j+1), (0 ≤ i ≤ N-1);
    for k := j step -1 until 0 do begin
        MASK1(i) := ¬bit(i,k)∧[(X(i+2^k) > X(i)) ⊕ MASK(i)],
                    (0 ≤ i ≤ N-1-2^k);
        MASK2(i) := MASK1(i-2^k), (2^k ≤ i ≤ N-1);
        TEMP(i)  := X(i+2^k),  (MASK1(i));
        TEMP(i)  := X(i-2^k),  (MASK2(i));
        X(i)     := TEMP(i),   (MASK1(i)∨MASK2(i));
    end;
end;
```

We now evaluate the execution time of the algorithm.

$$R = 4 \cdot \sum_{k=0}^{n-1} (n-k) = 2n^2 + 2n$$

$$= 2 \cdot \log_2^2 N + 2 \cdot \log_2 N = O(\log^2 N)$$

$$L = 4 \cdot \sum_{k=0}^{n-1} (n-k) \cdot \ell(2^k) = 2n \cdot 2^{n/2} - 6n + 16 \cdot 2^{n/2} - 16$$

$$= 2\sqrt{N} \log_2 N - 6 \cdot \log_2 N + 16\sqrt{N} - 16 = O(\sqrt{N} \log N)$$

This algorithm allows us to perform any arbitrary permutation in a time of $O(\sqrt{N} \log N)$. It was shown in Stone [1972] that it is not possible to perform these permutations in less than $O(\log N)$ time using any interconnection scheme with the same number of connections per processor. This shows that the maximum speedup obtained by any interconnection scheme with the same number of connections per processor as compared to ILLIAC IV-type interconnections is only $O(\sqrt{N})$.

An algorithm that could possibly be somewhat faster can be developed by combining the algorithm of Section V with the previous algorithm in the following manner. First apply the algorithm of Section V. Assume that the first conflict occurs when considering bit k. The preceeding steps have partitioned the data into $2^k$ groups in such a way that it is not necessary to consider moving data between groups. To complete the permutation we apply the previous algorithm to all groups in parallel. Clearly the

execution time of such an algorithm would be bounded by the execution times of the two individual algorithms. Since there is additional bookkeeping to be done in this composite algorithm, whether or not this algorithm would actually be faster would depend on how often the permutation being performed partially satisfies the requirements of the algorithm of Section V.

## VIII. CONCLUSIONS

In this paper we have presented a number of algorithms for performing permutations on a parallel computer with an interconnection scheme similar to that of ILLIAC IV. Particular algorithms are developed for the perfect shuffle and bit reversal. Other algorithms are developed for more general classes of permutations. All of these algorithms require less than linear time for execution. These new algorithms show that the speedup obtained by recently proposed interconnection network as compared to ILLIAC IV-type interconnections is less than was thought.

Acknowledgement

The author would like to thank Professor Harold S. Stone and Mr. Tomás Lang for their many helpful discussions on this topic.

BIBLIOGRAPHY

Batcher, K. E., "Sorting networks and their applications," 1968 Spring Joint Computer Conference, AFIPS Proceedings, vol. 32. Washington, D.C.: Thompson, pp. 307-314, 1968.

Flynn, M. J., "Very high-speed computing systems," Proceedings of the IEEE, vol. 54, no. 12, pp. 1901-1909, December 1966.

Knuth, D. E., The Art of Computer Programming, Vol. 3, Searching and Sorting. Reading, Massachusetts: Addison-Wesley, 1973.

Kogge, P. M., "Parallel algorithms for the efficient solution of recurrence problems," Rep. 43, Digital Systems Laboratory, Stanford University, Stanford, California, September 1972.

Lang, T., "Interconnections between processors and memory modules using the shuffle-exchange network," Submitted to IEEE Transactions on Computers (Available through IEEE Computer Society Repository, no. R74-19).

Lawrie, D. E., "Memory-processor connection networks," Ph.D. thesis, Rep. 557, Department of Computer Science, University of Illinois, Urbana, Illinois, February 1973.

Pease, M. C., "An adaptation of the fast Fourier transform for parallel processing," Journal of the ACM, vol. 15, no. 2, pp. 252-264, April 1968.

Polge, R. J. et al., "Fast computational algorithms for bit reversal," IEEE Transactions on Computers, vol. C-23, no. 1, pp. 1-9, January 1974.

Rohrbacher, D. L., "Advanced computer organization study," vols. I and II, Goodyear Aerospace Corp., Rep. GER-12314, April 1966 (DDC accession nos. AD631870 and AD631871).

Stone, H. S., "Parallel processing with the perfect shuffle," IEEE Transactions on Computers, vol. C-20, no. 2, pp. 153-161, February 1971.

Stone, H. S., "Dynamic memories with enhanced data access," IEEE Transactions on Computers, vol. C-21, no. 4, pp. 359-366, April 1972.

Stone, H. S., "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," Journal of the ACM, vol. 20, no. 1, pp. 27-38, January 1973.

Swanson, R. C., "Interconnections for parallel memories to unscramble p-ordered vectors," Rep. 72, Digital Systems Laboratory, Stanford University, Stanford, California, May 1973.