A SELF-ROUTING BENES NETWORK AND PARALLEL PERMUTATION ALGORITHMS

D. Nassimi and S. Sahni

Reprinted from IEEE Transactions on Computers, Vol. C-30, No. 5, May 1981
0018-9340/81/0500-0332$00.75

A Self-Routing Benes Network and Parallel Permutation Algorithms

DAVID NASSIMI AND SARTAJ SAHNI, MEMBER, IEEE

***Abstract*—A Benes permutation network capable of setting its own switches dynamically is presented. The total switch setting and delay time for the N input/output self-routing network is $0(\log N)$. It is shown that the network is capable of performing a rich class of permutations. The self-routing scheme leads to efficient $0(\log N)$ parallel algorithms to perform the same class of permutations on cube connected and perfect shuffle computers.**

***Index Terms*—Benes network, bit-permute-complement permutations, complexity, cube connected computer, inverse omega permutations, omega permutations, perfect shuffle computer.**

Manuscript received July 9, 1979; revised May 21, 1980. This work was supported in part by the National Science Foundation under Grant MCS 78-15455.

D. Nassimi is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

S. Sahni is with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

I. INTRODUCTION

THE $N = 2^n$ input/output Benes permutation network $B(n)$ is shown in Fig. 1. Each of the smaller boxes represents a binary (or two state) switch (Fig. 2). The states of a binary switch will be referred to as *zero* and *one*. The network $B(n)$ consists of a stage of $N/2$ binary switches followed by two copies of the network $B(n-1)$, followed by another stage of $N/2$ binary switches. $B(1)$ is simply the binary switch of Fig. 2. The number of stages in $B(n)$ is therefore $2 \log N - 1$ (all logarithms in this paper are base 2). The total number of binary switches in the network is $N \log N - N/2$. The Benes network is capable of connecting its inputs to its outputs according to any of $N!$ permutations. The network finds application as a subnetwork of a generalized connection network [9]. The Benes network could also be used in an SIMD (single

0018-9340/81/0500-0332$00.75

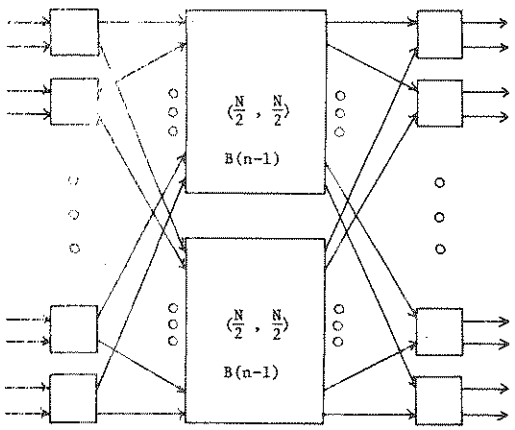

Fig. 1. Benes (N,N)-permutation network $B(n)$; $N = 2^n$.

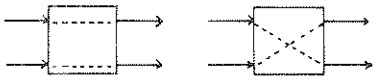

Fig. 2. The two possible states of a binary switch. (a) State 0. (b) State 1.

instruction stream, multiple data stream) computer to interconnect the N processing elements (PE's). In this configuration PE(i), $0 \le i \le N-1$, is connected to both input i and output i of $B(n)$. In an alternate configuration of an SIMD computer, the Benes network can be used to connect the N PE's to N memory modules, where PE(i) and memory module i are respectively connected to input i and output i of $B(n)$.

Since the Benes network has $0(\log N)$ stages, there is an $0(\log N)$ delay in transmitting data from the input terminals of the network to the output terminals. In practice, however, the time needed to accomplish an arbitrary input to output permutation is considerably larger as the switch settings must be determined first, and this is not a simple problem. Let $D = (D_0, D_1, \cdots, D_{N-1})$ be an arbitrary permutation of $(0, 1, \cdots, N-1)$. The *setup problem* is to compute from D the state of all switches in $B(n)$ so that input i is connected to output D_i, $0 \le i \le N-1$. The best known setup algorithm runs in $0(N \log N)$ time on a single-processor machine (see Waksman [10]).

The setup problem for the Benes network can be solved more efficiently by using an SIMD computer. (To avoid confusion, consider the Benes network completely detached from the SIMD machine.) We give the permutation $D = (D_0, D_1, \cdots, D_{N-1})$ to the machine. It returns $N \log N - N/2$ bits, where each bit is the state of a switch in the Benes network. Nassimi and Sahni [7] have developed parallel set-up algorithms using four different SIMD models. In these models each PE has a private local memory. The models differ in the way the PE's are interconnected. Let N' be the number of PE's. The four SIMD models are as follows.

1) Completely Interconnected Computer (CIC): In this model every pair of PE's is directly connected. Let $R(i)$ be a data word in the "routing register" of PE(i), $0 \le i \le N'-1$. Any permutation of $(R(0), R(1), \cdots, R(N'-1))$ may be realized on this model in a single step.

2) Mesh Connected Computer (MCC): In this model the PE's are logically arranged as in a two-dimensional array $A(0: m-1, 0: m-1)$, where $N' = m^2$. The PE at location $A(i, j)$ is connected to the PE's at locations $A(i \pm 1, j)$ and $A(i, j \pm 1)$, provided they exist.

3) Cube Connected Computer (CCC): Assume that $N' = 2^n$ and let $i_{n-1} \cdots i_0$ be the binary representation of i for $i \in [0, N'-1]$. Let $i^{(b)}$ be the number whose binary representation is $i_{n-1} \cdots i_{b+1} \bar{i}_b i_{b-1} \cdots i_0$, where $\bar{i}_b$ is the complement of i_b and $0 \le b < n$. In the cube model PE(i) is connected to PE($i^{(b)}$), $0 \le b < n$.

4) Perfect Shuffle Computer (PSC): Let N', n, i, and $i^{(b)}$ be as in the cube model. In the perfect shuffle model, PE(i) is connected to PE($i^{(0)}$), PE($i_{n-2} i_{n-3} \cdots i_0 i_{n-1}$) and PE($i_0 i_{n-1} i_{n-2} \cdots i_1$). These three connections are, respectively, called *exchange*, *shuffle*, and *unshuffle*.

It should be noted that in a CIC, MCC, CCC, and PSC, each PE is directly connected to $N'-1$, 4, $\log N'$, and three other PE's, respectively.

Let N be the number of inputs/outputs of the Benes network. The parallel setup algorithms developed by Nassimi and Sahni [7] run in $0(\log^2 N)$, $0(N^{1/2})$, and $0(k \log^3 N)$ time, respectively, on an N — PE CIC, $N^{1/2} \times N^{1/2}$ MCC, and a CCC or PSC with $N^{1+1/k}$ PE's, $1 \le k \le \log N$. So, even with these parallel setup algorithms, the time needed to perform an arbitrary permutation on the Benes network is dominated by the setup time.

Permutation networks have been proposed in the literature that are easier to set up than the Benes network. These, however, either have much larger transmission delay or use considerably more (binary) switches. For example, a full crossbar is trivial to set up, but uses $0(N^2)$ switches. The permutation network of Lang and Stone [3] is easy to set up but has a delay of $0(N^{1/2})$. Batcher's sorting network [11] is self-routing, but has $0(\log^2 N)$ delay and $0(N \log^2 N)$ switches. Nassimi and Sahni [13] have proposed a class of easy to set up permutation networks. Let $N = 2^n$ be the number of inputs and outputs. For a given M, $M \in \{2, 4, \cdots, N\}$, the permutation network of [13] has $0(NM(1 + \log N - \log M) \log N/\log M)$ binary switches, $0(\log^2 N/\log M)$ transmission delay, and can be set up in $0(\log^2 N/\log M)$ time.

In this paper we investigate the possibility of rapidly obtaining the switch settings of the Benes network for certain classes of permutations. We show that by providing a "destination tag" with each signal and by adding some simple logic to each switch in the Benes network, it is possible for each switch to determine its own setting dynamically (i.e., when it receives the incoming signal). The resulting network can perform certain permutations in $0(\log N)$ time (including the set-up time). We analyze the set of permutations F realizable on our "self-routing" network. We demonstrate the richness of the set F by showing that it includes most classes of permutations studied in the parallel processing literature.

To describe our simple control logic, let D_i be the "destination tag" on input terminal i, $0 \le i \le N-1$. Recall that $(D_0, D_1, \cdots, D_{N-1})$ is a permutation of $(0, 1, \cdots, N-1)$. The data at input terminal i is to be routed to output terminal D_i. The switch settings are determined from the binary representation of D_i. If an $N = 2^n$ input/output Benes network is being used, then there are $2n-1$ stages of switches whose

settings need to be obtained. Let the stages be numbered 0 through $2n-2$. The state of a switch in stage b or stage $2n-2-b$, $0 \le b \le n-1$, is determined by bit b of the destination tag of its upper input (see Fig. 3). If bit b is 0 the switch is set to state 0, otherwise to state 1. Fig. 4 illustrates the switch settings obtained by this procedure for $B(3)$. The permutation being performed is a bit reversal (i.e., input i is sent to output terminal j, where the binary representation of j is the reverse of that of i). The destination (in binary) for each switch input is given on the respective input line. As we shall see, all permutations cannot be performed using this scheme.

The switch setting scheme just described is quite similar to that used by Lawrie [4] to obtain the settings for his omega network. The number of switches and the delay in our self-routing network are both about twice the corresponding figures in a "self-routing" omega network. However, the number of permutations realizable on our network (i.e., the cardinality of the set F) is much larger than that of an omega network. Furthermore, if we allow the added capability of disabling the self-setting logic in our network and set up the switches externally, then the network can realize all $N!$ permutations. The same is not true of an omega network.

The problem of obtaining fast setup algorithms for the Benes network has been studied previously. Most notably, Lenfant [5] has proposed efficient set-up algorithms for five classes of permutations. The setup scheme we have discussed is simpler than each of the five different algorithms proposed by Lenfant for his five classes. In addition, his five classes of permutations are a small subset of our class F. In Section II we characterize the class F of permutations that can be performed on our self-routing network.

The self-routing Benes network can be easily "simulated" on an SIMD machine with a fixed interconnection network. Using this approach, we can perform any permutation in class F in $0(\log N)$ steps on an N - PE CCC or PSC. Using the same approach on an $N^{1/2} \times N^{1/2}$ MCC yields an efficient $0(N^{1/2})$ permutation algorithm for class F. The details of the simulation approach are discussed in Section III.

II. The Class F

Fig. 4 gives an example permutation (i.e., bit reversal) that can be performed on $B(3)$ using the switch setting scheme discussed in Section I. Fig. 5 shows that the permutation $D = (1, 3, 2, 0)$ cannot be performed on $B(2)$ using this switch setting scheme.

Let $F(n)$ denote the set of permutations performable on $B(n)$ using the self-routing scheme of Section I. In this section we are interested in a characterization of the class $F = \cup F_{n=1}^{\infty}(n)$. By the permutation $D = (D_0, D_1, \cdots, D_{N-1})$ we shall mean a permutation in which input i is sent to output D_i, $0 \le i < N$. Hence, D_i is the destination tag of input i.

Given an integer i, let $(i)_j$ denote bit j in the binary representation of i, where $(i)_0$ is the least-significant bit. Thus, if i is in the range $[0, 2^n - 1]$, then $i = (i)_{n-1}(i)_{n-2} \cdots (i)_0$. The notation $(i)_{j:k}$, where $j \ge k$, shall denote the integer with binary representation $(i)_j(i)_{j-1} \cdots (i)_k$. For example, if $i = 10110$, then $(i)_{3:1} = 011$ (=3). Note that $(i)_{j:j} = (i)_j$.

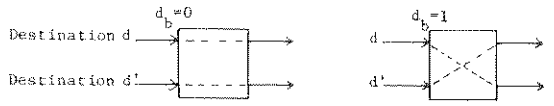

Fig. 3. The control of a switch at stage b or stage $2n-2-b$, $0 \le b \le n-1$, using the destination tags.

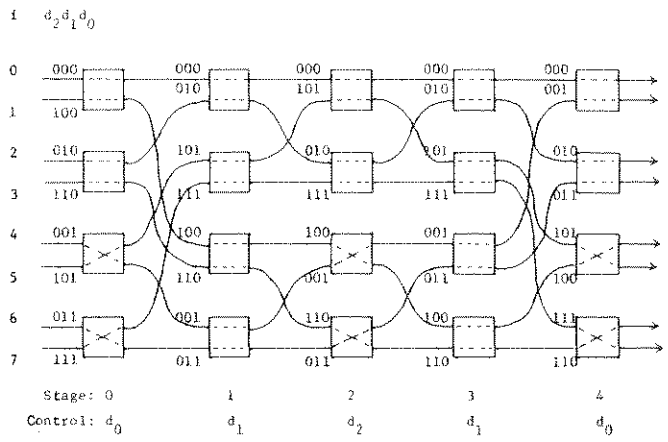

Fig. 4. Bit-reversal permutation.

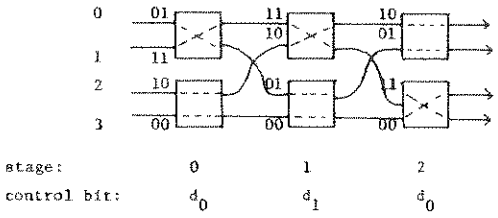

Fig. 5. A permutation not in F.

Let the switches in each stage of $B(n)$, $N = 2^n$, be numbered 0 to $N/2-1$, top to bottom. Recall that the stages are numbered 0 to $2n-2$, left to right. Let U_i be the destination tag of the upper output of switch i in stage 0. And let L_i be the destination tag of the lower output of the same switch. Note that the destination tags of the upper and lower inputs to this switch are D_{2i} and D_{2i+1}, respectively. The state of this switch is determined by $(D_{2i})_0$; that is, bit 0 of the destination tag of the upper input of the switch (see Fig. 3). So, it follows that

$$U_i = \begin{cases} D_{2i} & \text{if } (D_{2i})_0 = 0 \\ D_{2i+1} & \text{if } (D_{2i})_0 = 1 \end{cases} \quad (1)$$

$$L_i = \begin{cases} D_{2i} & \text{if } (D_{2i})_0 = 1 \\ D_{2i+1} & \text{if } (D_{2i})_0 = 0 \end{cases} \quad (2)$$

for $0 \le i \le N/2-1$.

The following theorem tells us whether or not $D \in F(n)$.

Theorem 1: The permutation $D = (D_0, \cdots, D_{N-1})$, where $N = 2^n$ and $n \ge 2$, is in $F(n)$ iff $U = ((U_0)_{n-1:1}, \cdots, (U_{N/2-1})_{n-1:1})$ and $L = ((L_0)_{n-1:1}, \cdots, (L_{N/2-1})_{n-1:1})$ are both permutations in $F(n-1)$. U_i and L_i are as given by (1) and (2).

Proof: Let the upper $B(n-1)$ subnetwork in our self-routing $B(n)$ network be denoted as $B_u(n-1)$. (See Fig. 1.) And let $B_l(n-1)$ denote the lower $B(n-1)$ subnetwork. Let us adopt the numbering $0, 1, \cdots, N/2-1$ for the inputs and outputs of each of $B_u(n-1)$ and $B_l(n-1)$. It should be easy to see that $(U_0, \cdots, U_{N/2-1})$ is the ordered set of destination tags of the inputs to $B_u(n-1)$, and $(L_0, \cdots, L_{N/2-1})$ is the

ordered set of destination tags of the inputs to $B_l(n-1)$. Now if U is a permutation and is in $F(n-1)$, then $B_u(n-1)$ will perform the permutation U on its $N/2$ inputs. Similarly, if L is a permutation and in $F(n-1)$, then $B_l(n-1)$ will route its $N/2$ inputs to its $N/2$ outputs according to the permutation L. That is, input i of $B_u(n-1)$ is routed to output $(U_i)_{n-1:1}$ of $B_u(n-1)$; and input i of $B_l(n-1)$ is routed to output $(L_i)_{n-1:1}$ of $B_l(n-1)$.

Now, consider switch j in the last stage of $B(n)$. (The switches in the last stage are numbered $0, 1, \cdots, N/2-1$.) The upper and lower inputs to this switch, respectively, come from output j of $B_u(n-1)$ and output j of $B_l(n-1)$. Thus, the destination tags D_u and D_l of the two inputs to the jth switch in the last stage are such that $(D_u)_{n-1:1} = (D_l)_{n-1:1} = j$. Since D is a permutation, it follows that $(D_u)_0 \neq (D_l)_0$ (as otherwise, $D_u = D_l$) and so one of D_u and D_l equals $2j$ and the other $2j+1$. Setting switch j (of the last stage) according to $(D_u)_0$ correctly routes its two inputs to their final destinations.

To prove the reverse, one may easily establish that if $D \in F(n)$, then L and U are permutations and $L \in F(n-1)$ and $U \in F(n-1)$. □

Having established a necessary and sufficient condition for a permutation to be realizable by our self-routing Benes network, we proceed to show that most of the permutation classes that have been studied in the literature are members of F.

First, consider the class BPC of bit-permute-complement permutations studied by Nassimi and Sahni [6]. A permutation in BPC(n) is specified by providing an n tuple $A = (A_{n-1}, \cdots, A_0)$, where $|A| = (|A_{n-1}|, \cdots, |A_0|)$ is a permutation of $(n-1, n-2, \cdots, 0)$. The vector A specifies a permutation on $N = 2^n$ inputs. The destination D_i of input i is obtained as follows:

$$(D_i)_{|A_j|} = \begin{cases} (i)_j & \text{if } A_j \geq 0 \\ 1 - (i)_j & \text{if } A_j < 0. \end{cases} \tag{3}$$

In A we distinguish between $+0$ and -0. We use $-0 < 0$. In words, the destination address D_i for input i is obtained from i by first complementing a subset of bits in the binary representation of i and then permuting the bits. The vector $(A_{n-1}, \cdots, \text{sign}(A_0))$ specifies which bits of i are complemented: Bit j of i is complemented iff $A_j < 0$. Then the vector $\subset(|A_{n-1}|, \cdots, |A_0|)$ says how to permute the bits: The bit from position j goes to position $|A_j|$. Thus, bit j of i, or its complement, becomes bit $|A_j|$ of the destination address D_i, $0 \leq i \leq N-1$. For example, consider $A = (0, -1, -2)$. The input $i = (i)_2(i)_1(i)_0$ has destination $D_i = (D_i)_2(D_i)_1(D_i)_0 = (\bar{i})_0(\bar{i})_1(i)_2$. Thus, $D_0 = 6$, $D_1 = 2$, $D_2 = 4$, $D_3 = 0$, $D_4 = 7$, $D_5 = 3$, $D_6 = 5$, $D_7 = 1$.

The class BPC(n), $N = 2^n$, only contains $N(\log N)!$ of the possible $N!$ permutations. Nevertheless, many of the permutations encountered in parallel algorithms are included in this class. For example, Lenfant [5] identifies five families of "frequently used bijections" (FUB). Three of his FUB families (namely, $\alpha^{(n)}$, $\beta^{(n)}$, $\gamma^{(n)}$) are included in our BPC(n). Table I gives the A vectors corresponding to several of the more popular permutations in BPC (n). Note that the bit reversal of Fig. 4 is in BPC(3).

It should be pointed out that our self-routing network "sees" the destination tags $D = (D_0, D_1, \cdots, D_{N-1})$. The fact that the permutation D as a member of BPC class has a more compact representation $A = (A_{n-1}, \cdots, A_0)$ is not "known" to the network. Theorem 2 shows that BPC$(n) \subseteq F(n)$. Before proceeding to the proof of this result, we need the result of the next lemma.

TABLE I
EXAMPLE PERMUTATIONS IN BPC(n)

Permutation	Vector Representation A
Matrix Transpose	{n/2-1, ..., 0, n-1, ..., n/2}
Bit Reversal	{0, 1, 2, ..., n-1}
Vector Reversal	{-(n-1), -(n-2), ..., -0}
Perfect Shuffle	{0, n-1, n-2, ..., 1}
Unshuffle	{n-2, n-3, ..., 0, n-1}
Shuffled Row Major	{n-1, n/2-1, n-2, n/2-2, ..., n/2, 0}
Bit Shuffle	{n-1, n-3, ..., 1, n-2, n-4, ..., 0}

Lemma 1: Let $A = (A_{n-1}, \cdots, A_0) \in$ BPC(n), $n > 1$, specify the permutation $D = (D_0, D_1, \cdots, D_{N-1})$, where $N = 2^n$ [see (3)]. Assume that $|A_0| \neq 0$, hence, $|A_k| = 0$ for some $k \in [1, n-1]$. (The existence of k follows from the fact that $(|A_0|, |A_1|, \cdots, |A_{n-1}|)$ is a permutation of $(0, 1, \cdots, n-1)$.) Then, $F1$ and $F2$ as defined below are permutations in BPC$(n-1)$.

1) $F1 = (Q_0, Q_1, \cdots, Q_{N/2-1})$
where

$$Q_i = \begin{cases} (D_{2i})_{n-1:1} & \text{if } (2i)_k = 0 \text{ [if } (2i)_k = (2i)_0\text{]} \\ (D_{2i+1})_{n-1:1} & \text{if } (2i)_k = 1 \text{ [if } (2i+1)_k = (2i+1)_0\text{]}. \end{cases}$$

2) $F2 = (R_0, R_1, \cdots, R_{N/2-1})$
where

$$R_i = \begin{cases} (D_{2i})_{n-1:1} & \text{if } (2i)_k = 1 \text{ [if } (2i)_k \neq (2i)_0\text{]} \\ (D_{2i+1})_{n-1:1} & \text{if } (2i)_k = 0 \text{ [if } (2i+1)_k \neq (2i+1)_0\text{]}. \end{cases}$$

Note that following each condition an equivalent condition is given inside brackets. For example, on the last line it is easy to see that the condition $(2i)_k = 0$ is equivalent to the condition $(2i+1)_k \neq (2i+1)_0$. (Recall that $k \neq 0$).

Proof: Let us represent the permutation $D = (D_0, D_1, \cdots, D_{n-1})$ as a set S of ordered pairs $\langle i, D_i \rangle$:

$$S = \{\langle i, D_i \rangle \mid 0 \leq i \leq N-1\}.$$

Consider the two subsets $G1$ and $G2$ of S defined below:

$$G1 = \{\langle i, D_i \rangle \mid \langle i, D_i \rangle \in S \text{ and } (i)_k = (i)_0\}$$

$$G2 = \{\langle i, D_i \rangle \mid \langle i, D_i \rangle \in S \text{ and } (i)_k \neq (i)_0\}.$$

Note that $|G1| = |G2| = N/2$ and $G1 \cup G2 = S$. Also observe that for any i, $0 \leq i \leq N/2-1$, one of $\langle 2i, D_{2i} \rangle$ and $\langle 2i+1, D_{2i+1} \rangle$ is in $G1$, while the other is in $G2$. Next, define $H1$ and $H2$ as below:

$$H1 = \{\langle (i)_{n-1:1}, (D_i)_{n-1:1} \rangle \mid \langle i, D_i \rangle \in G1\}$$

$$H2 = \{\langle (i)_{n-1:1}, (D_i)_{n-1:1} \rangle \mid \langle i, D_i \rangle \in G2\}.$$

It is easy to see that $H1$ and $H2$, respectively, correspond to $F1$ and $F2$ as defined in the statement of the lemma. That is, $H1 = \{\langle 0, Q_0 \rangle, \cdots, \langle N/2-1, Q_{N/2-1} \rangle\}$ and $H2 = \{\langle 0, R_0 \rangle, \cdots, \langle N/2-1, R_{N/2-1} \rangle\}$.

To show that $H1$ represents a permutation in BPC$(n-1)$, let us first examine $G1$. Let $\langle i, D_i\rangle \in G1$. Recall that $|A_k| = 0, k \in [1, n-1]$. Let $|A_0| = r, r \in [1, n-1]$. From (3) and the fact that $(i)_k = (i)_0$, it follows that

$$(D_i)_0 = \begin{cases} (i)_0 & \text{if } A_k = +0 \\ 1 - (i)_0 & \text{if } A_k = -0 \end{cases}$$

$$(D_i)_r = \begin{cases} (i)_k & \text{if } A_0 = +r \\ 1 - (i)_k & \text{if } A_0 = -r. \end{cases}$$

The remaining bits of D_i may still be computed from (3). In words, bit 0 of D_i is obtained from bit 0 of i. Bits $1, 2, \cdots, n-1$ of D_i are obtained from bits $1, 2, \cdots, n-1$ of i by complementing a subset of the latter bits and then permuting them.

From these results for $G1$ and the fact that for any i exactly one of $\langle 2i, D_{2i}\rangle$ or $\langle 2i+1, D_{2i+1}\rangle$ is in $G1$, it easily follows that $H1$ is a permutation in BPC$(n-1)$ class. More precisely, let

$$\text{LMAG}\ (A_j) = \text{SIGN}\ (A_j) * (|A_j| - 1). \tag{4}$$

The permutation $H1$ (or $F1$) is given by the vector

$$B = (B_{n-2}, \cdots, B_0)$$

where $B_j = \text{LMAG}\ (A_{j+1})$ for $j \neq k-1$, and $B_{k-1} = \text{LMAG}\ (A_0)$.

Similarly, it can be shown that $H2$ (or $F2$) is a permutation and in BPC$(n-1)$. This permutation is defined by the vector

$$C = (C_{n-2}, \cdots, C_0)$$

where $C_j = B_j$ for $j \neq k-1$, and $C_{k-1} = -B_{k-1}$. □

Theorem 2: BPC$(n) \subseteq F(n)$ for all $n, n \geq 1$.

Proof: The proof is by induction on n. Clearly, BPC$(1) = F(1)$. Assume that BPC$(m) \subseteq F(m)$ for all $m, 1 \leq m < n$. We shall show that BPC$(n) \subseteq F(n)$. This will establish the theorem. Let $A = (A_{n-1}, \cdots, A_0)$ specify a permutation $D = (D_0, \cdots, D_{N-1})$ in BPC(n). We consider two cases: 1) $|A_0| = 0$, and 2) $|A_0| \neq 0$.

Case 1: $|A_0| = 0$—If $A_0 = 0$ then $(D_i)_0 = (i)_0, 0 \leq i < N$, and all switches in stage 0 are in state 0. Using the notation of Theorem 1, $U_i = D_{2i}$ and $L_i = D_{2i+1}, 0 \leq i < N/2$. Hence, the permutations U and L to be performed by the $B(n-1)$ networks are given by $A' = (A'_{n-1}, \cdots, A'_0)$, where $A'_j = \text{LMAG}\ (A_{j+1})$ and LMAG is defined in (4). Clearly, A' defines a permutation in BPC$(n-1)$. From the induction hypothesis, BPC$(n-1) \subseteq F(n-1)$ and so from Theorem 1 it follows that $D \in F(n)$.

If $A_0 = -0$ then $(D_i)_0 = 1 - (i)_0, 0 \leq i < N$, and all switches in stage 0 are in state 1. The permutations U and L are again given by A' which defines a permutation in BCP$(n-1)$. So, $D \in F(n)$.

Case 2: $|A_0| \neq 0$—Let k be as defined in Lemma 1. We consider two possibilities: $A_k = 0$ and $A_k = -0$.

a) $A_k = 0$: In this case $(D_{2i})_0 = (2i)_k, 0 \leq i \leq N/2 - 1$. Recall that the state of switch i in stage $0, 0 \leq i \leq N/2 - 1$, is determined by $(D_{2i})_0$. Hence, the state of this switch $= (2i)_k$. The destination tags of the upper and lower outputs of this switch are U_i and L_i. So, from the definitions of Q_i and R_i in Lemma 1 we see that $(U_i)_{n-1:1} = Q_i$ and $(L_i)_{n-1:1} = R_i, 0 \leq i \leq N/2 - 1$. Therefore, $U = F1 \in$ BPC$(n-1)$ and $L = F2 \in$ BPC$(n-1)$. From the induction hypothesis and Theorem 1 it follows that $D \in F(n)$.

b) $A_k = -0$: In this case $(D_{2i})_0 = 1 - (2i)_k, 0 \leq i \leq N/2 - 1$. So, the state of switch i in stage $0 = 1 - (2i)_k$. Now we see that $(U_i)_{n-1:1} = R_i$ and $(L_i)_{n-1:1} = Q_i, 0 \leq i \leq N/2 - 1$. So, $U = F2 \in$ BPC$(n-1)$ and $L = F1 \in$ BPC$(n-1)$. Therefore, $D \in F(n)$. □

We next turn our attention to the class of omega (Ω) and inverse omega ($I\Omega$) permutations. Lawrie [4] has defined the class of omega permutations to consist of exactly those permutations realizable by his omega network. A permutation $D = (D_0, D_1, \cdots, D_{N-1}), N = 2^n$, is an $\Omega(n)$ permutation iff for every i and $j, 0 \leq i < N, 0 \leq j < N$ and $i \neq j$, the following is true:

$$(D_i)_{n-1:b}\ (i)_{b-1:0} \neq (D_j)_{n-1:b}\ (j)_{b-1:0}, \quad 1 \leq b \leq n-1.$$

D is an $I\Omega(n)$ permutation [4] iff for every i and $j, 0 \leq i < N$, $0 \leq j < N$, and $i \neq j$, the following is true:

$$(i)_{n-1:b}\ (D_i)_{b-1:0} \neq (j)_{n-1:b}\ (D_j)_{b-1:0}, \quad 1 \leq b \leq n-1.$$

Inverse omega permutations are realizable using an omega network backwards. Lawrie [4] lists several specific omega permutations that are useful in matrix computations. These correspond to fetching array elements by rows, columns, forward diagonals, backward diagonals, etc. Some interesting permutations contained in $I\Omega(n)$ are:

1) cyclic shift: Here, $D_i = (i + k)\ mod\ N, 0 \leq i < N$ and k is a fixed constant.

2) p-ordering: $D_i = (p \cdot i)\ mod\ N, 0 \leq i < N$, where p is an odd integer.

3) inverse p-order: This just unscrambles a p-ordering and is done by a q-ordering, where $(p \cdot q)\ mod\ N = 1$.

4) p-ordering and cyclic shift: $D_i = (p \cdot i + k)\ mod\ N, 0 \leq i < N$, where p is an odd integer and k an arbitrary integer.

5) cyclic shifts within segments: Given $r \in [1, n-1]$ and an arbitrary integer k, this permutation maps i to $D_i, 0 \leq i \leq 2^n - 1$, where $(D_i)_{n-1:r} = (i)_{n-1:r}$ and $(D_i)_{r-1:0} = ((i)_{r-1:0} + k)\ mod\ 2^r$. That is, a cyclic shift of k is performed within each segment of size 2^r.

6) conditional exchange: Given $k \in [1, n-1]$, this permutation is defined as $(D_i)_{n-1:1} = (i)_{n-1:1}$ and $(D_i)_0 = (i)_0 \oplus (i)_k$, where $\oplus$ is EXCLUSIVE-OR. In words, the elements of each pair $(2i, 2i+1)$ are exchanged iff bit k of $2i$ is $= 1$.

The set of permutations defined in 4) and 5) above are Lenfant's FUB families $\lambda^{(n)}$ and $\delta^{(n)}$, respectively [5]. The "conditional exchange" corresponds to Lenfant's $\eta^{(n)}$. (See Table I of [5].) It is interesting to note that all of the above $I\Omega(n)$ permutations are also members of $\Omega(n)$.

One may easily verify that there exist $\Omega(n)$ and $I\Omega(n)$ permutations that are not in BPC(n). For example, cyclic shift is not in BPC(n) unless, of course, $k\ mod\ N = 0$. Also, BPC(n) contains permutations not in $\Omega(n)$ nor $I\Omega(n)$. More specifically, every BPC permutation specified by $A = (A_{n-1}, \cdots, A_0)$, where $|A_j| \neq j$ for at least one j is in neither $\Omega(n)$ nor $I\Omega(n)$.

We leave it to the reader to confirm these noncontainment statements. The reader is referred to Lawrie [4] and Orcutt [8] for further examples and properties of Ω and $I\Omega$ permutations. Our next theorem shows that $I\Omega(n) \subseteq F(n)$. This result is not surprising as the first n stages of $B(n)$ correspond to an inverse omega network except for some rearrangement of switches. Similarly, the last n stages of $B(n)$ correspond to an omega network.

Theorem 3: $I\Omega(n) \subseteq F(n)$, $n \geq 1$.

Proof: The proof is by induction on n. When $n = 1$, $I\Omega(1) = F(1)$. Assume that $I\Omega(m) \subseteq F(m)$, $1 \leq m < n$. Let $D = (D_0, D_1, \cdots, D_{N-1})$ be any permutation in $I\Omega(n)$. In stage zero of $B(n)$, inputs $2i$ and $2i + 1$ are connected to switch i. Since $(2i)_{n-1:1} = (2i + 1)_{n-1:1}$ and $D \in I\Omega(n)$, it follows that $(D_{2i})_0 \neq (D_{2i+1})_0$. Let U_i and L_i be as in Theorem 1. From the switch setting scheme, it follows that $(U_i)_0 = 0$ and $(L_i)_0 = 1$, $0 \leq i < N/2$. Consequently, U and L both define permutations. It is now an easy matter to see that $U \in I\Omega(n-1)$ and $L \in I\Omega(n-1)$. □

Since we have shown that $BPC(n) \subseteq F(n)$ and $I\Omega(n) \subseteq F(n)$, we conclude that all of Lenfant's five families of "frequently used bijections" are included in $F(n)$. This is because, as commented earlier, three of the FUB families are in BPC and the remaining two in $I\Omega$.

Unfortunately, not all $\Omega(n)$ permutations are in $F(n)$. For example, $D = (1, 3, 2, 0) \in \Omega(2)$, but $D \notin F(2)$ as illustrated in Fig. 5. However, as mentioned earlier, many $\Omega(n)$ permutations of interest (e.g., cyclic shift and p-ordering) are also in $I\Omega(n)$, hence in $F(n)$. By providing our self-routing network with some additional simple logic, the network can handle all $\Omega(n)$ permutations as well. Since the last n stages of $B(n)$ correspond to an Ω network, it is easy to see that an $\Omega(n)$ permutation can be realized on our network if the switches in stages 0 through $n - 2$ are all placed in state 0, while the remaining n stages obey the self-routing scheme described earlier (Fig. 3). One way to implement this on our network is to provide an additional "omega" bit with each "destination tag." (This bit will be = 1 iff we are performing an Ω permutation.) Each switch in stages 0 through $n - 2$ places itself in state 0 if it finds the "omega" bit = 1, otherwise the switch determines its state as before (i.e., the scheme of Fig. 3). The logic of switches in the last n stages of our self-routing network is not altered.

In the remainder of this section we further characterize the class of $F(n)$ permutations by proving several composite properties. Our next theorem shows that if we partition 2^n elements into blocks of size 2^r (of not necessarily consecutive elements) and permute the elements within each block according to some F permutations, then the composite mapping is also an F permutation.

Let J be any subset of $\{n-1, \cdots, 0\}$ and let $N = 2^n$. J can be used to partition the numbers $0, 1, \cdots, N-1$ into equivalence classes or blocks with the property that i and j are in the same class or block iff $(i)_k = (j)_k$ for all $k \in J$. For example, if $n = 3$ and $J = \{1\}$ then the set $\{0, 1, \cdots, 7\}$ is partitioned into the two blocks $\{0, 1, 4, 5\}$ and $\{2, 3, 6, 7\}$. This partitioning into blocks will be referred to as *J-partitioning*. Note that if $|J| = n - r$, then each block is of size 2^r.

Theorem 4: Let the $N = 2^n$ numbers $0, 1, \cdots, N-1$ be J-partitioned according to some J, $J \subseteq \{n-1, \cdots, 0\}$ and let $|J| = n - r$. Let the N-inputs and N-outputs of $B(n)$ be partitioned into blocks in this way. Let $G_i \in F(r)$ be a permutation mapping the 2^r inputs in the ith partition onto the 2^r outputs in the same partition. This requires reindexing the 2^r inputs and 2^r outputs in this block $0, 1, \cdots, 2^r - 1$. The N-input permutation G, defined by $G_0, G_1, \cdots, G_p$, where $p = 2^{n-r} - 1$ is in $F(n)$.

Proof: We shall once again use induction on n. When $n = 1$, it is clear that $G \in F(1)$. So, assume $G \in F(n)$, $1 \leq n < m$. When $n = m$, consider any J with $|J| = m - r$. Let $G_i \in F(r)$, $0 \leq i \leq p$ and $p = 2^{m-r} - 1$. We consider two cases: 1) $0 \in J$ and 2) $0 \notin J$.

1) $0 \in J$: In this case if i and j are two inputs (outputs) in the same J-partition of $B(m)$, then $(i)_0 = (j)_0 = (D_i)_0 = (D_j)_0$. Consequently, all switches in stage 0 of $B(m)$ are in state 0 and all input blocks corresponding to even (odd) input indices get switched to the upper (lower) $B(m-1)$ network. Input $2i$ of $B(m)$ goes to input i of $B_u(m-1)$, while input $2i + 1$ of $B(m)$ goes to input i of $B_l(m-1)$, $0 \leq i \leq 2^{m-1} - 1$. Define $J' = \{j \mid j + 1 \in (J - \{0\})\}$. Let the inputs and outputs of each of $B(m-1)$ networks be J'-partitioned. Inputs in the same J-partition of $B(m)$ are in the same J' partition of $B(m-1)$. $U(L)$ is made up of the G_is corresponding to blocks in the J-partition with even (odd) indices. Since each $G_i \in F(r)$, it follows from the induction hypothesis that $U \in F(m-1)$ and $L \in F(m-1)$. From Theorem 1 it follows that $G \in F(m)$.

2) $0 \notin J$: Now both inputs to any switch in stage 0 of $B(m)$ are from the same 2^r block under J-partitioning. Consequently, an input from block i gets routed to the upper (lower) $B(m-1)$ network iff the corresponding input of a $B(r)$ network will be routed to the upper (lower) $B(r-1)$ network when G_i is performed on $B(r)$. Let $J' = \{j \mid j + 1 \in J\}$. If two inputs are in the same J'-partition of the upper (lower) $B(m-1)$ network, then they must also be in the same J-partition of $B(m)$. Let U and L be as in Theorem 1. Let U^i, L^i correspond to U and L of Theorem 1 when the permutation defined by G_i is performed on $B(r)$, $0 \leq i \leq p$. It follows from the previous discussion that U is made up of subpermutations corresponding to U^i, $0 \leq i \leq p$, and L is made up of subpermutations corresponding to L^i, $0 \leq i \leq p$. Each U^i and L^i defines a permutation mapping inputs in one J'-partition. Also, $U^i \in F(r-1)$ and $L^i \in F(r-1)$, $0 \leq i \leq p$. From the induction hypothesis it follows that $U \in F(m-1)$ and $L \in F(m-1)$. Now from Theorem 1, it follows that $G \in F(m)$. □

If we interpret the N inputs to $B(n)$ as representing the N elements in the $N^{1/2} \times N^{1/2}$ array A (in row-major order), then Theorem 4 shows that the following array mappings used by Cannon [1] and Dekel, Nassimi, and Sahni [2] are in F:

(3) $A(i, j) \rightarrow A(i, (i + j) \bmod N^{1/2})$

(4) $A(i, j) \rightarrow A((i + j) \bmod N^{1/2}, j)$

(5) $A(i, j) \rightarrow A(i, i \oplus j)$

(6) $A(i, j) \rightarrow A(i \oplus j, j)$

(7) $A(i, j) \rightarrow A(i^R, j)$ (i^R is obtained by reversing the binary representation of i).

Theorem 4 may be generalized as in the next two theorems.

Theorem 5: Let $J \subseteq \{n-1, \cdots, 0\}$ and $|J| = n - r$. Let G be a permutation that maps inputs in the ith block of the J-partition to outputs in block B_i of the J-partition using the permutation G_i, $0 \le i < 2^{n-r}$. If $G_i \in F(r)$, $0 \le i < 2^{n-r}$, and $B = (B_0, B_1, \cdots B_p) \in F(n - r)$, $p = 2^{n-r} - 1$, then $G \in F(n)$.

Proof: Similar to that of Theorem 4. □

Theorem 5 shows that permutations in which rows of a matrix get permuted and mapped onto different rows using G_is and B satisfying the theorem are in $F(n)$. Theorem 4 may also be generalized to include permutations that can be defined for k-dimensional arrays.

Definition: Let $J_1, J_2, \cdots, J_k$ be disjoint subsets of $\{n-1, \cdots, 0\}$. The $J_1|J_2|\cdots|J_k$-*partition* of $N = 2^n$ elements is obtained by first obtaining the J_1-partition. Each block in the J_1-partition is then partitioned using J_2. Each block is next partitioned using J_3 and so on. This partitioning is best described by a tree in which blocks at level i are partitioned using J_i. We assume that the root is at level 1 and represents a block containing all 2^n elements.

Theorem 6: Let $J_1, \cdots, J_k$ be disjoint subsets of $\{n-1, \cdots, 0\}$ such that $\cup_{i=1}^{k} J_i = \{n-1, \cdots, 0\}$. Consider the resulting $J_1|J_2|\cdots|J_k$-partitioning of $\{0, 1, \cdots, N-1\}$. Define a permutation G of N elements as follows:

for $i := k$ **down to** 1 **do**

Let the children of any block at level i be $B_0, B_1, \cdots, B_{p-1}$, where $p = 2^r$ and $|J_i| = r$. The elements of B_j are mapped onto elements of $B_{\phi(j)}$ without changing their relative order, $0 \le j < p$. $\phi = (\phi(0), \phi(1), \cdots, \phi(p-1))$ is any permutation in $F(r)$.

end

Then, $G \in F(n)$.

Proof: Follows from Theorem 5 by a simple induction on k. □

As an example of a permutation satisfying the requirements of Theorem 6, consider the three-dimensional array $A(0{:}2^r - 1, 0{:}2^s - 1, 0{:}2^t - 1)$. Let the elements be indexed in row-major order and let $n = r + s + t$. Let $J_1 = \{i \mid n - r - s \le i \le n - r - 1\}$, $J_2 = \{i \mid 0 \le i \le n - r - s - 1\}$, and $J_3 = \{i \mid n - r \le i \le n - 1\}$. Consider the $J_1|J_2|J_3$-partition of the 2^n elements in A. From Theorem 6 it follows that the mapping $A(i, j, k) \rightarrow A(i', j', k')$, where

$$i' = (i + j * k) \bmod 2^r$$

$$j' = (p * j) \bmod 2^s, \; p \text{ is an odd constant}$$

and

$$k' = j \oplus k$$

is in $F(n)$.

We conclude this section with the remark that F is not closed under product. To see this, consider the permutations $A = (3, 0, 1, 2)$ and $B = (0, 1, 3, 2)$. $A \circ B = (2, 0, 1, 3)$. $A \in F(2)$, $B \in F(2)$, and $A \circ B \notin F(2)$.

III. Permutation Algorithms for a CCC, PSC, and MCC

In this section we are concerned with permuting data on an SIMD computer with limited interconnection. In particular, we consider an N-PE CCC, PSC, and MCC. Let $\langle R(i), D(i)\rangle$ be in PE(i), $0 \le i \le N-1$, where $R(i)$ is the data and $D(i)$ is its destination address. $D = (D(0), D(1), \cdots, D(N-1))$ is a permutation of $(0, 1, \cdots, N-1)$. The data $R(i)$ are to be routed from PE(i) to PE($D(i)$), $0 \le i \le N-1$. Assume $N = 2^n$.

Thompson [9] has observed that a permutation can be performed on a CCC, PSC, or MCC by "simulating" a Benes network on these machines. Assuming that the switch settings of the Benes network are known for the desired permutation, then the permutation can be performed in $0(\log N)$ routing steps on a CCC or PSC, and in $0(N^{1/2})$ routing steps on an MCC [9]. However, given the destination-address representation of an arbitrary permutation $D = (D(0), \cdots, D(N-1))$, extensive "preprocessing" is needed to determine the switch settings of the Benes network before the "simulation" can be carried out. For example, consider an N-PE CCC or PSC (with destination address $D(i)$ in PE(i), $0 \le i \le N-1$). The best known "preprocessing" algorithm (i.e., the algorithm to determine the switch settings of a Benes network for permutation D) requires $0(\log^4 N)$ time on an N-PE CCC or PSC [7]. So, including the preprocessing time, the total time needed to perform the permutation becomes $0(\log^4 N)$. For an MCC, using the $0(N^{1/2})$ preprocessing algorithm of [7], the total permutation time is $0(N^{1/2})$, but with a very large constant of proportionality.

Another method for performing a permutation D is to sort the records $\langle R(i), D(i)\rangle$ using $D(i)$ as the sort key. Batcher's bitonic sort algorithm [11] yields a permutation algorithm with time complexity $0(\log^2 N)$ for a CCC or PSC [15] and $0(N^{1/2})$ for an MCC [14], [16]. These are asymptotically best known algorithms for performing an *arbitrary* permutation on these machines.

For the special class of $F(n)$ permutations, we can "simulate" our self-routing Benes network without requiring any preprocessing. As a result, any $F(n)$ permutation can be realized in $0(\log N)$ steps on a CCC or PSC. For an MCC the resulting permutation algorithm requires $0(N^{1/2})$ steps (with a smaller constant of proportionality than the MCC algorithms mentioned in the previous two paragraphs). The "simulation" of the self-routing network is very straightforward. The remainder of this section gives the details of this simulation.

In specifying the algorithms we shall use "$\leftarrow$" to denote an assignment requiring data movement between two adjacent (i.e., directly connected) PE's. We shall use $i^{(b)}$ to denote the number whose binary representation differs from that of i only in bit b. PE selectivity is specified by providing an enable mask. Thus, the instruction

$$R(i^{(b)}) \leftarrow R(i), ((i)_b = 0)$$

requires records to be transferred from PE's with bit $b = 0$ to PE's with bit $b = 1$.

The permutation algorithm for a CCC is simply the following loop:

for $b := 0, 1, \cdots, n-2, n-1, n-2, \cdots, 0$ **do**

$$\langle R(i^{(b)}), D(i^{(b)})\rangle \leftrightarrow \langle R(i), D(i)\rangle, ((i)_b = 0, \text{ **and** } (D(i))_b = 1)$$

end.

The algorithm is illustrated in Fig. 6 for bit-reversal permutation. Note that the permutation is the same as that performed in Fig. 4. The column $D(i)^k$ in the figure shows the destination address in each PE after the kth iteration of the loop. An exchange between a pair of PE's is controlled by the underlined bit in the destination address. For example, during the first iteration (i.e., $b = 0$) an exchange is made between PE(6) and PE(7) because $(D(6))_0 = 1$. During the iteration $b = 2$, no exchange is made between PE(0) and PE(4) since $(D(0))_2 = 0$; an exchange is made between PE(1) and PE(5) since $(D(1))_2 = 1$, and so on.

As for the number of routing steps in the CCC algorithm, if both R and D fit into one word and the interchange ($\leftrightarrow$) can be performed with one unit-route, then the number of unit-routes needed is $2n - 1 = 2 \log N - 1$. If the interchange needs two unit-routes, then $4 \log N - 2$ unit-routes are needed to perform any permutation in $F(n)$. Ω permutations can be performed by skipping the first $n - 1$ iterations of the above loop. For $I\Omega(n)$ we may skip the last $n - 1$ iterations. For a BPC permutation given by the vector $A = (A_{n-1}, \cdots, A_0)$, if $A_j = j$ then the iteration(s) $b = j$ may be skipped. This is because $(D(i))_j = (i)_j$, hence, no routing across "dimension" j of the cube is needed. For a BPC permutation the number of routing steps used by the algorithm is within a factor of two from the optimal. See Nassimi and Sahni [12] for an optimal algorithm to perform BPC permutations on a CCC.

We next give the algorithm for realizing $F(n)$ permutations on a PSC. Let EXCHANGE $\langle R(i), D(i)\rangle$, SHUFFLE $\langle R(i), D(i)\rangle$, and UNSHUFFLE $\langle R(i), D(i)\rangle$ route $\langle R(i), D(i)\rangle$ along the corresponding connection. Note that EXCHANGE interchanges $\langle R(i), D(i)\rangle$ and $\langle R(i^{(0)}), D(i^{(0)})\rangle$. The above loop is simulated by the following code:

for $b := 0$ **to** $n - 2$ **do**
  EXCHANGE $\langle R(i), D(i)\rangle$, $(i_0 = 0$ **and** $(D(i))_b = 1)$
  UNSHUFFLE $\langle R(i), D(i)\rangle$
end
EXCHANGE $\langle R(i), D(i)\rangle$, $(i_0 = 0$ **and** $(D(i))_{n-1} = 1)$
for $b := n - 2$ **down to** 0 **do**
  SHUFFLE $\langle R(i), D(i)\rangle$
  EXCHANGE $\langle R(i), D(i)\rangle$, $(i_0 = 0$ **and** $(D(i))_b = 1)$
end.

The number of unit-routes needed is $4 \log N - 3$. To perform an Ω permutation, the first **for** loop should be replaced by a shuffle on $\langle R(i), D(i)\rangle$.

The permutation algorithm for an $N^{1/2} \times N^{1/2}$ MCC is obtained directly from the CCC algorithm. A pair of PE's that differ in bit b of their row-major indices have a horizontal distance of 2^b if $b < \log N^{1/2}$ and have a vertical distance of 2^{b-j} if $b \geq \log N^{1/2}$ ($j = \log N^{1/2}$). An interchange between elements 2^k apart requires 2^{k+1} unit-routes (2^k in each direction). So, all permutations in $F(n)$ can be performed with $7N^{1/2} - 8$ unit-routes. For permutations in BPC(n) the resulting algorithm is optimal to within a factor of four. Nassimi and Sahni [6] present an optimal algorithm to perform BPC permutations on an MCC.

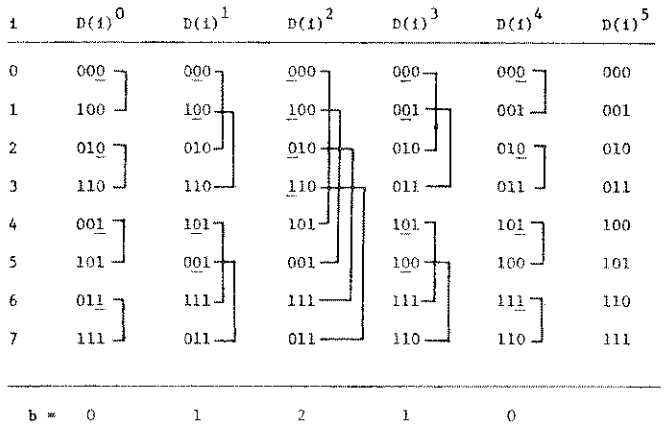

Fig. 6. Bit-reversal permutation on a CCC with $N = 8$ PE's.

Finally, it should be noted that the destination tags ($D(0)$, $\cdots$, $D(N - 1)$) may be efficiently computed from a more compact representation (if there is any). For example, a BPC(n) permutation may be represented by a vector $A = (A_{n-1}, \cdots, A_0)$. Given the A vector ($\log N$ words) in the instruction stream, and assuming that an appropriate set of simple instructions is available, it is easy to see that each PE can compute its own destination tag in $0(\log N)$ steps. (This holds independent of the interconnection scheme as the computation does not involve any PE-to-PE communication.) Hence, the total number of steps needed to perform a BPC permutation from its A-vector representation is still $0(\log N)$ on a CCC or PSC, and $0(N^{1/2})$ on an MCC. As another example, consider "p-ordering and cyclic shift" permutation

$$D(i) = (p \cdot i + k) \bmod N, \qquad 0 \leq i \leq N - 1.$$

It should be obvious that, given the constants p and k in the instruction stream, the destination tags can be computed in $0(1)$ steps.

IV. Conclusions

We have presented a self-routing Benes network. A destination tag ($\log N$ bits) is passed through the network along with each input. A very simple logic is required in each switch. This logic determines the binary state of the switch from a particular bit of the destination tag of its upper input. The resulting network is capable of realizing a rich class of permutations, $F(n)$, $N = 2^n$, in $0(\log N)$ time. We showed that the class $F(n)$ includes Lawrie's $I\Omega(n)$ permutations, Nassimi and Sahni's BPC(n) permutations, and consequently Lenfant's FUB families. Any $\Omega(n)$ permutation can also be realized on this network if the switches in the first $n - 1$ stages are "forced" into state 0. We further demonstrated the richness of $F(n)$ class by proving several composite properties for $F(n)$ permutations. Finally, in Section III we showed that by "simulating" the self-routing network, any $F(n)$ permutation can be performed in $0(\log N)$ steps on an N-PE CCC or PSC and in $0(N^{1/2})$ steps on an $N^{1/2} \times N^{1/2}$ MCC.

We believe that the self-routing Benes network promises an

effective interconnection network for SIMD computers. We propose an SIMD computer with *two* interconnection networks as follows.

1) A network $E(n)$ providing direct connections between PE's, hence capable of performing some permutations in 0(1) time. This may be, for example, the connection scheme of a PSC or MCC.

2) The self-routing Benes network $B(n)$ with 0(lot N) delay. PE(i) is connected to input i and output i of $B(n)$, $0 \leq i \leq N-1$.

Then some permutations are performed more efficiently through $E(n)$, while some others via $B(n)$.

For the case when the first network $E(n)$ is the connection scheme of a PSC, the utility of the second network $B(n)$ deserves a comment. Realizing an $F(n)$ permutation through $B(n)$ requires 0(log N) *gate delays*. From the results of Section III we know that the same permutation can be performed via $E(n)$ in 0(log N) *routing steps*. However, each routing step involves broadcasting an instruction to all PE's, and gating data from register of one PE to that of another PE. Therefore, much less time is required to perform the permutation through $B(n)$.

Finally, we would like to point out that by providing registers between the stages of $B(n)$, the network may operate in *pipelined* mode. That is, a new N-element vector may enter the network every clock-period. Hence, when permuting a sequence of vectors (not necessarily according to the same permutation), the network will output the first permuted vector after 0(log N) delay, while each subsequent permuted vector will emerge after unit delay. Of course, the network may still operate in nonpipelined mode with appropriate logic to bypass the registers. This enables us to rapidly permute a *single* vector as well.

REFERENCES

[1] L. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Montana State Univ., Bozeman, 1969.

[2] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," in *Proc. 17th Annu. Allerton Conf. Commun., Contr., and Comput.*, Oct. 1979, pp. 27–36.

[3] T. Lang and H. Stone, "A shuffle-exchange network with simplified control," *IEEE Trans. Comput.*, vol. C-25, pp. 55–65, Jan. 1976.

[4] D. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145–1155, Dec. 1975.

[5] J. Lenfant, "Parallel permutations of data: A Benes network control algorithm for frequently used permutations," *IEEE Trans. Comput.*, vol. C-27, pp. 637–647, July 1978.

[6] D. Nassimi and S. Sahni, "An optimal routing algorithm for mesh-connected parallel computers," *J. Ass. Comput. Mach.*, vol. 27, pp. 6–29, Jan. 1980.

[7] ———, "Parallel algorithms to set-up the Benes permutation network," Univ. of Minnesota, Minneapolis, Tech. Rep. 79-19, June 1979; also to appear in *Proc. Workshop on Interconnection Networks for Parallel and Distributed Processing*, Purdue University, Lafayette, IN, Apr. 1980.

[8] S. Orcutt, "Implementation of permutation functions on Illiac-IV type computers," *IEEE Trans. Comput.*, vol. C-25, pp. 929–936, Sept. 1976.

[9] C. Thompson, "Generalized connection networks for parallel processor intercommunication," *IEEE Trans. Comput.*, vol. C-27, pp. 1119–1125, Dec. 1978.

[10] A. Waksman, "A permutation network," *J. Ass. Comput. Mach.*, vol. 15, no. 1, pp. 159–163, 1968.

[11] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS 1968 SJCC*, vol. 32, Montvale, NJ: AFIPS Press, pp. 307–314.

[12] D. Nassimi and S. Sahni, "A self-routing Benes network and parallel permutation algorithms," Univ. of Minnesota, Minneapolis, Tech. Rep. 79-13, May 1979.

[13] ———, "Parallel permutation and sorting algorithms and a new generalized connection network," *J. Ass. Comput. Mach.*, to be published.

[14] ———, "Bitonic sort on a mesh-connected parallel computer," *IEEE Trans. Comput.*, vol. C-28, pp. 2–6, Jan. 1979.

[15] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

[16] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. Ass. Comput. Mach.*, vol. 20, no. 4, pp. 263–271, Apr. 1977.

David Nassimi, for a photograph and biography, see p. 107 of the February 1981 issue of this TRANSACTIONS.

Sartaj Sahni (M'79), for a photograph and biography, see p. 107 of the February 1981 issue of this TRANSACTIONS.