Short Notes

A Fast Parallel Algorithm for Routing Unicast Assignments in Beneš Networks

Ching-Yi Lee and A. Yavuz Oruç

***Abstract*—This paper presents a new parallel algorithm for routing unicast (one-to-one) assignments in Beneš networks. Parallel routing algorithms for such networks were reported earlier, but these algorithms were designed primarily to route permutation assignments. The routing algorithm presented in this paper removes this restriction without an increase in the order of routing cost or routing time. We realize this new routing algorithm on two different topologies. The algorithm routes a unicast assignment involving $O(k)$ pairs of inputs and outputs in $O(\lg^2 k + \lg n)$ time on a completely connected network of n processors and in $O(\lg^4 k + \lg^2 k \lg n)$ time on an extended shuffle-exchange network of n processors. Using $O(n \lg n)$ processors, the same algorithm can be pipelined to route α unicast assignments, each involving $O(k)$ pairs of inputs and outputs, in $O(\lg^2 k + \lg n + (\alpha - 1) \lg k)$ time on a completely connected network and in $O(\lg^4 k + \lg^2 k \lg n + (\alpha - 1)(\lg^3 k + \lg k \lg n))$ time on the extended shuffle-exchange network. These yield an average routing time of $O(\lg k)$ in the first case, and $O(\lg^3 k + \lg k \lg n)$ in the second case, for all $\alpha \geq \lg n$. These complexities indicate that the algorithm given in this paper is as fast as Nassimi and Sahni's algorithm for unicast assignments, and with pipelining, it is faster than the same algorithm at least by a factor of $O(\lg n)$ on both topologies. Furthermore, for sparse assignments, i.e., when $k = O(1)$, it is the first algorithm which has an average routing time of $O(\lg n)$ on a topology with $O(n)$ links.**

***Index Terms*—Connector, packet switching network, permutation network, unicast assignment.**

I. Introduction

The Beneš network has received much attention in interconnection network literature because of its $O(n \lg n)$[1] cost and $O(\lg n)$ depth [1], [9]–[12]. In a way, this network can be considered the forerunner of most multistage interconnection networks that have been extensively studied and used in some real parallel computer systems for interprocessor or processor-memory communications [7], [14]. Recently, there has been some renewed interest in multistage networks in particular in Beneš and Clos networks for ATM switching due to their expandibility and modularity [5]. To avoid packet loss without sacrificing switching speed, these networks need to be routed fast. Several routing algorithms have been reported for the Beneš network including the $O(n \lg n)$ time recursive looping procedure [13], [16], nonrecursive complete residue partition tree algorithm [9] and $O(\lg^2 n)$ time parallel algorithms of Lev *et al.* [10], and Nassimi and Sahni [12]. Other routing algorithms for the Beneš network include matching and edge-coloring schemes [2], [3], [6].

While some of these algorithms are fast, their time complexities are still higher than the $O(\lg n)$ depth of the Beneš network. Furthermore, these algorithms are primarily designed to route permutation assignments. In case of incomplete assignments where some inputs are idle, they cannot be used unless the idle inputs are given dummy outputs. This, however, takes additional time and may render these algorithms inefficient, especially in case of sparse assignments–assignments involving $O(k)$ pairs of inputs and outputs, where $k \ll n$.

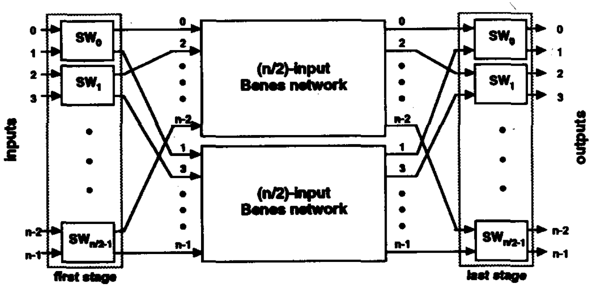


Fig. 1. The recursive construction of an n-input Beneš network.

In this paper, we present an efficient parallel algorithm for routing incomplete assignments on the Beneš network. This routing algorithm can be run on any parallel computer whose processors are equipped with a constant number of $O(\lg n)$-bit registers and some simple arithmetic and logic circuitry that can compare $O(\lg n)$ bit numbers and perform some counting and decoding tasks on them. We realize our routing scheme on two different topologies. If every pair of processors are interconnected by a direct arc then routing an assignment involving $O(k)$ pairs of inputs and outputs takes $O(\lg^2 k + \lg n)$ time without pipelining and $O(\lg k)$ time with pipelining. We also establish that using a weaker topology, namely the extended shuffle-exchange network (which will be defined in Section IV), leads to a routing algorithm with $O(\lg^4 k + \lg^2 k \lg n)$ time without pipelining and $O(\lg^3 k + \lg k \lg n)$ time with pipelining.

II. Basic Facts and Definitions

An n-network is a directed acyclic graph with n distinguished source vertices, called inputs, n distinguished sink vertices, called outputs, and some internal vertices, called switching nodes. A k-assignment for an n-network is a pairing of k of its inputs with k of its outputs such that each output appears in at most one pair. An assignment is called one-to-one or *unicast* if each input appears in at most one pair. A *permutation* assignment for an n-network is a unicast n-assignment. An n-network is said to *realize* an assignment if, for each pair (a, b) in the assignment, a path can be formed from input a to output b by setting the switching nodes in the network with the constraint that the paths for no two pairs (a, b) and (c, d) overlap unless $a = c$. An n-network that can realize all unicast assignments is called a *unicast* n-network.

The well-known Beneš network is a unicast n-network that is constructed recursively as shown in Fig. 1. Each 2×2 switch can be set in two ways: either *through* state where the two inputs are connected straight-through to the two outputs, or *cross* state where the two inputs are connected to opposite outputs.

Manuscript received February 1, 1993; revised April 29, 1994. This work is supported in part by the National Science Foundation under Grant No. CCR-8708864 and in part by the Minta Martin Fund of the School of Engineering at the University of Maryland, College Park, MD.

The authors are with the Electrical Engineering Department and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742-3025 USA.

IEEE Log Number 9408135.

[1] All logarithms are in base 2 unless otherwise stated, and lg n denotes the logarithm of n in base 2.

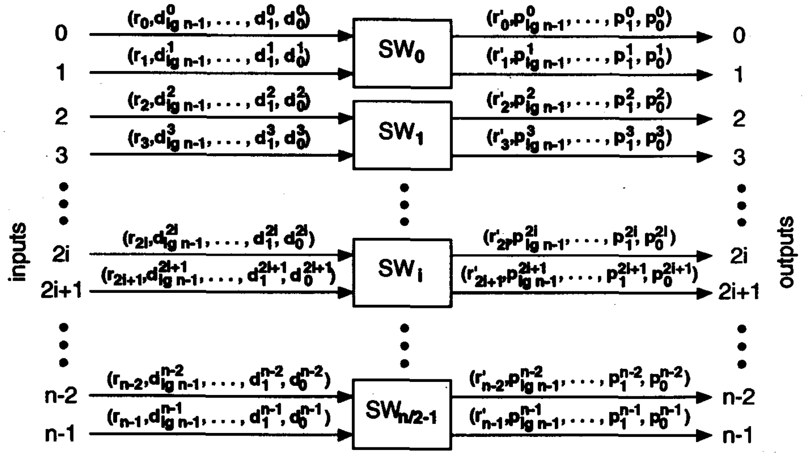

inputs
outputs
0
1
2
3
2i
2i+1
n-2
n-1
SW0
SW1
SWi
SWn/2-1
(r0,d0lg n-1, . . . , d01, d00)
(r1,d1lg n-1, . . . , d11, d10)
(r2,d2lg n-1, . . . , d21, d20)
(r3,d3lg n-1, . . . , d31, d30)
(r2i,d2ilg n-1, . . . , d2i1, d2i0)
(r2i+1,d2i+1lg n-1, . . . , d2i+11, d2i+10)
(rn-2,dn-2lg n-1, . . . , dn-21, dn-20)
(rn-1,dn-1lg n-1, . . . , dn-11, dn-10)
(r'0,p0lg n-1, . . . , p01, p00)
(r'1,p1lg n-1, . . . , p11, p10)
(r'2,p2lg n-1, . . . , p21, p20)
(r'3,p3lg n-1, . . . , p31, p30)
(r'2i,p2ilg n-1, . . . , p2i1, p2i0)
(r'2i+1,p2i+1lg n-1, . . . , p2i+11, p2i+10)
(r'n-2,pn-2lg n-1, . . . , pn-21, pn-20)
(r'n-1,pn-1lg n-1, . . . , pn-11, pn-10)

Fig. 2. $H(n)$: a one-stage n-network consisting of $n/2$ 2×2 switches.

Paths in an n-network will be established by specifying some routing bits at its inputs. It is assumed that the routing bits for each input is accompanied by some binary coded message that is to be routed from that input to the output specified in the routing bits. A message and routing bits combined together will be termed a *packet*. For an n-input Beneš network, the routing part of a packet, to be called the *header*, is assumed to have lg n + 1 bits, and will be denoted as $(r_i, d^i_{\lg\ n-1}, \cdots, d^i_1, d^i_0)$ for a packet at input i. The bit r_i specifies whether input i is paired with some output. Input i is said to be *busy* if $r_i = 1$, and it is said to be *idle* if $r_i = 0$. Assuming that $r_i = 1$, the remaining bits, i.e., $(d^i_{\lg\ n-1}, \cdots, d^i_1, d^i_0)$, form the output address which specifies the binary representation of the output paired with input i with $d^i_{\lg\ n-1}$ being the most significant bit. Besides, a switch is said to be *busy* if both of its inputs are busy; it is said to be *semi-busy* if one of its inputs is busy and the other input is idle, and it is said to be *idle* if both of its inputs are idle.

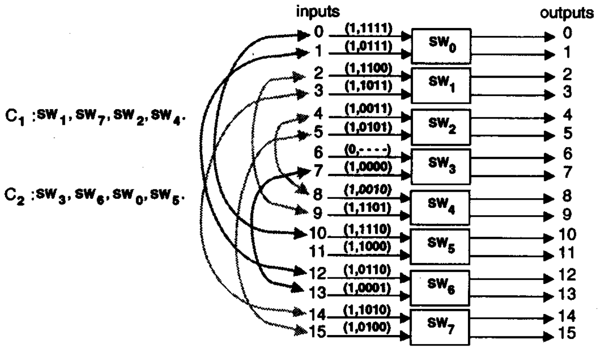


Fig. 3. Two chains with respect to a unicast 15-assignment for $H(n)$ where $n = 16$.

III. The Routing Principle

Unicast assignments for an n-input Beneš network can be recursively decomposed into half-sized unicast assignments stage by stage from left to right.

Notation: For $0 \leq i \leq n-1$, if $(b^i_{\lg\ n-1}, \cdots, b^i_1, b^i_0)$ is the binary representation of i, then $\bar{i}$ denotes the integer which has the binary representation $(b^i_{\lg\ n-1}, \cdots, b^i_1, \bar{b}^i_0)$, and[2] i and $\bar{i}$ are called a *dual pair* of integers. ||

Now let $H(n)$ denote a one-stage n-network constituting the first stage of an n-input Beneš network and comprising $n/2$ 2×2 switches, $SW_0, SW_1, \cdots, SW_{n/2-1}$, as shown in Fig. 2.

Definition 1: Given a unicast assignment $\{(i, (r_i, d^i_{\lg\ n-1}, \cdots, d^i_1, d^i_0)) \colon 0 \leq i \leq n-1\}$ for $H(n)$, a sequence of switches $SW_{i_0}, SW_{i_1}, \cdots, SW_{i_{p-1}}$ in $H(n)$ is said to form a *chain* with respect to that assignment if, for all $0 \leq q \leq p-2$, one of the inputs of SW_{i_q} and one of the inputs of $SW_{i_{q+1}}$ are paired with a dual pair of outputs and have their r_i's set to 1. Furthermore, $SW_{i_0}, SW_{i_1}, \cdots, SW_{i_{p-1}}$ is said to be a *closed chain* if the other input of $SW_{i_{p-1}}$ and the other input of SW_{i_0} are also paired with a dual pair of outputs. It is said to be an *open chain* otherwise. ||

The *size* of a chain is the number of switches in the chain. Given a unicast k-assignment for $H(n)$, the size of a chain can be as large as $\min\{\lfloor (k+2)/2 \rfloor, n/2\}$ and[3] as small as 1. The number of chains can be as large as $\min\{k, n/2\}$ (when each chain has size 1) and as small as 1 (when that chain is of size $\lfloor (k+2)/2 \rfloor$ or $\lfloor k/2 \rfloor$). Fig. 3 shows a closed chain C_1 and an open chain C_2 with respect to a unicast 15-assignment for $H(n)$ where $n = 16$.

An open chain, say $SW_{i_0}, SW_{i_1}, \cdots, SW_{i_{p-1}}$, has two *end* switches (i.e., SW_{i_0} and $SW_{i_{p-1}}$). Based on the status of the end switches, two types of open chains are distinguished.

Definition 2: An open chain is said to be a *full open chain* if both of its end switches are busy or both are semi-busy, and it is said to be a *half open chain* if one of its end switches is busy and the other end switch is semi-busy. ||

For example, C_2 given in Fig. 3 is a half open chain since SW_5 is busy and SW_3 is semi-busy.

Theorem 1: Given $\{(i, (r_i, d^i_{\lg\ n-1}, \cdots, d^i_1, d^i_0)) \colon 0 \leq i \leq n-1\}$, a unicast k-assignment for $H(n)$, let $(r'_i, p^i_{\lg\ n-1}, \cdots, p^i_1, p^i_0)$ denote the header of the packet at output i of $H(n)$, $0 \leq i \leq n-1$, as shown in Fig. 2. There exist settings for $SW_0, SW_1, \cdots, SW_{n/2-1}$ such that $\{(2i, (r'_{2i}, p^{2i}_{\lg\ n-1}, \cdots, p^{2i}_2, p^{2i}_1)) \colon 0 \leq i \leq n/2-1\}$ is a unicast k_0-assignment and $\{(2i+1, (r'_{2i+1}, p^{2i+1}_{\lg\ n-1}, \cdots, p^{2i+1}_2, p^{2i+1}_1)) \colon 0 \leq i \leq n/2-1\}$ is a unicast k_1-assignment, where $k_0 = \lceil k/2 \rceil$ and $k_1 = \lfloor k/2 \rfloor$.

[2] $\bar{b}$ denotes the binary complement of bit b.

[3] $\lfloor x \rfloor$ denotes the largest integer equal to or smaller than x, and $\lceil x \rceil$ denotes the smallest integer equal to or larger than x.

Proof: With respect to the given unicast k-assignment and without loss of generality, suppose that there are c chains, $1 \leq c \leq \min\{k, n/2\}$, and there are f half open chains among these c chains, $0 \leq f \leq c$. From Definition 1, the switches of each chain can be set such that, given inputs i and j which have $r_i = r_j = 1$ and $(d^i_{\lg n-1}, \cdots, d^i_2, d^i_1, d^i_0) = (d^j_{\lg n-1}, \cdots, d^j_2, d^j_1, \bar{d}^j_0)$, one is routed to an even-numbered output and the other is routed to an odd-numbered output, and once a switch of the chain is set then the settings of the other switches of the chain are fixed. That is, each chain can be set in exactly two ways such that the above condition can be satisfied. Besides, different chains can be set mutually independently, and there are 2^c ways to set the given c chains since each chain has two ways of settings. It is obvious that, if any one of these 2^c ways is used to set the switches in $H(n)$, then $\{(2i, (r'_{2i}, p^{2i}_{\lg n-1}, \cdots, p^{2i}_2, p^{2i}_1)) \colon 0 \leq i \leq n/2-1\}$ is a unicast k_0-assignment and $\{(2i+1, (r'_{2i+1}, p^{2i+1}_{\lg n-1}, \cdots, p^{2i+1}_2, p^{2i+1}_1)) \colon 0 \leq i \leq n/2-1\}$ is a unicast k_1-assignment for some integers k_0 and k_1 with $k_0 + k_1 = k$. Thus, it suffices to show at least one of the 2^c ways of settings will result in $k_0 = \lceil k/2 \rceil$ and $k_1 = \lfloor k/2 \rfloor$. For the closed and full open chains, no matter how they are set, k_0 and k_1 will increase by the same integer since such chains have even numbers of busy inputs. However, for a half open chain (it has an odd number of busy inputs), one of its two settings, named *type-0* setting, will have k_0 increase one more than k_1, and the other setting, named *type-1* setting, will have k_0 increase one less than k_1. Suppose the chanins are set in such a way that $\lceil f/2 \rceil$ of the f half open chains are in their *type-0* settings and the other $\lfloor f/2 \rfloor$ half open chains are in their *type-1* settings. Then, $k_0 = \lceil k/2 \rceil$ and $k_1 = \lfloor k/2 \rfloor$, and the statement follows. ||

Any algorithm satisfying the statement of Theorem 1 can be used recursively to set the switches in the first $\lg n - 1$ stages. Thereafter the packets can be routed on a self-routing basis through the last $\lg n$ stages to their final destinations by decoding the output addresses bit by bit as identified before [9].

IV. The Parallel Routing Algorithm

Following the discussion in the previous section, it is only necessary to describe a parallel algorithm for $H(n)$ so that the routing principle stated in Theorem 1 is satisfied. The parallel routing algorithm for the Beneš network then follows.

It is assumed that there are n interconnected processors, $PR(0), PR(1), \cdots, PR(n-1)$, and that a packet header $(r_i, d^i_{\lg n-1}, \cdots, d^i_1, d^i_0)$ is initially input to $PR(i)$. $PR(i)$ and $PR(\bar{i})$ are called a *dual pair* of processors and will determine the setting of $SW_{\lfloor i/2 \rfloor}$ of $H(n)$, $0 \leq i \leq n-1$. The time complexity of the parallel algorithm depends on the interconnection topology between these n processors. The algorithm will run on two connection topologies, namely the *completely connected network* and the *extended shuffle-exchange* network.

A. The Processor Topologies

The parallel algorithm uses two macro functions: *move process* and *pairing process* for exchanging certain routing bits among k-subsets of n processors. A move process permutes routing bits among a subset of k processors. A pairing process marks a subset of k processors such that $\lfloor k/2 \rfloor$ of them are marked type 0 and $\lceil k/2 \rceil$ of them are marked type 1.

The time complexity to execute these two processes depends on the topology which interconnects the n processors. We will consider two topologies. The first is the completely connected network in which there is a link between every two of the n processors. It is obvious that the move process can be accomplished on this topology in $O(1)$ time for any $k, 1 \leq k \leq n$. To carry out the pairing process, consider any k of the n processors. We can always construct a ($2^{\lceil \lg k \rceil}$-1)-node binary tree containing these k processors using the links of the completely connected network interconnecting the n processors. It is not difficult to see that the pairing process can then be performed on the binary tree in $O(\lg k)$ time.

The second topology we will use is called the *extended shuffle-exchange* network in which (a) each dual pair of processors is connected by a link, and (b) processors $PR(0), PR(1), \cdots, PR(n/2^m - 1)$ are $(n/2^m)$-shuffle connected for $m = 0, 1, \cdots, \lg n - 2$, (i.e., $PR(0), PR(1), \cdots, PR(n-1)$ is n-shuffle connected, $PR(0), PR(1), \cdots, PR(n/2-1)$ is $n/2$-shuffle connected, and so on). Compared with the $O(n^2)$ links needed for the completely connected network, it is easily verified that the extended shuffle-exchange network requires only $O(n)$ links which is also the case with the ordinary shuffle-exchange network.

To execute a move process, the extended shuffle-exchange network takes three steps. Suppose that some routing bits are to be permuted among some k processors, $2 \leq k \leq n$ ($k = 1$ is a trivial case). In the first step, the k routing bits are moved to any k of the first $2^{\lceil \lg k \rceil}$ processors by passing $\lg n$ times through the n-shuffle connection. In the second step, these k routing bits are sorted according to their destination addresses by passing $\lg^2 k'$ times through the k'-shuffle interconnection between $PR(0), PR(1), \cdots, PR(k'-1)$, where $k' = 2^{\lceil \lg k \rceil}$ [8], [15]. In the third step, the sorted packets are moved to their final destinations by passing them $\lg n$ times through the n-shuffle connection. Therefore, a move process involving k routing bits can be executed over the extended shuffle-exchange network in $O(\lg^2 k + \lg n)$ time. We note that if we had used an ordinary shuffle-exchange network then the second step would take $O(\lg^2 n)$ time. The extended shuffle-exchange network avoids this by first concentrating the k routing bits into the first k processors and then using the smallest shuffle-exchange subnetwork that contains those k processors.

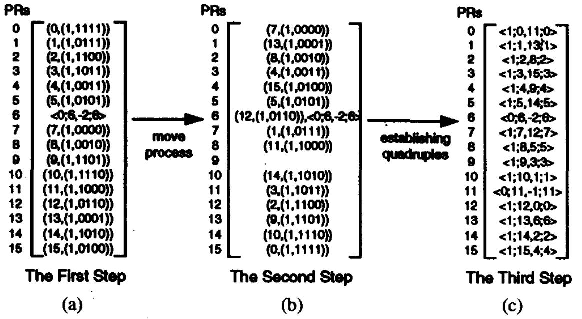
As for the pairing process, we first observe a binary tree of n processors can be emulated by passing $\lg n$ times over the ordinary n-processor shuffle-exchange network. Since every k processors we choose to mark constitute the leaves of this binary tree, the pairing of k processors can be performed in $O(\lg n)$ time as is already observed earlier.

B. The Parallel Routing Scheme

From the proof of Theorem 1 in Section III, each chain has two settings (*type-0* and *type-1* settings), and the inputs of its switches can be partitioned into two *equivalence classes* such that the inputs in one equivalence class are connected to even-numbered outputs and the inputs in the other equivalence class are connected to odd-numbered outputs. Once the equivalence classes of inputs in a chain are established, the settings of switches in the chain are straightforward. The parallel algorithm that follows will use such equivalence classes to speed up the switch settings. The following proposition shows how to determine the pairs of inputs that are in the same equivalence class, and is similar to the observation given on p. 150 in [12].

Proposition 1: Let $\{(i, (r_i, d^i_{\lg n-1}, \cdots, d^i_1, d^i_0)) \colon 0 \leq i \leq n-1\}$ be a unicast assignment for $H(n)$. If $(d^i_{\lg n-1}, \cdots, d^i_1, d^i_0) = (d^j_{\lg n-1}, \cdots, d^j_1, \bar{d}^j_0)$, $r_i = r_j = 1$ and $i \neq j$, then inputs i and $\bar{j}$ are in the same equivalence class and inputs j and $\bar{i}$ are in another equivalence class.

Remark 1: This equivalence among the inputs will be noted by associating each input with an ordered *quadruple* $\langle c_i, i, e_i, p_i \rangle$ where c_i is a single bit used to indicate if this input belongs to a closed or open chain ($c_i = 0$ means i belongs to an open chain and $c_i = 1$ means i belongs to a closed chain), i denotes the input itself, e_i points



The First Step (a) The Second Step (b) The Third Step (c)

Fig. 4. The three steps of the first phase by using the unicast 15-assignment given in Fig. 3 as an example.

to an input that is in the same equivalence class as this input, and p_i, called the *representative* of this input, will be used to route this input. If i and j satisfy the hypothesis of Proposition 1, two ordered quadruples $\langle 1; i, \bar{j}; p_i\rangle$ and $\langle 1; j, \bar{i}; p_j\rangle$ will be established, where their first elements are initialized to 1 and their fourth elements are initialized to $p_i = \min\{i, \bar{j}\}$ and $p_j = \min\{j, \bar{i}\}$. If $r_i = 1$ and there is no $r_j = 1$ for which $(d^i_{\lg n-1}, \cdots, d^i_1, d^i_0) = (d^j_{\lg n-1}, \cdots, d^j_1, \bar{d}^j_0)$ (i.e., input i is paired with an output whose dual output is idle), an ordered quadruple $\langle 1; i, -1; i\rangle$ will be established, where -1 is used to denote that input i belongs to a busy end switch of an open chain, or an open chain of size 1. If $r_i = 0$ and $r_{\bar{i}} = 1$, an ordered quadruple $\langle 0; i, -2; i\rangle$ will be established, where -2 is used to denote that input i belongs to a semi-busy end switch of an open chain. ||

The parallel algorithm can be roughly divided into four phases. In the first phase, ordered quadruples as defined in Remark 1 are established. In the second phase, the representative in each quadruple is computed such that each input knows to which chain it belongs and all the inputs in the same equivalence class will agree on a common representative. In the third phase, each half open chain is assigned a *type-0* or *type-1* setting so that the statement of Theorem 1 (i.e., $k_0 = \lceil k/2\rceil$ and $k_1 = \lfloor k/2\rfloor$) is satisfied. In the fourth phase, each switch is set by using the representatives of its two inputs.

1) First Phase: The first phase applies Remark 1 to establish ordered quadruples, and can be further decomposed into three steps. In the first step, packet headers are input to the processors, and the idle inputs which belong to semi-busy switches have their quadruples established. In the second step, packet headers are permuted among the processors so that each dual pair of processors can apply the steps in Remark 1 in parallel. In the third step, the remaining quadruples are established and moved to the processors specified by their second elements, and their first elements are changed to 0 if their third elements are -1. An illustration of these three steps is given in Fig. 4.

2) Second Phase: The second phase is an iterative procedure which computes the representative of each quadruple using a pointer jumping technique [4]. A chain, depending upon its type, is decomposed into two sequences of quadruples in the first phase where the quadruples in each sequence belong to the same equivalence class. Let us use the quadruples in each sequence as nodes to form a directed graph in which a directed arc is established from a quadruple to another quadruple if and only if the third element of the former quadruple is equal to the second element of the latter quadruple. Then, a k-size closed chain will form two k-node *closed subchains* for which each node has an incoming arc and an outgoing arc, and a k-size open chain will form two k-node *open subchains* for which each source node has an outgoing arc, each sink node has an incoming arc, and each of the other nodes has an incoming arc and an outgoing arc. For example, Fig. 5 explicitly depicts the four subchains obtained from the quadruples in (c) of Fig. 4.

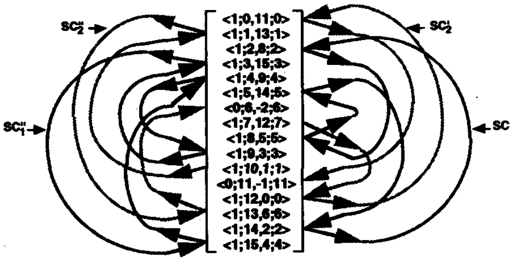
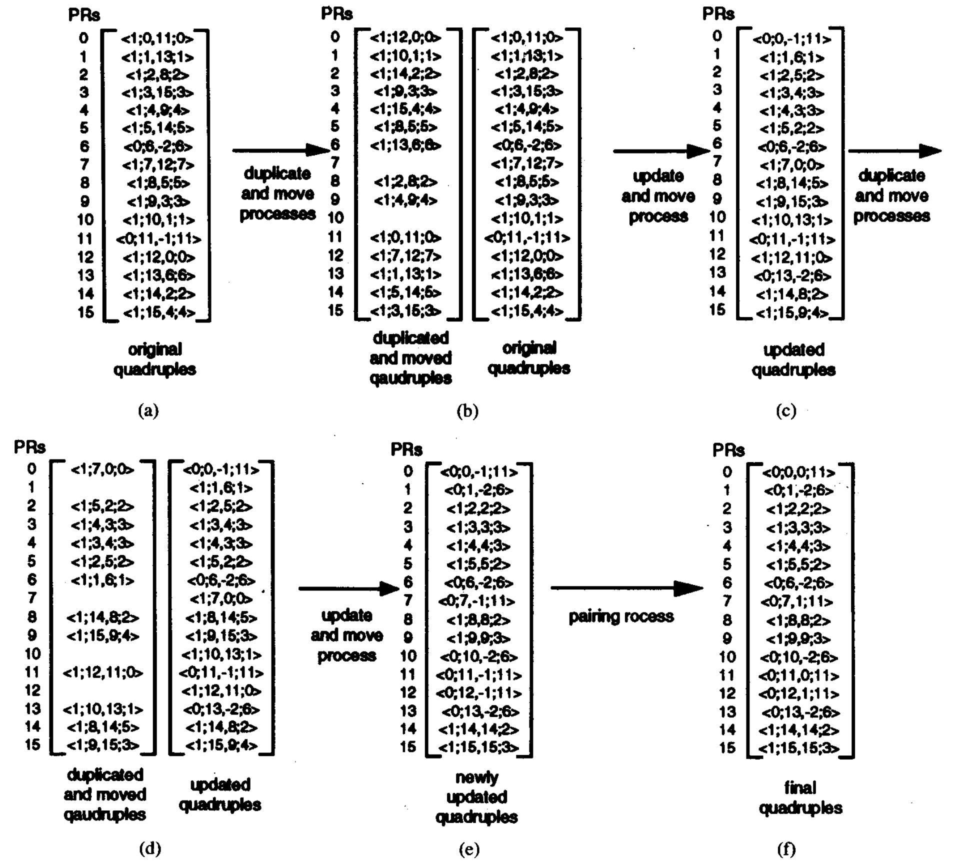


Fig. 5. The subchains formed from the quadruples in Fig. 4(c).

These subchains will be used to facilitate an understanding of the iterative procedure. If a quadruple is in a closed subchain in the mth iteration, its first element is 1, its third element points to the second element of its 2^mth successor and its last element points to the smallest input among the 2^m second elements of its first 2^m successors. If a quadruple is in an open subchain, the way its elements are updated depends on when it is known that this quadruple is in an open subchain. When this quadruple is at least 2^m far away from the sink quadruple in the mth iteration, its elements are updated in the same way as if this quadruple is in a closed subchain. When this quadruple is at distance less than $2^m - 1$ away from the sink quadruple in the mth iteration, it is recognized to be in an open subchain, and its first element is changed to 0, its third element is changed to -1 or -2 (the same as the third element of the sink quadruple) and its last element points to the second element of the sink quadruple. That is, for each open subchain, the updating information is exponentially propagated from the sink quadruple to the source quadruple. Using this updating procedure, after $\lg\lceil k\rceil$ iterations, any k quadruples that form a closed subchain will select the smallest input among their second elements as their common representative, and any k quadruples that form an open subchain will select the second element of the sink quadruple as their common representative. For example, the four subchains in Fig. 5 need two iterations to determine their representatives at the end of which inputs 2, 3, 11 and 6 are selected as the representatives of the quadruples in SC'_1, SC''_1, SC'_2 and SC''_2, in that order.

Now we proceed to describe the second phase of the algorithm. Given a unicast k'-assignment, the second phase can take as many as $\lg\lceil k\rceil$ iterations, where $k = \min\{\lfloor(k'+2)/2\rfloor, n/2\}$. This is because there is no prior information about the exact sizes of the subchains and $\min\{\lfloor(k'+2)/2\rfloor, n/2\}$ is an upper bound for the sizes of the subchains with respect to the k'-assignment as stated in the previous section. Technically, each iteration can be further decomposed into three steps. In the first step, those quadruples whose first elements are 1 are duplicated and their copies are moved to new processors specified by their third elements. In the second step, each processor updates the quadruple(s) that it holds as follows: 1) when it holds only a quadruple, it "discards" the quadruple if the first element of the quadruple is 1; otherwise it keeps the quadruple intact; 2) when it holds $\langle 1; l, i; p_l\rangle$ and $\langle 1; i, j; p_i\rangle$, it replaces the first quadruple by $\langle 1; l, j; p'_l\rangle$ where $p'_l = \min\{p_i, p_l\}$ and discards the second quadruple; 3) when it holds $\langle 1; l, i; p_l\rangle$ and $\langle 0; i, *; j\rangle$ where $*$ is -2 or -1, it replaces the first quadruple by $\langle 1; l, *; i\rangle$ and keeps the second quadruple intact. In the third step, each updated quadruple is moved to the processor specified by its second element if its first element remains 1, and its first element is changed to 0 if its third element is -1 or -2. For example, we illustrate the second phase in (a) to (e) of Fig. 6, where the transitions from (a) to (b) and from (b) to (c) constitute the first iteration, and the transitions from (c) to (d) and from (d) to (e) constitute the second iteration. Note that, after



(a)

PRs	original quadruples
0	<1;0,11;0>
1	<1;1,13;1>
2	<1;2,8;2>
3	<1;3,15;3>
4	<1;4,9;4>
5	<1;5,14;5>
6	<0;6,-2;6>
7	<1;7,12;7>
8	<1;8,5;5>
9	<1;9,3;3>
10	<1;10,1;1>
11	<0;11,-1;11>
12	<1;12,0;0>
13	<1;13,6;6>
14	<1;14,2;2>
15	<1;15,4;4>

(b)

PRs	duplicated and moved qaudruples	original quadruples
0	<1;12,0;0>	<1;0,11;0>
1	<1;10,1;1>	<1;1,13;1>
2	<1;14,2;2>	<1;2,8;2>
3	<1;9,3;3>	<1;3,15;3>
4	<1;15,4;4>	<1;4,9;4>
5	<1;8,5;5>	<1;5,14;5>
6	<1;13,6;6>	<0;6,-2;6>
7		<1;7,12;7>
8	<1;2,8;2>	<1;8,5;5>
9	<1;4,9;4>	<1;9,3;3>
10		<1;10,1;1>
11	<1;0,11;0>	<0;11,-1;11>
12	<1;7,12;7>	<1;12,0;0>
13	<1;1,13;1>	<1;13,6;6>
14	<1;5,14;5>	<1;14,2;2>
15	<1;3,15;3>	<1;15,4;4>

(c)

PRs	updated quadruples
0	<0;0,-1;11>
1	<1;1,6;1>
2	<1;2,5;2>
3	<1;3,4;3>
4	<1;4,3;3>
5	<1;5,2;2>
6	<0;6,-2;6>
7	<1;7,0;0>
8	<1;8,14;5>
9	<1;9,15;3>
10	<1;10,13;1>
11	<0;11,-1;11>
12	<1;12,11;0>
13	<0;13,-2;6>
14	<1;14,8;2>
15	<1;15,9;4>

(d)

PRs	duplicated and moved qaudruples	updated quadruples
0	<1;7,0;0>	<0;0,-1;11>
1		<1;1,6;1>
2	<1;5,2;2>	<1;2,5;2>
3	<1;4,3;3>	<1;3,4;3>
4	<1;3,4;3>	<1;4,3;3>
5	<1;2,5;2>	<1;5,2;2>
6	<1;1,6;1>	<0;6,-2;6>
7		<1;7,0;0>
8	<1;14,8;2>	<1;8,14;5>
9	<1;15,9;4>	<1;9,15;3>
10		<1;10,13;1>
11	<1;12,11;0>	<0;11,-1;11>
12		<1;12,11;0>
13	<1;10,13;1>	<0;13,-2;6>
14	<1;8,14;5>	<1;14,8;2>
15	<1;9,15;3>	<1;15,9;4>

(e)

PRs	newly updated quadruples
0	<0;0,-1;11>
1	<0;1,-2;6>
2	<1;2,2;2>
3	<1;3,3;3>
4	<1;4,4;3>
5	<1;5,5;2>
6	<0;6,-2;6>
7	<0;7,-1;11>
8	<1;8,8;2>
9	<1;9,9;3>
10	<0;10,-2;6>
11	<0;11,-1;11>
12	<0;12,-1;11>
13	<0;13,-2;6>
14	<1;14,14;2>
15	<1;15,15;3>

(f)

PRs	final quadruples
0	<0;0,0;11>
1	<0;1,-2;6>
2	<1;2,2;2>
3	<1;3,3;3>
4	<1;4,4;3>
5	<1;5,5;2>
6	<0;6,-2;6>
7	<0;7,1;11>
8	<1;8,8;2>
9	<1;9,9;3>
10	<0;10,-2;6>
11	<0;11,0;11>
12	<0;12,1;11>
13	<0;13,-2;6>
14	<1;14,14;2>
15	<1;15,15;3>

Fig. 6. The second and third phases of the parallel algorithm that follow the first phase shown in Fig. 4.

the computation of the second phase, the two quadruples held by $PR(i)$ and $PR(\bar{i})$ correspond to inputs i and $\bar{i}$ of $SW_{\lfloor i/2 \rfloor}$. Besides, if the first elements of the two quadruples are 1, $SW_{\lfloor i/2 \rfloor}$ belongs to a closed chain; otherwise, it belongs to an open chain. Furthermore, $SW_{\lfloor i/2 \rfloor}$ belongs to a full open chain if the third elements are both -1 or -2, and to a half open chain if one of the third elements is -1 and the other is -2.

3) Third Phase: The third phase assigns types of settings to half open chains as discussed in the proof of Theorem 1 and using the pairing process as described in Section IV-A. At the end of the pairing process, the third element in the quadruple corresponding to the busy input of each semi-busy end switch is changed to 0 if the corresponding processor is marked type 0 or changed to 1 if it is marked type 1. For example, Fig. 6(f) shows a result of the third phase, where the third elements in the quadruples in $PR(0), PR(11)$ are changed to 0 and the third elements in the quadruples in $PR(7), PR(12)$ are changed to 1.

4) Fourth Phase: Having decided the types of each chain and computed the representative of each quadruple, the fourth phase determines the settings of switches in $H(n)$. Each switch is set by a dual pair of processors which hold the two quadruples that correspond to the inputs of this switch, and the setting depends on the type of the chain to which this switch belongs.

Case 1: If $PR(i)$ holds $\langle 1; i, k; j \rangle$ and $PR(\bar{i})$ holds $\langle 1; \bar{i}, l; \bar{j} \rangle$, then $SW_{\lfloor i/2 \rfloor}$ is in a closed chain. Assuming that $SW_{\lfloor j/2 \rfloor}$ is set *through*, $SW_{\lfloor i/2 \rfloor}$ must be set *through* if $i - j$ is even and *cross* otherwise.

Case 2: If $PR(i)$ holds $\langle 0; i, p; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, p; l \rangle$ where $p = -1$ or $p = -2$, then $SW_{\lfloor i/2 \rfloor}$ is in a full open chain, and $SW_{\lfloor j/2 \rfloor}$ and $SW_{\lfloor l/2 \rfloor}$ are the end switches. Assuming that the smaller one of $SW_{\lfloor j/2 \rfloor}$ and $SW_{\lfloor l/2 \rfloor}$ is set *through*, when $j < l, SW_{\lfloor i/2 \rfloor}$ must be set *through* if $i - j$ is even and *cross* otherwise, and when $l < j$, $SW_{\lfloor i/2 \rfloor}$ must be set *through* if $\bar{i} - l$ is even and *cross* otherwise.

Case 3: If $PR(i)$ holds $\langle 0; i, q; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, -2; l \rangle$ where $q = 0$ or 1, then $SW_{\lfloor i/2 \rfloor}$ is in a half open chain and $SW_{\lfloor j/2 \rfloor}$ is the semi-busy end switch. When $q = 0$ (the half open chain is assigned *Type-0 Setting*), $SW_{\lfloor i/2 \rfloor}$ must be set *through* if i is even and *cross* otherwise, and when $q = 1$ (the half open chain is assigned *Type-1 Setting*), $SW_{\lfloor i/2 \rfloor}$ must be set *through* if i is odd and *cross* otherwise. For example, switches SW_0, SW_1, SW_4, SW_6 and SW_7 in Fig. 3 are set *through* and switches SW_2, SW_3 and SW_5 are set *cross* by checking the final quadruples held in each dual pair of processors as shown in Fig. 6(f).

C. The Parallel Algorithm

The following parallel routing algorithm formalizes the steps outlined in the previous subsection.

Step 1: Given $(i, (r_i, d^i_{\lg n-1}, \cdots, d^i_1, d^i_0))$ input to $PR(i), PR(i)$ establishes $\langle 0; i, -2; i \rangle$ if $r_i = 0, 0 \leq i \leq n-1$. Let k' be the number of r_i's whose value are 1, and let $k = \min \{\lfloor (k'+2)/2 \rfloor, n/2\}$. Let m be a parameter initialized to 0.

Step 2: Move $(i, (r_i, d^i_{\lg\ n-1}, \cdots, d^i_1, d^i_0))$ to $PR(x)$ if $r_i = 1$ and $(d^i_{\lg\ n-1}, \cdots, d^i_1, d^i_0)$ is the binary representation of $x, 0 \le i \le n-1$.

Step 3: Given $PR(x)$ holding (i, r_i) and $PR(\bar{x})$ holding $(j, r_j), PR(x)$ establishes $\langle 1; i, \bar{j}; p_i \rangle$ and $PR(\bar{x})$ establishes $\langle 1; j, \bar{i}; p_j \rangle$ where $p_i = \min\{i, \bar{j}\}$ and $p_j = \min\{j, \bar{i}\}$ if $r_i = r_j = 1$, or $PR(x)$ establishes $\langle 1; i, -1; i \rangle$ if $r_i = 1$ and $r_j = 0$, or $PR(\bar{x})$ establishes $\langle 1; j, -1; j \rangle$ if $r_i = 0$ and $r_j = 1, 0 \le x \le n-1$. Then, $\langle 1; i, j; p_i \rangle$ is moved to $PR(i)$, and if the third element is "-1" then $PR(i)$ changes the first element to 0, $0 \le i \le n-1$. (Step 1, Step 2 and Step 3 constitute the first phase.)

Step 4: $m = m + 1$. If $m \le \lceil \lg\ k \rceil$ then go to Step 5, else go to Step 8.

Step 5: Duplicate $\langle 1; l, i; p_l \rangle$ and move a copy to $PR(i), 0 \le i \le n-1$. (Those quadruples whose first elements are 1 are duplicated and moved to new processors specified by their third elements.)

Step 6: The action of each processor depends on the quadruple(s) that it holds:

1) when $PR(i)$ holds only 1 quadruple: If the first element of the quadruple is 1 then $PR(i)$ discards the quadruple, else $PR(i)$ keeps the quadruple intact;
2) when $PR(i)$ holds $\langle 1; l, i; p_l \rangle$ and $\langle 1; i, j; p_i \rangle : PR(i)$ replaces the first quadruple by $\langle 1; l, j; p'_l \rangle$ where $p'_l = \min\{p_i, p_l\}$ and discards the second quadruple;
3) when $PR(i)$ holds $\langle 1; l, i; p_l \rangle$ and $\langle 0; i, *; j \rangle$ where $*$ is -2 or -1: $PR(i)$ replaces the first quadruple by $\langle 1; l, *; i \rangle$ and keeps the second quadruple intact.

Step 7: Move $\langle 1; i, j; p_i \rangle$ to $PR(i)$, and if the third element is "-1" or "-2" then $PR(i)$ changes the first element to 0, $0 \le i \le n-1$. Go to Step 4. (Step 4, Step 5, Step 6 and Step 7 constitute the second phase.)

Step 8: If $PR(i)$ holds $\langle 0; i, -1; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, -2; \bar{i} \rangle$ (i.e., input i is the busy input of $SW_{\lfloor i/2 \rfloor}$ which is the semi-busy switch of a half open chain), then $PR(i)$ is marked type 0 or type 1 by the pairing process, $0 \le i \le n-1$. If $PR(i)$ is marked type 0 then $\langle 0; i, -1; j \rangle$ is changed to $\langle 0; i, 0; j \rangle$. Otherwise, it is changed to $\langle 0; i, 1; j \rangle$. (Step 8 constitutes the third phase.)

Step 9:

Case 1: If $PR(i)$ holds $\langle 1; i, j; p_i \rangle$ and $PR(\bar{i})$ holds $\langle 1; \bar{i}, l; \bar{p}_i \rangle$, then $PR(i)$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $i - p_i$ is even and *cross* otherwise, $0 \le i \le n-1$.

Case 2: If $PR(i)$ holds $\langle 0; i, p; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, p; l \rangle$ where $p = -1$ or $p = -2$, then they compare j and l. When $j < l, PR(i)$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $i - j$ is even and *cross* otherwise, and when $l < j, PR(\bar{i})$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $\bar{i} - l$ is even and *cross* otherwise, $0 \le i \le n-1$.

Case 3: If $PR(i)$ holds $\langle 0; i, q; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, -2; l \rangle$ where $q = 0$ or $q = 1, PR(i)$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $i - q$ is even and *cross* otherwise, $0 \le i \le n-1$. (Step 9 constitutes the fourth phase.) ||

That this algorithm is correct can easily be proved and is omitted for lack of space. In the algorithm, move processes dominate the time complexity. As shown in Section IV-A, each move process can be executed in $O(1)$ time if the processors are completely interconnected, and in $O(\lg^2\ k + \lg\ n)$ time if the processors are extended shuffle-exchange interconnected. For a unicast k-assignment, since there are $O(\lg\ k)$ move processes in the algorithm, the switches in $H(n)$ can be set in $O(\lg\ k)$ time if the processors are interconnected by a completely connected network, and in $O(\lg^3\ k + \lg\ k \lg\ n)$ time if they are interconnected by an extended shuffle-exchange network. This algorithm can be recursively applied to set switches in the first half stages. In fact, only the first $\lfloor \lg\ k \rfloor$ stages would apply this algorithm for a unicast k-assignment since this algorithm decomposes an assignment into two half-sized assignments stage by stage as required in the statement of Theorem 1. Therefore, by using this parallel algorithm, the routing time for an n-input Beneš network to realize a unicast k-assignment is $O(\lg^2\ k + \lg\ n)$ if the interprocessor connection topology is complete, and is $O(\lg^4\ k + \lg^2\ k \lg\ n)$ if the interprocessor connection topology is extended shuffle-exchange network.

V. Concluding Remarks

The paper presented an extension of Nassimi and Sahni's parallel routing algorithm for the Beneš network to unicast assignments. For any unicast k-assignment, the algorithm takes $O(\lg^2\ k + \lg\ n)$ time on the n-processor completely connected network and it takes $O(\lg^4\ k + \lg^2\ k\ \lg\ n)$ time on the n-processor extended perfect shuffle-exchange network. The algorithm can easily be pipelined to route a set of unicast assignments using $O(n\ \lg\ n)$ processors. Assuming that we have α unicast assignments that need to be realized on an n-input Beneš network, it can be shown that, with pipelining, the total routing time is $O(\lg^2\ k + \lg\ n + (\alpha - 1)\ \lg\ k)$ for completely connected topology, and is $O(\lg^4\ k + \lg^2\ k\ \lg\ n + (\alpha - 1)(\lg^3\ k + \lg\ k\ \lg\ n))$ for extended shuffle-exchange topology. Thus, when $\alpha \ge \lg\ n$, the average routing time to realize a unicast k-assignment is reduced with pipelining to $O(\lg\ k)$ in the first case, and to $O(\lg^3\ k + \lg\ k\ \lg\ n)$ in the second case.

References

[1] V. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic.* New York: Academic, 1965.

[2] C. Cardot, "Comments on a simple algorithm for the control of rearrangeable switching networks," *IEEE Trans. Commun.*, p. 395, Apr. 1986.

[3] J. Carpinelli and A. Y. Oruç, "Applications of edge-coloring algorithms to routing in parallel computers," in *Proc. Int. Conf. Supercomp.*, Santa Clara, CA, May 1988.

[4] T. Cormen, C. E. Lesiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: The MIT Press, 1990, pp. 692–701.

[5] M. J. Karol and Chih-Lin I, "Performance analysis of a growable architecture for broadband packet (ATM) switching," *IEEE Trans. Commun.*, pp. 431–439, Feb. 1992.

[6] F. R. Hwang, "Control algorithms for rearrangeable Clos networks," *IEEE Trans. Commun.*, pp. 952–954, Aug. 1983.

[7] J. Konicek *et al.*, "The organization of the cedar system," in *Proc. Int. Conf. Parallel Processing*, St. Charles, IL, Aug. 1991, pp. 49–56.

[8] D. Knuth, *The Art of Computer Programming: Sorting and Searching.* Reading, MA: Addison-Wesley, 1973.

[9] K. Y. Lee, "A new Benes network control algorithm," *IEEE Trans. Comput.*, pp. 768–772, June 1987.

[10] G. F. Lev, N. Pippenger, and L. Valiant, "A fast parallel routing algorithm for routing in permutation networks," *IEEE Trans. Comput.*, pp. 93–100, Feb. 1981.

[11] D. Nassimi and S. Sahni, "A self-routing Beneš network and parallel permutation algorithms," *IEEE Trans. Comput.*, pp. 332–340, May 1981.

[12] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Beneš network," *IEEE Trans. Comput.*, pp. 148–154, Feb. 1982.

[13] D. Opferman and N. Tsao-Wu, "On a class of rearrangeable switching networks," *Bell Syst. Tech. J.*, pp. 1579–1618, May–June 1971.

[14] H. J. Siegel *et al.*, "Using the multistage cube network topology in parallel supercomputers," *Proc. IEEE*, pp. 1932–1953, Dec. 1989.

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

[16] A. Waksman, "A permutation network," *J. ACM*, pp. 159–163, Jan. 1968.