Short Notes

A Fast Parallel Algorithm for Routing
Unicast Assignments in Benes Networks

Ching-Yi Lee and A. Yavuz Oruç

Abstract—This paper presents a new parallel algorithm for routing unicast (one-to-one) assignments in Benes 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 increasing 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 $O(k + \lg n)$ time on a completely connected network of $n$ processors and in $O((k^2 + k + \lg k) \lg n)$ time on an extended shuffle-exchange network of $n$ processors. Using the new algorithm, one can be pipelined to route $O(k)$ unicast assignments, each involving $O(k)$ pairs of inputs and outputs, in $O((k^2 + k + \lg n + (\alpha - 1)) T_{\text{seq}})$ time on a completely connected network and in $O((k^2 + k + \lg n + (\alpha - 1)) T_{\text{seq}})$ time on the extended shuffle-exchange network. These yield an average routing time of $O(k \lg n)$ in the first case, and $O((k^2 + k + \lg n))$ 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 at least as fast as the algorithm presented by Vatan et al. [9]–[12]. 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 Benes network has received much attention in interconnection network literature because of its $O(n \lg n)$ 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 Benes and Clos networks for ATM switching due to their expandability 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 Benes network including the $O(n \lg n)$ time recursive looping procedure [13], [16], nonrecursive complete residue partition tree algorithm [9] and $O((\lg n)^2)$ time parallel algorithms of Lev et al. [10], and Nassimi and Sahni [12]. Other routing algorithms for the Benes 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 Benes 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 $k < n$.

In this paper, we present an efficient parallel algorithm for routing incomplete assignments on the Benes 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 $O((\lg k^2 + \lg n))$ 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 $O((\lg k^2 + \lg n) \lg n)$ time without pipelining and $O((\lg k^2 + \lg k) \lg n)$ 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 Benes network is a unicast $n$-network that is constructed recursively as shown in Fig. 1. Each $2 \times 2$ 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.
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 Benes 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 \((r_1, d_{lg n-1}, \ldots, d_{1}, d_0)\) for a packet at input \(i\). The bit \(r_1\) specifies whether input \(i\) is paired with some output. Input \(i\) is said to be busy if \(r_1 = 1\), and it is said to be idle if \(r_1 = 0\). Assuming that \(r_1 = 1\), the remaining bits, i.e., \((d_{lg n-1}, \ldots, d_{1}, d_0)\), form the output address which specifies the binary representation of the output paired with input \(i\) with \(d_{lg n-1}\) 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.

III. THE ROUTING PRINCIPLE

Unicast assignments for an n-input Benes 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 \((b_{lg n-1}, \ldots, b_1, b_0)\) is the binary representation of \(i\), then \(\bar{i}\) denotes the integer which has the binary representation \((b_{lg n-1}, \ldots, b_1, b_0)\), and \(i\) 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 Benes network and comprising \(n/2 \times 2\) switches, \(SW_0, SW_1, \ldots, SW_{n/2-1}\), as shown in Fig. 2.

**Definition 1:** Given a unicast assignment \(\{(i, (r_1, d_{lg n-1}, \ldots, d_{1}, d_0))\); \(0 \leq i \leq n - 1\}\) for \(H(n)\), a sequence of switches \(SW_0, SW_1, \ldots, SW_{n/2-1}\) 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 \(SW_q\) and one of the inputs of \(SW_{p+1}\) are paired with a dual pair of outputs and have their \(r_1\),'s set to 1. Furthermore, \(SW_0, SW_1, \ldots, SW_{n/2-1}\) is said to be a closed chain if the other input of \(SW_{n/2-1}\) and the other input of \(SW_0\) 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 \(\min\{\lceil(k + 2)/2\rceil, n/2\}\) and \(2^k\) 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 \(\lceil(k + 2)/2\rceil\) or \(\lceil k/2 \rceil\) in size. 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 \(SW_0, SW_1, \ldots, SW_{n/2-1}\), has two end switches (i.e., \(SW_0\) and \(SW_{n/2-1}\)). 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 \(\{(i, (r_1, d_{lg n-1}, \ldots, d_{1}, d_0))\); \(0 \leq i \leq n - 1\}\), a unicast \(k\)-assignment for \(H(n)\), \(let (r_1, p_{lg n-1}, \ldots, p_1, p_0)\) 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 \(SW_0, SW_1, \ldots, SW_{n/2-1}\) such that \(\{(2i, (r_1, p_{lg n-1}, \ldots, p_1, p_0))\); \(0 \leq i \leq n/2 - 1\}\) is a unicast \(k_0\)-assignment and \(\{2(i + 1), (r_1, p_{lg n-1}, \ldots, p_1, p_0)\}; \(0 \leq i \leq n/2 - 1\}\) is a unicast \(k_1\)-assignment, where \(k_0 = \lceil k/2 \rceil\) and \(k_1 = \lfloor k/2 \rfloor\).

\(^3\lceil x \rceil\) denotes the largest integer equal to or larger than \(x\), and \(\lfloor x \rfloor\) denotes the smallest integer equal to or larger than \(x\).
Proof: With respect to the given unicast-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 $(d_{k-1}^i, \ldots, d_0^i, d_0^j) = (d_{k-1}^j, \ldots, d_0^j, d_0^i)$, 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 $\{(2i, (2i+1, P_{2^n-1}^{2^n-1}, \ldots, P_{2^1}^{2+1}, P_{2^1}^{2+1})): 0 \leq i \leq n/2 - 1\}$ is a unicast $k$-assignment and $\{(2i + 1, (2i+1, P_{2^n-1}^{2^n-1}, \ldots, P_{2^1}^{2+1}, P_{2^1}^{2+1})): 0 \leq i \leq n/2 - 1\}$ is a unicast $k$-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 chains are set in such a way that $\lceil k/2 \rceil$ of the $f$ half open chains are in their type-0 settings and the other $\lfloor k/2 \rfloor$ 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 Benes network then follows. It is assumed that there are $n$ interprocessor connections, $PR(0), PR(1), \ldots, PR(n - 1)$, and that a packet header $(r_i, d_{k-1}^i, \ldots, d_0^i)$ is initially input to $PR(i)$. $PR(i)$ and $PR(j)$ are called a dual pair of processors and will determine the setting of $SW_{[i, j]}$ 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 $\lceil k/2 \rceil$ of them are marked type 0 and $\lfloor k/2 \rfloor$ 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 $(2\lceil \lg k \rceil)$-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 $PR(0), PR(1), \ldots, PR(n/2 - 1)$ are $(n/2^c)$-shuffle connected for $m = 0, 1, \ldots, \lg n - 2$, (i.e., $PR(0), PR(1), \ldots, PR(n - 1)$ is $n$-shuffle connected, $PR(0), PR(1), \ldots, PR(n/2 - 1)$ 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 $i$ 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 $PR(0), PR(1), \ldots, PR(k' - 1)$, 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 switches of inputs 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 I:** Let $\{(i, (r_i, d_{k-1}^i, \ldots, d_0^i)): 0 \leq i \leq n - 1\}$ be a unicast assignment for $H(n)$. If $(d_{k-1}^i, \ldots, d_0^i) = (d_{k-1}^j, \ldots, d_0^j)$, $r_i = r_j = 1$ and $i \neq j$, then inputs $i$ and $j$ are in the same equivalence class and inputs $j$ and $i$ are in another equivalence class.

**Remark I:** This equivalence among the inputs will be noted by associating each input with an ordered quadruple $(c_i, i, e_i, p_i)$ 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...
Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 30, 2009 at 14:50 from IEEE Xplore. Restrictions apply.
the computation of the second phase, the two quadruples held by \( PR(i) \) and \( PR(i) \) correspond to inputs \( i \) and \( i \) of \( SW / \frac{1}{2} \). Besides, if the first elements of the two quadruples are 1, \( SW / \frac{1}{2} \) belongs to a closed chain; otherwise, it belongs to an open chain. Furthermore, \( SW / \frac{1}{2} \) 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 \((0; i, p, j)\) and \( PR(i) \) holds \((0; i, p, j)\), then \( SW / \frac{1}{2} \) is in a closed chain. Assuming that \( SW / \frac{1}{2} \) is set through, \( SW / \frac{1}{2} \) must be set through if \( i - j \) is even and cross otherwise.

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

Case 3: If \( PR(i) \) holds \((0; i, g, j)\) and \( PR(i) \) holds \((0; i, g, j)\) where \( q = 0 \) or 1, then \( SW / \frac{1}{2} \) is in a half open chain and \( SW / \frac{1}{2} \) is the semi-busy end switch. When \( q = 0 \) (the half open chain is assigned Type-0 Setting), \( SW / \frac{1}{2} \) must be set through if \( i \) is even and cross otherwise, and when \( q = 1 \) (the half open chain is assigned Type-1 Setting), \( SW / \frac{1}{2} \) must be set through if \( i \) is odd and cross otherwise. For example, switches \( SW_2, SW_4, SW_6 \) and \( SW_7 \) in Fig. 3 are set through and switches \( SW_3, SW_5 \) 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 \((i, \{r, d_{x, i}, d_{x, i} \}, n)\) input to \( PR(i) \), \( PR(i) \) establishes \((0; i, -2; i) \) if \( r_i = 0, 0 \leq i \leq n-1 \). Let \( k' \) be the number of \( r_i \)'s whose value is 1, and let \( k = \min \{ [(k' + 2)/2], n/2 \} \). Let \( m \) be a parameter initialized to 0.
Step 2: Move \((i_1, r_1, d_{i_1}^{i_2}, \ldots, d_{i_1}^{i_k})\) to \(PR(i)\) if \(r_1 = 1\) and \((d_{i_1}^{i_2}, \ldots, d_{i_1}^{i_k})\) is the binary representation of \(x, 0 \leq i \leq n - 1\).

Step 3: Given \(PR(x)\) holding \((r_1, j_1, p_1)\) and \(PR(y)\) holding \((r_2, j_2, p_2)\), \(PR(z)\) establishes \((i_1, r_1, j_1, p_1)\) and \(PR(x)\) establishes \((i_2, r_2, j_2, p_2)\) where \(p_1 = \min\{i, j\}\) and \(p_2 = \min\{i, j\}\) if \(r_1 = r_2 = 1\), or \(PR(z)\) establishes \((i_1, 1, j_1, i_2)\) if \(r_1 = 1\) and \(r_2 = 0\), or \(PR(z)\) establishes \((i_1, j_1, i_2)\) if \(r_1 = 0\) and \(r_2 = 1, 0 \leq i \leq n - 1\). Then, \((i_1, j_1, p_1, i_2)\) is moved to \(PR(i)\), and if the third element is "1" then \(PR(i)\) changes the first element to 0, \(0 \leq i \leq n - 1\). (Step 1, Step 2 and Step 3 constitute the first phase.)

Step 4: \(m = m + 1\). If \(m \leq \lceil \log k \rceil\) then go to Step 5, else go to Step 8.

Step 5: Duplicate \((i_1, i_2, p_1)\) and move a copy to \(PR(i)\), \(0 \leq i \leq n - 1\). (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 \((1, i, p_1)\) and \((1, i, j, p_2)\): \(PR(i)\) replaces the first quadruple by \((1, 1, j, p_2)\) where \(p_1 = \min\{i, j\}\) and discards the second quadruple;
3) when \(PR(i)\) holds \((1, i, p_1)\) and \((0, i, j, p_2)\) where \(i = 2\) or \(-1\): \(PR(i)\) replaces the first quadruple by \((1, i, *; p_2)\) and keeps the quadruple intact.

Step 7: Move \((i_1, j_1, p_1)\) to \(PR(i)\), and if the third element is "1" or "2" then \(PR(i)\) changes the first element to 0, \(0 \leq i \leq n - 1\). Go to Step 4. (Step 4, Step 5, Step 6 and Step 7 constitute the second phase.)

Step 8: If \(PR(i)\) holds \((i_1, i_2; p_1)\) and \(PR(i)\) holds \((0, i_2; p_2)\) then \(PR(i)\) sets \(SW_{i_1}^{i_2}(\theta_2)\) through \(i - p_1\) is even and \(SW_{i_1}^{i_2}(\theta_2)\) otherwise, \(0 \leq i \leq n - 1\).

Step 9: If \(PR(i)\) holds \((i_1, j_1, p_1)\) and \(PR(i)\) holds \((i_1, i_2, \cdot; p_2)\), then \(PR(i)\) sets \(SW_{i_1}^{i_2}(\theta_2)\) through \(i - p_1\) is even and \(SW_{i_1}^{i_2}(\theta_2)\) otherwise, \(0 \leq i \leq n - 1\).

That this algorithm can be recursively applied to set switches in the first half stages. In fact, only the first \(\lceil \log k \rceil\) 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 Benes network to realize a unicast \(k\)-assignment is \(O(\log^2 k + \log n)\) if the interprocessor connection topology is complete, and is \(O(\log^4 k + \log^2 k \log n)\) if the interprocessor connection topology is extended shuffle-exchange network.

V. CONCLUDING REMARKS

The paper presented an extension of Nasimi and Sahni's parallel routing algorithm for the Benes network to unicast assignments. For any unicast \(k\)-assignment, the algorithm takes \(O(\log^2 k + \log n)\) time on the \(n\)-processor completely connected network and it takes \(O(\log^4 k + \log^2 k \log n)\) time on the \(n\)-processor extended perfect shuffle-exchange network. The algorithm can easily be pipelined to route a set of unicast assignments using \(O(n \log n)\) processors.

REFERENCES