A Fast Parallel Routing Algorithm for Benes Group Switches

Enyue Lu and S. Q. Zheng
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688, USA
{enyue, sizheng}@utdallas.edu

ABSTRACT
A parallel routing algorithm for controlling the class of interconnection networks called group connectors is presented. Given any legal mapping from input to output groups with $O(K)$ busy inputs, this algorithm can determine the switch setting of a Benes group connector with N inputs and n output groups in $O(\log^2 K + \log N)$ time on a completely connected computer or the EREW PRAM model with N processing elements. The implementations of this algorithm on various realistic parallel machine models are also discussed.

KEY WORDS
Group Connector, Rearrangeable Nonblocking Interconnection Network, Parallel Algorithm, Dense Wavelength-Division Multiplexing (DWDM), Network Switch, Network Router.

1 Introduction

An $N \times N$ permutation network is an interconnection network consisting of N inputs, N outputs, switching elements (SEs) and fabrics. Such a network can achieve any one-to-one correspondence between inputs and outputs by properly setting up SEs to establish link-disjoint paths from inputs to their corresponding outputs. An $N \times N$ crossbar switching network is a straightforward permutation network. To improve the scalability of crossbar switches, several permutation networks with reduced number of SEs were proposed. Most noticeable ones include Clos networks and Benes networks. Permutation networks are widely used as switching matrices in network routers and switches.

Recently, a new class of interconnection networks called group connectors were introduced [18]. A *group connector* $G(N, n)$ is defined as an interconnection network that consists of N inputs and N outputs such that (1) its N outputs are divided into n output groups with N/n functionally equivalent outputs in each group; and (2) it can provide any simultaneous (N/n)-to-one connections from N inputs to n output groups, possibly without the ability of distinguishing the order of outputs within each group. In this paper, we always assume that $N = 2^m$, $n = 2^{m-k}$, $0 \le k \le m$ and $0 \le K \le N$ where K is the number of busy inputs. Another type of $N \times N$ group connector $G'(N, n)$ can be defined by dividing its N inputs and N outputs into n equal-size groups, respectively. For $G'(N, n)$, if the inputs in the same input group are allowed to be connected to the outputs in different output groups, $G'(N, n)$ and $G(N, n)$ are the same in functionality; otherwise n separate planes of $N/n \times N/n$ permutation networks can be used to implement $G'(N, n)$.

Clearly, an $N \times N$ permutation network is a group connector $G(N, n)$ with $n = N$. Nonblocking and rearrangeable nonblocking group connectors based on Clos and Benes networks were proposed in [18], and it was shown that a group connector $G(N, n)$ can be built at a lower hardware cost than that of a permutation network of the same size. For example, an $N \times N$ Benes network, denoted by $B(N)$, is a rearrangeable nonblocking permutation network with $(N/2) \cdot (2 \log N - 1)$ SEs, and a rearrangeable nonblocking $G(N, N/2^k)$ based on $B(N)$ can be built with $(N/2) \cdot (2 \log N - 1 - k)$ SEs. The saving in SEs by using $G(N, N/2^k)$ is $(N/2) \cdot k$. (In this paper, all logarithms are in base 2 and all SEs are in size of 2×2.)

In general, a group connector $G(N, n)$ captures the simultaneous connections between N clients and N servers which are divided into n equal-size server groups such that the N/n servers in each group are functionally equivalent. A group connector $G(N, n)$ can also be viewed as an $n \times n$ permutation network with internal speedup of factor N/n achieved by space-division multiplexing [17].

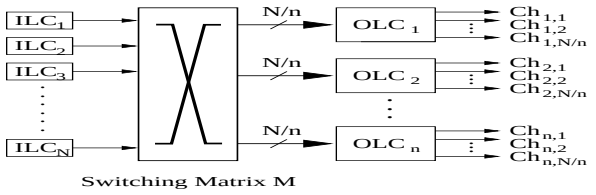

Figure 1: Block diagram of an ingress edge router.

Group connectors are particularly useful in dense wavelength-division multiplexing (DWDM) networks. With DWDM, it is now possible to transmit different wavelengths of light over the same fiber, which has provided another dimension to increase bandwidth capacity. A group connector can be used as a switching network in a DWDM router. For example, if some inputs and one or more groups of outputs are connected to a local node, a group connector can be used as an add-drop cross-connect switching matrix. Group connectors have applications in the construction of ingress edge routers of DWDM networks. An ingress edge router in a DWDM optical network has a set of N electrical or optical input links and a set of n optical output links. Each optical output link i consists of a set of N/n data channels $Ch_{i,1}, \cdots, Ch_{i,N/n}$, each using a different wavelength. Associated with each input link, there is an input line card (ILC) and associated with each output link there is an output line card (OLC). There is a switching matrix M between ILCs and OLCs. There are N/n connections from the output of M to each OLC. The main function of each ILC is to route input packets to appropriate OLCs by routing table lookup. Each OLC transmits the packets it received using n optical channels of the link it controls. The block diagram of a DWDM ingress edge router is shown in Figure 1. A group connector $G(N, n)$ served as the

major switching matrix M in the design of ingress edge routers of a burst-switched DWDM network [19].

A permutation network is *rearrangeable nonblocking* if it can realize all possible permutation connections between inputs and outputs when the rearrangement to existing connections is permitted [2]. Similarly, a group connector is rearrangeable nonblocking if it can realize all possible connections between the inputs and group outputs when the rearrangement to existing connections is permitted. We show that $G(N,n)$ is a rearrangeable nonblocking group connector in the theorem 1 and call a rearrangeable nonblocking group connector $G(N,n)$ based on Benes network a *Benes group connector*, or a *Benes group switch*. Rearrangeable nonblocking networks, including Benes networks and Benes group connectors, are very attractive for fixed-size cell switching architecture. In such a switch, variable length packets are segmented into cells upon arrival, transferred across the switch matrix, and then reassembled again before they depart. Using fixed-size cells allows for slotted switching, which makes it easier for the scheduler to configure the switch matrix for high throughput.

When a group connector is used as a switching matrix in a high-speed packet router/switch, packet-forwarding speed is crucial. There are several factors that affect packet-forwarding speed: routing (label) table lookup, switch scheduling, switch setup and switch internal transmission. For group connectors, the implementations of switch scheduling and switch setup are of particular importance.

In this paper, we address the issue of how to quickly set up SEs so that K edge-disjoint paths between inputs and output groups are established in $G(N,n)$. In particular, we present a parallel algorithm, and its variations, for the setup of a Benes group connector $G(N,n)$ with K busy inputs. In this context, Benes permutation network $B(N)$ is a special case $G(N,N)$ of Benes group connectors. Thus, our algorithms can be applied to Benes networks directly. Given any permutation assignment, all known sequential algorithms for setting up the $B(N)$ take $O(N\log N)$ time [7, 13, 15], and the best time complexity of parallel algorithms is $O(\log^2 N)$ [11, 12]. Given any non-full permutation assignment involving $O(K)$ pairs of input and output, the parallel algorithms to set up Benes network in $O(\log^2 N)$ time and in $O(\log^2 K + \log N)$ time were proposed in [9] and [6] respectively. Our main algorithm extends the algorithm [6] to set up Benes group connector for non-maximum mapping between inputs and output groups, using techniques such as pointer jumping [12], parallel prefix sum [5] and parallel sorting [4]. As the algorithm of [7], our algorithm sets up SEs in the first $\log N - 1$ stages of $G(N,n)$ so that the SEs in the remaining stages can be set up by self-routing. Furthermore, as the algorithm of [15], our algorithm implies that additional $2N-4$ crossing points can be eliminated from Benes group connector $G(N,n)$. Consequently, the number of saved crossing points in Benes group connector $G(N,N/2^k)$ is increased to $2N\cdot k + 2N - 4$, compared with the $N\times N$ Benes network. On the other hand, given any non-maximum mapping with K busy inputs, our algorithm runs in $O(\log^2 K + \log N)$ time on a completely connected computer or the EREW PRAM model with N processing elements(PEs) as the algorithm of [6]. When implemented on a perfect shuffle computer and a hypercube of N PEs, $O(\log^4 K + \log^2 K\cdot\log N)$ time is sufficient. Our algorithm takes the advantages of algorithms of [7] and [6, 12]. To our knowledge, all known algorithms for setting up Benes networks $B(N)$ cannot be directly applied to set up Benes group connectors. However, by letting $n = N$, our algorithm for $G(N,n)$ can be directly applied to set up Benes network $B(N)$ with the same time complexity.

2 Preliminaries

A Benes network $B(N)$ is a multistage rearrangeable nonblocking permutation network [1]. Each SE in a Benes network has two inputs and two outputs, named sibling inputs and outputs. Each busy input is connected with one of sibling outputs. If the busy input is connected with output horizontally, we say that this input is set straight, and cross otherwise (Figure 2(a)). Benes networks are constructed recursively. A $B(N)$ consists of a switching stage with N SEs, an $N/2\times 2$ shuffle connection(i.e. the output i is connected to the input $S_{q\times r}(i) = (qi + \lfloor i/r \rfloor) \bmod qr$ in two continue stages [14]), followed by two $B(N/2)$, a $2\times N/2$ shuffle connection, and a switching stage with N SEs. A $B(16)$ is shown in Figure 2(b).

A Benes group connector $G(N,n)$ is constructed from a Benes network $B(N)$ by permanently setting all inputs in its last k stages straight, which leads to eliminating these SEs (see Figure 2 (c) for an example).

An *s-level subnetwork* $(0 \le s \le \log N - 1)$ of Benes group connector $G(N,n)$ is defined as a Benes group connector $G(H,h)$ so that $H = 2^{m-s}$ and $h = \min\{H,n\}$. Thus a $G(N,n)$ contains 2^s s-level subnetworks. Clearly, 0-level subnetwork of $G(N,n)$ is itself. Figure 2(c) shows a $G(16,4)$, which contains 2 1-level subnetworks $G(8,4)$, 4 2-level subnetworks $G(4,4)$, and 8 3-level subnetworks $G(2,2)$. All SEs in the first(resp. last) stage of subnetwork $G(H,h)$ are called input(resp. output) SEs of $G(H,h)$.

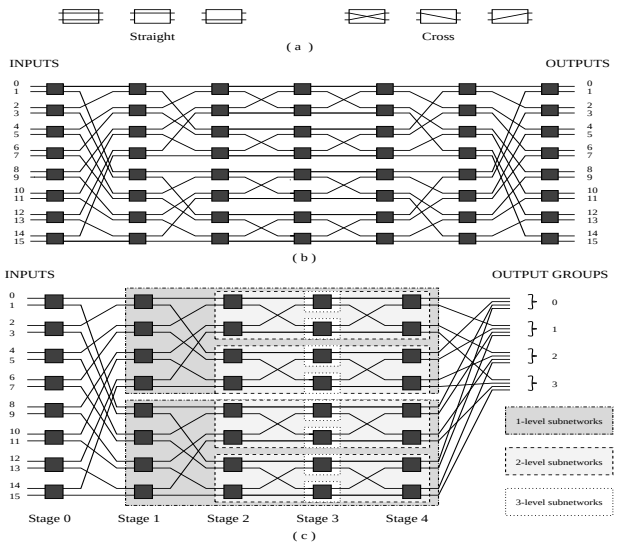

Figure 2: Construction of a Benes group connector: (a) States of SEs; (b) A 16×16 Benes network; (c) A Benes group connector $G(16,4)$ with $k=2$ based on 16×16 Benes network.

For a $G(N,n)$, we label the N inputs(outputs) as $0,1,\cdots,N-1$, n group outputs as $0,\cdots,n-1$, and $N/2$ SEs in each stage as $0,\cdots,N/2-1$ from top to bottom. And $2\log N-1$ stages are indexed 0 through $2\log N-2$ from left to right. Denote every input, output and output group of $G(N,n)$ by I_i, O_i, and G_j respectively, where $0\le i\le N-1$ and $0\le j\le n-1$. Thus O_i connects to G_j where $j = i \bmod n$ in the last stage of

$G(N,n)$ and every G_j consists of $O_{jN/n+l}$, $0 \le l \le N/n-1$. For a pair of sibling inputs(resp. outputs), one with even label is called upper input(resp. output) and another with odd label is called lower input(resp. output). Let I, O and G be the sets of N inputs, N outputs and n output groups of $G(N,n)$ respectively. Let $\pi : I \longmapsto G$ be an I/G *mapping* that indicates connection requests from inputs to output groups. If there is a connection request from I_i to O_j, set $\pi(i) = j$ and call I_i as a *busy* input; otherwise set $\pi(i) = -1$ and call I_i as an *idle* input. An I/G mapping from I to G is *legal* if each I_i is mapped to at most one G_j and at most N/n different I_i's are mapped to the same G_j. When used as a switching matrix, legal mappings can be enforced by using the arbitration hardware [16, 20]. A legal I/G mapping is *maximum* if all inputs are busy inputs, and *non-maximum* otherwise. We denote a legal I/G mapping involving K busy inputs as $\pi|K$. Clearly, if $K = N$, $\pi|N$ is a legal maximum I/G mapping. If an input SE has 2(1 or 0) busy inputs, it is called busy(semi-busy or idle) input SE. A Benes group connector $G(N,n)$ has a *feasible configuration* for a given $\pi|K$ if all SEs can be set up so that there are K edge-disjoint paths with each of them connecting the busy input I_i to output group $G_{\pi(i)}$. Thus, $G(N,n)$ is rearrangeable nonblocking if it has a feasible configuration for any $\pi|K$.

3 Graph Model

Each $G(N,n)$ with a legal mapping $\pi|K$ can be represented as a graph $\hat{G}$ as following:
Case 1: $N = n$.
The vertex set $V(\hat{G}) = \{v | v$ is an input SE or an output SE$\}$ and edge set $E(\hat{G}) = \{(v, w; i) |$ there is a busy input i of input SE v with $\pi(i)$ being an output of output SE $w\}$.
Case 2: $N > n$.
The vertex set $V(\hat{G}) = \{v | v$ is an input SE or a group output $\}$ and edge set $E(\hat{G}) = \{(v, w; i) |$ there is a busy input i of input SE v with $\pi(i) = w\}$.

It is clear that $\hat{G}$ is a bipartite graph in both cases with all input SEs as one part A and all output SEs for Case 1 or all output groups for Case 2 as another part B. And each edge is corresponding to a pair of input and its mapped output group. There is a one-to-one corresponding relation between every busy input of $G(N,n)$ and every edge of $\hat{G}$. We label each edge by its corresponding busy input. Hence, we can exchange notation of edge and its corresponding input. If an edge is the end edge of a path, we say its labeled input is the end input of the path; if two edges are adjacent, we say their labeled inputs are adjacent; if an edge is colored with some color, we say its labeled input is colored with that color.

A graph $\hat{G}$ is 2-*coloring* if we can color $E(\hat{G})$ by 2 colors so that every vertex with degree d has $\lceil d/2 \rceil$ edges with one color and $\lfloor d/2 \rfloor$ another color. In the following theorem, Theorem 1, we show that $G(N,n)$ with a legal I/G mapping $\pi|K$ has a feasible configuration, which is done by showing $\hat{G}$ is 2-coloring. The proof of theorem 1 not only shows Benes group connector is rearrangeable nonblocking, but also implies a sequential algorithm, which we will implement in parallel in next section, to set up SEs for any legal I/G mapping of Benes group connector.

Theorem 1 *Given any legal I/G mapping $\pi|K$ of a Benes group connector $G(N,n)$, $G(N,n)$ has a feasible configuration.*

Proof. Let $G(H,h)$ be the s-level subnetwork of $G(N,n)$. The proof is done by induction on s. If $s = \log N - 1$, $G(H,h)$ is $G(2,2)$ or $G(2,1)$ which consists of a single node (i.e., a single 2×2 SE) and the claim is obviously true. Assume that the claim is true for any s-level($0 < s \le \log N - 1$) subnetwork $G(H,h)$. For any legal I/G mapping of $G(N,n)$, we know that $G(N,n)$(i.e. 0-level subnetwork) can be represented as a bipartite graph $\hat{G}$. We first prove $\hat{G}$ is 2-coloring. Since $\hat{G}$ is bipartite, $\hat{G}$ does not contain any odd cycle, i.e. any cycle in $\hat{G}$ has even number of edges [3]. So $E(\hat{G})$ is the union of a set of even cycles and paths. Thus we can alternately color each edge with one of two different colors beginning with any busy input along each even cycle or path so that the adjacent edges on the same cycle or path have different colors. We know that every vertex in part A has degree ≤ 2 since each input SE has at most 2 busy inputs, and every vertex in part B has degree ≤ 2 for Case 1 or $\le 2^k$ for Case 2 since each output SE has at most 2 outputs or each output group are mapped by at most 2^k busy inputs respectively. Thus, if a vertex with degree d, then its adjacent $\lceil d/2 \rceil$ edges are colored with one color and $\lfloor d/2 \rfloor$ edges are colored with another color. Therefore $\hat{G}$ is 2-coloring.

Then we show that there is a feasible configuration of Benes Group Connector $G(N,n)$ if its graph model $\hat{G}$ is 2-coloring. We let two ends(i.e. input and its mapped output group) of the edges with the same color connect with the same 1-level subnetwork. Thus every pair of input and its mapped output group is connected with the same 1-level subnetwork, which generates two legal I/G mappings for two 1-level subnetworks of $G(N,n)$. By induction, each of 1-level subnetworks has a feasible configuration. Therefore, $G(N,n)$ has a feasible configuration. ■

We define *mapping constraints* as follows: for any s-level subnetwork $G(H,h)$ of $G(N,n)$ ($0 \le s \le \log N - 2$),
(1) Every busy input of input SEs and its mapped output group are connected with the same $s+1$-level subnetwork.
(2) Two sibling inputs(outputs) are connected with the different $s+1$-level subnetworks.
(3) If $H > h$, the busy inputs of $G(H,h)$ mapped to the same output group are enforced to be partitioned into two parts with each size $\le H/(2h)$, which are connected with the different $s+1$-level subnetworks $G(H/2,h)$; otherwise (i.e. $H = h$), two inputs mapped to two sibling outputs must be connected through its two different $G(H/2, H/2)$ subnetworks.

By theorem 1 and topology of $G(N,n)$, we have the following corollary.

Corollary 1 *Given any legal I/G mapping $\pi|K$, $\mathcal{G}$ is a feasible configuration if and only if $\mathcal{G}$ satisfies the mapping constraints.*

4 Algorithm

For a legal I/G mapping $\pi|K$, any busy input I_i is specified to be connected to a unique output group G_j. We consider the operation of setting up K edge-disjoint paths from busy inputs to their mapped output groups as a routing process, and an algorithm for establishing I/G connections as a routing algorithm. Our routing algorithm is based on the sequential algorithm of [7] and the parallel algorithm of [6, 12] for routing a permutation in Benes network.

To facilitate our discussion, we denote $b_l(x)$ be the l-th significant bit b_l of the binary representation $b_v b_{v-1} \cdots b_l \cdots b_1 b_0$ of integer x, and denote $\bar{x}$ be the integer with binary representation $b_v b_{v-1} \cdots b_l \cdots b_1 (1 - b_0)$.

A $G(N,n)$ consists of $2\log N - 1 - k$ stages. It can be regarded as a concatenation of two parts, *P1* being the first $\log N - 1$

stages, and *P2* being the remaining stages. For each busy input I_i of P2 in stage s of $G(N,n)$, we define its *control bit* to be $b_{2\log N-2-k-s}(\pi(i))$. The control bit is used to do self-routing. If the control bit for a busy upper input is 0(resp. 1), then this busy input is set straight(resp. cross); if the control bit for a busy lower input is 0(resp. 1), then this busy input is set cross(resp. straight).

Lemma 1 *If a Benes group connector has a feasible configuration for a legal I/G mapping $\pi|K$, then all busy inputs of P2 can be set up by control bits.*

Proof. It is clear by figure 3, where $C, x \in \{0,1\}$ and C is the control bit. ■

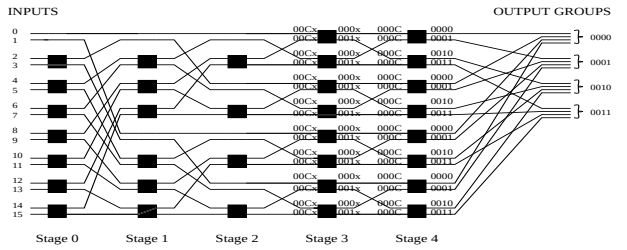

Figure 3: Hardware redundancy of $P1$ and control bit selection of $P2$ in $G(16,4)$.

If we define *P2-passable* condition of $G(N,n)$ as (2) and (3) of the mapping constraints, by corollary 1 and lemma 1, we have the following claim:

Theorem 2 *For any legal I/G mapping $\pi|K$ of $G(N,n)$, if P1 is set up to satisfy P2-passable condition and P2 is set up according to control bits, then the setting of $G(N,n)$ is a feasible configuration.*

Our algorithm, named *ROUTE*, is presented for a parallel computer with N PEs that are completely connected, i.e. the interconnection topology of this computer is a complete graph of N vertices. This algorithm is equivalent to an EREW PRAM algorithm. The N PEs are labeled as $0, 1, \cdots, N-1$, and PE_i and $PE_{\bar{i}}$ are called sibling PEs. Algorithm *ROUTE* consists of three phases: INITIAL_PHASE, PHASE_I and PHASE_II. In INITIAL_PHASE, the total number of busy inputs is computed. In PHASE_I, the setting of all SEs in the first $\log N - 1$ stages are determined. In PHASE_II, self-routing is performed for the SEs in the remaining stages. Conceptually, PHASE_I consists of $\log N - 1$ iterations and each iteration contains 4 steps. In the i-th iteration, $1 \le i \le k$, the setting of busy inputs of stage $i-1$ are determined in the following way: we only consider 2^{i-1} independent $(i-1)$-level subnetworks so that P2-passable condition is satisfied in each subnetwork. In the $(k+1)$-th iteration, we encounter 2^k independent $B(n)$ routing problems. Then, for each such problem, our algorithm degenerates to a parallel routing algorithm based on the sequential algorithm of [7]. PHASE_II is self-routing process. Since P2-passable condition is satisfied after PHASE_I, this guarantees that self-routing for P2 is always possible by theorem 2. P2 is set up by control bits. The basic structure of algorithm is given as follows:

Algorithm *ROUTE*
Input: A legal mapping $\pi|K$ for $G(N,n)$
Output: A feasible configuration of $G(N,n)$
begin
  INITIAL_PHASE: find total number of busy inputs
  PHASE_I: set up busy inputs in stage s $(0 \le s \le \log N - 2)$
    step 1: find dual inputs
    step 2: find representatives
    step 3: set busy inputs
    step 4: assign mapping
  PHASE_II: set up busy inputs in remaining stages by control bits
end

We discuss the idea and implement technique of our algorithm in the following, and omit the details for brevity. Initially, each PE_i is associated with the value of $\pi(i)$.

INITIAL_PHASE: find total number of busy inputs, which can be done by *Parallel Prefix Sums* algorithm in [5]. We mark every PE_i with $\pi(i) \neq -1$ for parallel prefix sums. After INITIAL_PHASE, the total number of busy inputs is stored into the global variable K.

Step 1: find dual input for busy input.
In graph model $\hat{G}$, the degree of each vertex in part B is d with $0 \le d \le 2^l$ where $l = 1$ for Case 1 and $l = k$ for Case 2, which shows d busy inputs are mapped to the sibling outputs for Case 1 and to the same output group for Case 2 respectively. Every set of those d inputs will be paired up as $\lfloor d/2 \rfloor$ pairs of dual inputs, which can be done by parallel integer sorting on K busy inputs. In order to pair two inputs as dual inputs, firstly, we construct K pairs with each one consisting of the labels of one busy input and its mapped output group. Then, we sort those K pairs by the keys being the second values of pairs, which are output groups mapped by busy inputs. In stage s with $s \ge k$, Case 1 of graph model happens, i.e. at most one input is mapped to the same output group. In this case, we sort all pairs by permutation on keys, which takes $O(1)$ time. After permutation, if there are two pairs in a pair of sibling PEs, those inputs in the first values of two pairs are paired up as dual inputs. In stage s with $s < k$, Case 2 of graph model happens, i.e. there are maybe more than one input mapped to the same output group. In this case, we first sort all pairs by parallel integer sorting on the keys [4], which takes $O(\log K)$ time. Each pair $(i, \pi(i))$ is stored in $PE_{r(i)}$ where $r(i)$ is the rank of I_i by sorting. If there are two pairs in a pair of sibling PEs, those inputs in the first values of two pairs are paired up as dual inputs. It is not difficult to see that if there is odd number of busy inputs mapped to some output group, we have finished finding dual inputs for those busy inputs, but not for the case in which an even number of busy inputs is mapped to some output group. Because two inputs, one I_{min} with the smallest label and another I_{max} with the largest label among all inputs mapped to the same output group as I_{min} and I_{max}, might not be paired up due to I_{min} and I_{max} not in a pair of sibling PEs. In order to pair up those busy inputs, we mark every I_i with minimum or maximum label but not both among all inputs mapped to $G_{\pi(i)}$, which is can be simply done by comparing the second value of pair stored in PE_i to see if it is different as exactly one of PE_{i-1}'s and PE_{i+1}'s if $1 \le i \le K-2$, and otherwise(i.e. $i = 0$ or $K-1$), comparison is only done to PE_1 for $i = 0$ and to PE_{K-2} for $i = K-1$. Then we perform parallel prefix sum on marked busy inputs and each marked busy input will get a rank. We permute the marked inputs by their ranks. After permutation, for every pair of sibling PEs, if neither of their stored inputs is paired up and both of their mapped output groups are same, we pair them up as dual inputs. By step 1, we know that two busy inputs are adjacent in a path or cycle if and only if they are sibling inputs or dual inputs. Thus $\hat{G}$ is partitioned into a set of edge-disjointed even cycles and paths

with length no more than $2K$ since each cycle or path visits every vertex of $\hat{G}$ at most once.

Step 2: find the representative for every busy input.
By the proof of Theorem 1, we know graph model $\hat{G}$ is 2-coloring. We choose a representative for the edges with the same color on every cycle or path. Thus, each cycle or path has two different representatives, and each represents the edges with the same color in this cycle or path. For a cycle, we choose each representative be the busy input with smallest label among all inputs with same color in the cycle. It is clear that two representatives of a cycle must be a pair of sibling inputs. For a path, If the end input is the busy input of a semi-busy SE, it is chose as a representative of the path; otherwise the input adjacent to this end input is chose as a representative of the path. Thus, for an even path, if the end inputs are in semi-busy input SEs, we choose two representatives be those two end inputs, and otherwise we choose two inputs adjacent to end inputs. For an odd path with length > 1, only one of two end inputs of the path is in a semi-busy input SE, and this end input is chosen as one representative, called primary representative of the odd path. And another representative, called non-primary representative, is chosen be the input adjacent to another end input in a busy SE. For a path with length 1, two representatives are same, corresponding to the only one input of the path, and this one representative is also called primary representative. The representative of busy input I_i, denoted by $m(i)$, is one of the representatives of the cycle or path in which inputs i and $m(i)$ are so that they are colored with same color. Every busy input will find its representative by pointer jumping[5]. If the sibling of input is idle or the input has no dual input, we know that the busy input is the end input of a path. For every non-end busy input, if the dual input of its sibling input is busy, the pointer is initialized to point to it; otherwise it points to itself. Since the length of a cycle or a path is at most K, for each busy input, $\log K$ times of pointer jumping are enough to get the information what its representative is and if it is in a path or cycle.

Step 3: set all busy inputs by their colors.
Two different representatives of a cycle or a path are colored with two different colors. Since two representatives of a cycle are sibling inputs, we can color every busy input in a cycle by the parity of its representative. In an odd path, the number of edges with the color of the primary representative is one more than the number of edges with color of the non-primary representative. In order to connect no more than half busy inputs with the same subnetwork of next level, we color all representatives first so that the difference of number of primary representatives with different colors is at most 1. Then, we color all busy inputs according to their representatives. Before coloring the representatives, each busy input needs to find another representative of cycle or path in which it is. If neither of the sibling or dual input of a busy input is busy, this busy input is in a path with length $= 1$. In this case, two representatives are same. Otherwise, the busy input can find another representative by its sibling input or dual input. If two representatives are identical or they are in two SEs with different types, i.e. one in a busy SE and another in a semi-busy SE, this busy input knows it is in an odd path and in an even path otherwise. For all busy inputs in odd paths, we mark those inputs with labels same as primary representatives and perform parallel prefix sum to give each of them a unique rank. In order to set the input with minimum label of each subnetworks straight, we check if it is among those marked inputs. If yes, we color all marked inputs with same parity as it's as red, and others as blue. Otherwise, all marked inputs(i.e. primary representatives) with different parity of rank gets different colors. The non-primary representative can be colored with different color as the corresponding primary representative. Similarly, all representatives of even paths can be colored, as we can mark all busy inputs with the smaller one between two representatives and perform parallel prefix sum on them. Finally, color remaining busy inputs as the same colors as their representatives. We set the busy input with even label and red color or that with odd label and blue color as straight, and as cross otherwise. It is easy to see that the difference of number of busy inputs connected to different subnetworks of next level is at most 1. Thus the number of busy inputs connected to the same subnetwork of next level is reduced by half after each iteration.
Step 4: reassign mapping $\pi(i)$ for each input i in next stage.
After finishing setting of all inputs in the stage s, the input i of the stage s is connected to one of inputs $\{j, j + N/2^{s+1}\}$ ($j = (i \bmod N/2^s)/2 + N/2^s \cdot \lfloor i/(N/2^s) \rfloor$) of stage $s+1$ according to the setting of input i, i.e. if i is even and is set straight or i is odd and is set cross, input i of stage s is connected to input j of stage $s+1$, otherwise input i of stage s is connected input $j + N/2^{s+1}$ of stage $s+1$. Thus each PE_i gets the values of $\pi(i)$ for setting of busy input i of next stage in the next iteration.

Our algorithm in PHASE_II is presented in such a way that all PEs participate routing process for P2 of $G(N, n)$. Actually, once the setting of SEs in P1 is determined, cells can be injected into $G(N, n)$. When the cells reach P2, the SEs can determine its setting by inspecting the control bits of SEs.

We need $O(\log N)$ time to find the number of busy inputs by performing prefix sums. Since the length of cycle or path is at most K, we need $O(\log K)$ time to perform pointer jumping, prefix sums and sorting in first $\log K$ iterations. Because the number of busy inputs connecting to the same subnetwork of next level is reduced by half after each iteration, each iteration in PHASE_I only takes $O(1)$ time after $O(\log K)$ iterations. Thus, the total time for PHASE_I is $O(\log^2 K + \log N)$. There are $2 \log N - k$ iterations in PHASE_II, each takes $O(1)$ time. Therefore, the total time complexity of algorithm *ROUTE* is $O(\log^2 K + \log N)$.

In summary, we have the following claim:

Theorem 3 *For any legal I/G mapping $\pi|K$, algorithm ROUTE correctly computes a feasible configuration of $G(N, n)$ in $O(\log^2 K + \log N)$ time on a parallel completely connected computer or the EREW PRAM model using N PEs.*

Example 1 *Figure 4 and the following discussion(in this example, if some value of an input does not exist, it is set as -1) show main idea of parallel algorithm ROUTE to set up busy inputs in the first stage of $G(16, 4)$ for a legal I/G mapping*

$$\begin{pmatrix} i: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \pi(i): & 2 & -1 & -1 & 1 & 2 & 0 & 0 & 3 & 3 & 0 & 1 & 2 & 3 & 0 & -1 & 2 \end{pmatrix}$$

By INITIAL_PHASE, the total number of busy inputs is computed, which is equal to 13. In step 1 of PHASE_1, every busy input gets its rank $r(i)$ by parallel integer sorting as follows:

$$\begin{pmatrix} i: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ r(i): & 6 & -1 & -1 & 4 & 7 & 0 & 1 & 10 & 11 & 2 & 5 & 8 & 12 & 3 & -1 & 9 \end{pmatrix}$$

And, two busy inputs stored in a pair of sibling PEs are paired as dual inputs:

$$\begin{pmatrix} PE_i: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ input: & 5 & 6 & 9 & 13 & 3 & 10 & 0 & 4 & 11 & 15 & 7 & 8 & 12 \\ dual\ input: & 6 & 5 & 13 & 9 & 10 & 3 & 4 & 0 & 15 & 11 & 8 & 7 & -1 \end{pmatrix}$$

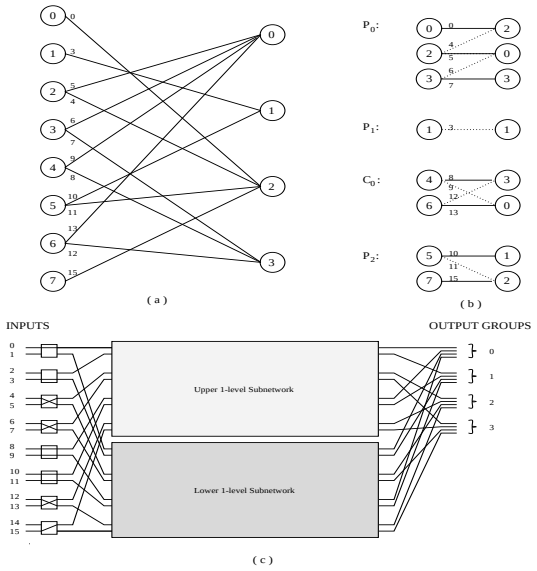

Figure 4: Setting up SEs by parallel algorithm *ROUTE*: (a) Graph model $\hat{G}$ for a legal mapping of $G(16,4)$; (b) 2-coloring of paths and cycle(each edge is labeled by its input number); (c) setting up SEs in the first stage of $G(16,4)$.

In the step 2, each input i finds its representative $m(i)$ by pointer jumping:

$$\begin{pmatrix} i: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ m(i): & 0 & -1 & -1 & 3 & 6 & 0 & 6 & 0 & 8 & 9 & 15 & 11 & 9 & 8 & -1 & 15 \end{pmatrix}$$

In step 3, every path and cycle finds its two representatives $m(i)$ and $m'(i)$. So P_0, P_1, C_0 and P_2 find pairs of $(m(i), m'(i))$ as $(0,6)$, $(3,3)$, $(8,9)$ and $(15,11)$ respectively. Color representatives of cycle C_0, inputs 8 and 9, by its parity as red and blue. The primary representatives of the odd paths P_0, P_1 and P_2, inputs 0, 3 and 15, are alternately colored with red, blue and red. Thus all busy inputs of input SEs in $G(16,4)$ can be colored as their representatives in figure 4 (b) and set up as figure 4 (c).

In the step 4, two new mappings are given to inputs of input SEs in two subnetworks of next level as follows:

$$\begin{pmatrix} i: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \pi(i): & 2 & -1 & 0 & 3 & 3 & 1 & 0 & 2 \end{pmatrix}$$

and

$$\begin{pmatrix} i: & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \pi(i): & -1 & 1 & 2 & 0 & 0 & 2 & 3 & -1 \end{pmatrix}$$

5 Generalizations and Concluding Remarks

The parallel machine models we used are not realistic. However, our algorithm can be converted to fit any realistic machine model. A parallel operation involving interprocessor communication can be achieved by sorting. Let $\hat{S}(N)$ be the time for sorting N elements on a parallel machine M with N processors. Then, as the algorithm for routing in Benes network $B(N)$ of [12], our algorithm can be implemented on a machine with N PEs in no more than $O(\log^2 K \cdot \hat{S}(N) + \log N \cdot \hat{S}(N))$ running time where there are $O(\log K)$ busy inputs. For example, when implemented on parallel computers whose PEs are connected by perfect shuffle and hypercube networks, our algorithm takes $O(\log^4 K + \log^2 K \cdot \log N)$ time.

Also, it is not difficult to see that the proposed parallel algorithm for $G(N,n)$ always set the SEs indexed by $\{0, \cdots, (N/2^{s+1}) \times j\}$, where $0 \le s \le \log N - 2$ and $j \in \{0, \cdots, 2^s - 1\}$, in the s-th stage to the straight state. Thus, these SEs can be eliminated. So the hardware redundancy in first $\log N - 1$ stages is $\sum_{i=0}^{\log N - 2} 2^i = 2^{m-1} - 1 = N/2 - 1$. Translated into crossing points, the number of saved crossing points in Benes group connector $G(N, N/2^k)$ is increased to $2N \cdot k + 2N - 4$, compared with the $N \times N$ Benes network. For Example 1, we can reduce $1 + 2 + 4 = 7$ SEs in the first 3 stages. Comparing with the Figure 2(c), the Benes group connector in Figure 3 has much lower hardware cost. Thus, when $k = 0$ in which case $G(N,n)$ is the $N \times N$ Benes network $B(N)$, the hardware redundancy achieved by our algorithm is the same as the number given in [15].

To our knowledge, all known algorithms for setting up Benes networks $B(N)$ cannot be directly applied to set up Benes group connectors. However, our algorithm for $G(N,n)$ can be directly applied to set up Benes network $B(N)$ by selecting $n = N$.

References

[1] V.E. Benes, On rearrangeable three-stage connecting networks, *Bell System Technical Journal*, vol.41, no.5, pp.1481-1492, September 1962.

[2] V.E. Benes, *Mathematical theory of connecting networks and telephone traffic*, Academic Press, 1965.

[3] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, Elsevier North-Holland, 1976.

[4] R. Cole, Parallel Merge Sort, *SIAM Journal on Computing*, vol.17, no.4, pp.770-785, Aug. 1988.

[5] J. Jaja, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[6] Ching-Yi Lee, A. Yavuz Oruc, A fast parallel algorithm for routing unicast assignments in Benes networks, *IEEE Trans. on Paral. and Distr. Systems*, vol.6, no.3, pp.329-334, March 1995.

[7] K.Y. Lee, A New Benes Network Control Algorithm, *IEEE Trans. Computers*, vol.36, no.6, pp.768-772, June 1987.

[8] K.Y. Lee, On the rearrangeability of a ($2 \log N - 1$) stage permutation network, *IEEE Trans. Computers*, vol.34, no.5, pp.412-425, May 1985.

[9] T.T. Lee and S.Y. Liew, Parallel routing algorithms in Benes-Clos networks, *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation., Proceedings IEEE*, vol.1, pp.279-286, March 1996.

[10] F.T. Leighton, *Introduction to parallel algorithms and architectures: Arrays.Trees.Hypercubers*, M. Kaufmann Publishers, 1992.

[11] G.F. Lev, N. Pippenger and L.G. Valiant, A fast parallel algorithm for routing in permutation networks, *IEEE Trans. Comput.* , vol.30, pp.93-100, Feb. 1981.

[12] N. Nassimi and S. Sahni, Parallel Algorithms to Set up the Benes Permutation Network, *IEEE Trans. Computers*, vol.31, no.2, pp.148-154, Feb. 1982.

[13] D.C. Opferman and N.T. Tsao-Wu, On a Class of Rearrangeable Switching Networks, Part I: Control Algorithm, *Bell System Technical J.*, vol.50, pp.1,579-1,600, 1971.

[14] J.H. Patel, Processor-memory Interconnections for Multiprocessors, *Proc. 6th Ann. Symp. Comput. Architecture*, pp.168-177, Apr. 1979.

[15] A. Waksman, A permutation Network, *J. ACM*, vol.15, no.1, pp.159-163, Jan. 1968.

[16] M. Yang and S.Q. Zheng, The kDDR scheduling algorithms for multi-server packet switches, to appear in *Proc. of the ISCA 15th International Conference on Parallel and Distributed Computing Systems*, 2002.

[17] M. Yang and S.Q. Zheng, Efficient scheduling for CIOQ switches with space-division multiplexing and grouped inputs/outputs, submitted for publication.

[18] Y. Yang, S.Q. Zheng, D. Verchere, Group switching for DWDM networks, submitted for publication.

[19] S.Q. Zheng and Y. Xiong, Ingress edge router architecture and related channel scheduling algorithms for OBS networks, Alcatel internal technical report, 2000.

[20] S.Q. Zheng, M. Yang, and F. Masetti, Hardware scheduling in high-speed, high-capacity IP routers, in this proceeding.