Performing Permutations on Interconnection Networks by Regularly Changing Switch States

Wei Kuang Lai, Member, IEEE

Abstract—In this paper, we present an algorithm for performing permutations of messages on multistage interconnection networks. Permutations of messages are needed in many parallel algorithms. The proposed algorithm is feasible for any networks that can connect each input to each output using a set of $N$ nonblocking connections, where $N$ is the number of ports on the network. Messages are segmented into $N$ submessages that are sent independently in each time step. For any permutation, the settings of switches are changed with fixed patterns. Partitioning of the network into independent subnetworks is also supported, each capable of simultaneously routing a different permutation.

Index Terms—Permutation, multistage interconnection networks, deterministic.

1 INTRODUCTION

Rapid communications among processors are important for parallel computers. Permutations are frequently used communications. For example, data are permuted among processors for the next step of parallel algorithm or data are scrambled before they are stored in the parallel memory [35]. A permutation can be viewed as a one-to-one and onto mapping of sources to desired destinations. In the past, researchers have worked on the control aspect of different networks. Most of the early works are on the telephone switching system [2], [3], [21]. Then researchers begin to focus on the Beneš network and interconnection networks. Waksman has shown how the Beneš network can be set to realize any permutation with an algorithm complexity of $O(N \log N)$ [34]. Lee proposes a nonrecursive algorithm to set up switching nodes for the Beneš network [22]. The algorithm sets switches one stage at a time, stage by stage.

Many kinds of interconnection networks have been investigated [24], [25]. Previous investigators have found that by performing multiple passes through some networks, any permutation can be realized. A two-pass algorithm ($2 \log_2 N$ steps) for the reverse-exchange interconnection network is proposed in [8]. The SW-Banyan network is shown to be able to simulate the Beneš network in two passes and thus can perform any permutation [11]. Parker indicates that three passes are sufficient to generate any permutation for the inverse Omega network, the indirect binary $n$-cube, and the R-network (derived from three-stage networks) [2], [3] [1]. Stone shows how to realize any permutation using $O(\sqrt{N \log_2 N})$ perfect-shuffle steps [4]. Opferman and Tsao-Wu also focus on the control aspect of a class of networks [23]. Many single-stage networks are also studied. Moreover, Wu and Feng show that $3 \log_2 N - 1$ passes through a single-stage shuffle network are sufficient to realize any permutation [9]. Huang and Tripathi show that $2 \log_2 N - 1$ passes of shuffle-exchange are necessary and $3 \log_2 N - 1$ passes of shuffle-exchange are sufficient to realize any permutation in a single-stage shuffle network [10]. This paper will focus on multistage interconnection networks (MINs).

Due to the time complexity in set-up time, researchers also consider many modifications to improve the speed of set-up time. For example, a fully interconnected computer can be used to improve the set-up time to $O(\log^2 N)$ [26]. The extra cost is that the scheme needs $N$ processors with $O(N^2)$ connections. Cellular permutation arrays can reduce the set-up time to $O(N)$ [27]. The hardware complexity of switches is $O(N^2)$.

However, modifications to MINs will increase hardware cost and will not be considered in this paper. For all previous researches, the set-up time is $O(N \log N)$ in general for MINs without modifications. In addition, the control algorithm has to be recalculated each time when the performed permutation is changed. Owing to the control algorithm’s time complexity, Lang and Stone propose a relatively simple algorithm to realize some important permutations [5]. Lenfant also proposes a control algorithm for the Beneš network to realize frequently used permutations [32].

In this paper, we will design a simple control algorithm to realize any permutation in two passes. The Generalized Cube network [12] is used to show how to perform any permutation in two passes. From Agrawal’s paper, we know that all MINs satisfying a characteristic called the “buddy property” are topologically equivalent [28]. Hence, the Generalized Cube can be used as a representative of many networks in that they are topologically equivalent to it [12], [18], [8], which means the algorithm can be used in any networks that are capable of distributing $N$ messages of each node to $N$ nodes in $N$ time steps. Multistage interconnection networks, such as the baseline [8], delta [13], Generalized Cube [12], indirect binary $n$-cube [14], omega [15], STARAN flip [16], and SW-Banyan ($S = F = 2$)
are all capable of distributing messages to all nodes in $N$ time steps.

The time complexity of our algorithm is $O(N)$. The proposed algorithm regularly changes states of switching nodes which are suitable for hybrid electro-optical MINs with optical interconnects [36]. Moreover, the switching patterns for those switching nodes are fixed so the switching nodes do not need to have any intelligence and the switching patterns do not need to be recalculated when different permutations are performed. The algorithm bears some resemblance to a routing algorithm proposed by Valliant for hypercube that begins with a randomization step followed by deterministic routing [6], [7]. However, it differs from Valliant’s algorithm in three ways: First, messages are not segmented in Valliant’s algorithm. The messages are distributed randomly in the first pass. Then in the second pass, all messages are routed to their destinations. Although in most cases, the permutation can be finished within a fixed time, conflicts can happen in the first pass and the second pass and the actual finished time is not time deterministic. On the other hand, in our algorithm, a message is divided into $N$ submessages, where $N$ is the total number of input ports in a MIN. Each submessage has a fixed size. In the following, we will call a message with length equal to $N$ submessages a complete message. A large data is segmented into several complete messages and those messages with length less than $N$ submessages are padded to form complete messages. This is a typical strategy in packet switching networks and circuit switching mode of telephone networks. For example, a message can be a packet and a submessage can be a cell in an ATM network [29]. Our algorithm also differs with Valliant’s algorithm on the finished time. In the first pass, our algorithm segments each complete message into $N$ submessages and distributes them to $N$ different nodes. Then in the second pass, submessages are routed to their destinations. For both passes, switches in each stage are set up according to a fixed pattern which will not cause any communication conflicts. Hence, both passes are time deterministic. In addition, two algorithms are applied for different networks. The Valliant’s algorithm is for hypercube networks while ours is for interconnection networks. The idea of regularly changing switch states also bears some resemblance to Qiao and Melhem’s algorithm. However, it differs from their scheme in three ways. First, their scheme is for emulating other networks such as rings, meshes, hypercubes, etc, while our scheme is for performing permutations. Secondarily, messages are not segmented in their scheme and are segmented in our scheme. Moreover, their scheme needs to recalculate and change switching patterns for emulating different networks. In our scheme, the same switching patterns can be applied to different permutations.

The comparisons of our algorithm, the distributing algorithm, and other routing algorithms for permutations are listed in Table 1. Note that for $q$ different permutations, the distributing algorithm only needs $q + 1$ passes. It will be explained further in Section 3. The rest of the paper is organized as follows: Section 2 is preliminaries for the proposed algorithm. The permutation algorithm and related issues are studied in Section 3. Concluding remarks are in Section 4.

### 2 Preliminaries

**Definition 1.** Each switch is a two-input/two-output device and is individually controlled. A switch can be set to one of two states. Allow the upper input and output lines be labeled $i$ and the lower input and output lines be labeled $j$. The two states are 1) straight—input $i$ to output $i$, input $j$ to output $j$; (2) exchange—input $i$ to output $j$, input $j$ to output $i$ [15].

For the permutation algorithm developed here, only the straight setting and the exchange setting are used (Fig. 1). This study uses 0 to represent the straight and 1 to represent the exchange in a switch.

![Straight and Exchange States](image)
**Definition 2.** The Generalized Cube network has $N$ input ports and $N$ output ports, where $N = 2^n$. The network ports are numbered from 0 to $N - 1$. Input and output ports are network interfaces to external devices called sources and destinations, respectively, which have addresses corresponding to their port numbers. The Generalized Cube topology has $n$ stages and the stages are numbered from stage $n - 1$ to stage 0, where each stage has $N/2$ switches. The connections in the Generalized Cube are based on the cube interconnection functions [12], [19]. Let $P = p_{n-1}...p_0$ be the binary representation of an arbitrary I/O line label. The $n$ cube interconnection functions can be defined as follows:

$$\text{cube}_i(p_{n-1} ... p_0) = p_{n-1} ... p_i ... p_0,$$

where $0 \leq i < n$, $0 \leq P < N$ and $p_i$ denotes the complement of $p_i$.

This means that the cube interconnection function connects $P$ to $\text{cube}_i(P)$, where $\text{cube}_i(P)$ is the I/O line whose label differs from $P$ in just the $i$th bit position. Stage $i$ of the Generalized Cube topology contains the $\text{cube}_i$ interconnection function, i.e., it pairs I/O lines that differ in the $i$th bit position. It is the only stage which can map a source to a destination with an address different from the source in the $i$th bit position. An example of the Generalized Cube with eight input ports and eight output ports is shown in Fig. 2.

**Definition 3.** The interconnection network can be described as a set of interconnection functions, where each is a permutation (bijection) on the set of port numbers or source/destination addresses [19]. When an interconnection function $f$ is applied, input $S$ is connected to output $f(S) = D$ for all $0 \leq S < N$, simultaneously. That is, stating that the interconnection function maps the source address $S$ to the destination address $D$ is equivalent to stating that the interconnection function causes data sent on the input port with address $S$ to be routed to the output port with address $D$.

MINs can be built by either packet switched MINs or circuit switched MINs. Because various techniques have made packet switched MINs more prevalent [30], [31], this paper will focus on packet switched MINs although the algorithm can also be applied to circuit switched MINs [33]. Some of the issues are considered here.

1. The packets are normally of fixed size [30]. The time for sending a packet from one stage to another stage is time deterministic.
2. There are finite buffers in the input buffers of each switch.

**Definition 4.** Assume a complete message is divided into $N$ submessages of the same size. A time step refers to the time needed to send one submessage in the input buffer of a switch to the input buffer of a switch in the next stage, including the time to change the switch states, if necessary. It means that within a time step, one submessage in each input is moved to an input in the next stage.

A central clock is used to synchronize the settings of switches for each time step. Each switch has a finite number of buffers with it. Fig. 3 is a model of the Generalized Cube network with a central clock, eight input ports, and eight output ports.

**Definition 5.** A switching pattern $S_{ji} = x_1, x_2, ..., x_j$ defines the sequence for setting a switch in stage $i$ in performing permutation, where $0 \leq i \leq n - 1, 1 < j \leq N, x_k = 1$ or 0, and $1 \leq k \leq j$. If $x_k = 0$, the switch is set straight in time step $k$. If $x_k = 1$, the switch is set exchange in time step $k$. A permutation can be realized by applying switching patterns to switches of MINs.

**Definition 6.** A routing tag $T = x_{n-1}, x_{n-2}, ..., x_0$ defines the path of a submessage in a MIN, where $x_l = 0$ or $1, 0 \leq l \leq n - 1$. If $x_l = 0$, it means when the submessage passes through a switch in stage $l$, the switch is set straight. If $x_l = 1$, it means when the submessage passes through a switch in stage $l$, the switch is set exchange.

![Fig. 2. A Generalized Cube network with eight input ports and output ports ($N = 8$).](image-url)
Definition 7. The translation of a node $S$ by $T; T_r S; T_r^*$ implies the bit-wise exclusive-OR, $\oplus$, of all its address $S$ by $T$. For example, if $S = 000111_2$ and $D = 101011_2$, then $S \oplus D = 101002$.

Definition 8. Let $S = s_{n-1}s_{n-2}\ldots s_1s_0$ be the binary representation of a source address. A cyclic shift function $s_i$ is an interconnection function and is defined as follows:

$$sh_i(S) = s_{n-i-1}s_{n-i-2}\ldots s_{1}s_{0}s_{n-i-1} \ldots s_{n-1}s_n,$$

where $0 \leq i \leq n - 1$.

A shift family is a subset of interconnection functions which include $sh_0, sh_1, sh_2, \ldots, sh_{n-2},$ and $sh_{n-1}$.

3 THE PERMUTATION METHOD

Here, we assume that the buffer size in each switch is equal to a submessage for simplicity. Also assume that nodes are connected to a Generalized Cube network with $N$ input ports and $N$ output ports. Each node is connected to one input port and one output port. There are $N$ sending buffers and $N$ receiving buffers in each node. The $N$ sending buffers are numbered from 1 to $N$ and are denoted as the first buffer, the second buffer, $\ldots$, and the $N$th buffer. For simplicity, the first sending buffer of each node is assumed to be the input buffer of the switch that the node is connected to. The receiving buffers are also numbered and denoted in a similar way. The submessage in the $i$th buffer is called the $i$th submessage. When a complete message is sent, the first submessage is sent out first, then the second submessage, $\ldots$, and the $N$th submessage. In the following section, we discuss the distributing algorithm for realizing any permutation and related issues.

3.1 The Distributing Algorithm

The Distributing Algorithm

1. Every node segments its complete message into $N$ submessages, each with a destination address and a sequence number. The destination address will be needed in the second pass of a permutation and the sequence number will be needed in the reassembling of the complete message.

2. In the first time step, a submessage in each node is sent out from the input buffer of a switch in stage $n - 1$ to the input buffer of a switch in stage $n - 2$. In each time step of the following $N - 1$ time steps, a submessage in each node is sent out from the input buffer of a switch in stage $n - 1$ to the input buffer of a switch in stage $n - 2$ and those submessages in intermediate stages are forwarded one stage until they arrive at the intermediate nodes. For a total of $N + n - 1$ time steps, all submessages will arrive at the intermediate nodes because after $N$ time steps, the $N$th submessages are in the input buffers of switches in stage $n - 2$ and additional $n - 1$ time steps are needed to move the $N$th submessages to the intermediate nodes. The settings of switches are changed at the beginning of each time step. Switches in different stages of the Generalized Cube network follow different switching patterns. Switches in the same stage have the same switching pattern. There are many possible ways in designing switching patterns. The aim is to distribute each complete message into $N$ different nodes with one submessage in each node in the first pass and to route those distributed submessages, $N^2$ in total, to their destinations in the second pass. One possible design of the switching pattern will be described later.

3. After the end of the first pass, each node receives $N$ submessages which originate from $N$ different nodes and have $N$ different destinations. For every
submessage received in each node, the node places this submessage in an appropriate buffer space. Note that submessages placed in different buffer spaces have different routing tags and will arrive at different destination addresses. Hence, we can place each in an appropriate buffer according to its final destination, one buffer space for one submessage. The way to place each submessage in the appropriate buffer will be described later in this section after the explanation of the switching patterns and the routing tag.

4. Send out submessages received in the first pass to a Generalized cube network again. In here, the receiving buffers in each node now play the role of sending buffers. The switching patterns can be set to be the same as we use in Step 2. After the end of the second pass, the \( N \) received submessages in each node can be reassembled according to their sequence numbers.

For the Generalized cube network with one stage, the switching pattern \( S_{p0} \) is 012. The following is one possible way for designing switching patterns with more than one stage. One advantage of the switching patterns is that we only need to change switch states every two time steps. Hence, power consumption can be saved and the changing speed of switching states can be reduced by a half.

a) For switches in stage \( i \), \( 1 \leq i \leq n - 1 \), the switching pattern \( S_{pi} \) is

\[
0_{1}0_{2} \ldots 0_{2i-1}1_{2i-1} \ldots 1_{2i-1}1_{2i+1} \ldots 1_{2n-1}1_{2n}.
\]

The switching pattern \( S_{p0} \) is repeated in each stage until the end of permutations.

b) For switches in stage 0, the switching pattern \( S_{p0} \) is 00112. That is, the switching patterns for the stage 0 is the same as the switching patterns for stage 1. The switching pattern \( S_{p0} \) is also repeated until the end of permutations.

Table 2 displays the switching patterns for a two-stage Generalized Cube MIN. Both the switching pattern for switches in stage 0 and the switching pattern for switches in stage 1 are 00112. Table 3 displays the switching patterns for a three-stage Generalized Cube MIN. The switching pattern for switches in stage 2 is 000011112. The switching patterns are repeated until the end of permutations.

**Observation 1.** For the switching pattern stated above, a Generalized Cube Network can distribute \( N \) submessages in each node to \( N \) different nodes. For those submessages in the same buffer spaces of \( N \) different nodes, they will be distributed to \( N \) different destination nodes.

**Explanation.** It is easy to show that it is true for a Generalized Cube network with one stage. For a Generalized Cube network with two stages, Fig. 4 is an example of permutation. In the Fig. 4, the buffers within the switches are drawn in front of the switches for easy explanation. Fig. 5 is the first pass of the permutation. For the submessages in the first buffer spaces, that is \( A, E, I \), and \( M \), they pass through stage 1 when stage 1 is set straight in the first time step. Then they pass through

---

**TABLE 2**

Switching Patterns for a Two-Stage Generalized Cube Network with Four Nodes

<table>
<thead>
<tr>
<th>Time step</th>
<th>Switching Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

**TABLE 3**

Switching Patterns for a Three-Stage Generalized Cube Network with Eight Nodes

<table>
<thead>
<tr>
<th>Time step</th>
<th>Switching Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>
stage 0 when stage 0 is set straight in the second time step. Hence, the submessages have a routing tag 002. For the submessages in the second buffer spaces, that is B, F, J, and N, they pass through stage 1 when stage 1 is set straight in the second time step. Then, they pass through stage 0 when stage 0 is set exchange in the third time step. Thus, the routing tag for them is 012. Routing tags for submessages in the third buffer spaces and the fourth buffer spaces can be derived in a similar way.

Table 4 shows routing tags for different time steps. Hence, the submessages stored in different buffer spaces have different routing tags and are distributed to different nodes. Also, because submessages in the same buffer spaces of the four nodes have the same routing tag and different source addresses, they will be distributed to four different destination nodes. Note that the destination of each submessage can be obtained by performing the translation operation on the source address $S$ and the routing tag $T$.

We assume that this is true for a Generalized Cube network with $n - 1$ stages. We will show this is also true for a Generalized Cube network with $n$ stages. By setting up the switching patterns according to the way stated above, the switching pattern for stage $n - 1$ is 000...0111...1 with $N=2^N$. When a submessage passes through the $n$ stage Generalized Cube, it first passes stage $n - 1$ in the first time step and then passes the switches between stage $n - 2$ and stage 0 in the following $n - 1$ time steps. So, for submessages stored in the buffer spaces numbered from 0 to $N/2 - 1$, the switches in stage $n - 1$ are set straight when the submessages pass through them, and for submessages in the buffer spaces numbered from $N/2$ to $N$, the switches in stage $n - 1$ are set exchange when the submessages pass through them. We know that the Generalized Cube network can be partitioned into independent subnetworks by setting all the switches in a particular stage to the straight setting [20]. It means the submessages in the first $N/2$ buffers will be distributed to nodes from node 0 to node $N/2 - 1$, and the submessages in the second $N/2$ buffers will be distributed to nodes from node $N/2$ to node $N - 1$. Also, because there are $n - 1$ stages between stage $n - 2$ and stage 0, a Generalized Cube network with $n - 1$ stages is capable of distributing $N/2$ submessages in each node to $N/2$ different nodes. It is also capable of distributing $N/2$ submessages in the same buffer spaces of $N/2$ different nodes to $N/2$ different nodes. From the above explanation, we know that all $N$ submessages in each node will be distributed to $N$ different nodes. We also know that the $N$ submessages in the same buffer spaces of $N$ different nodes will be distributed to $N$ different nodes.

**Table 4**

Routing Tags for Submessages Beginning in Different Time Steps on a Two-Stage Generalized Cube Network

<table>
<thead>
<tr>
<th>Time step</th>
<th>Routing Tag T</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>9</td>
<td>00</td>
<td>S ⊕ T</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>S ⊕ T</td>
</tr>
</tbody>
</table>
We can use Table 4 to decide the destinations of submessages in different buffers for a two-stage Generalized Cube MIN. In general, for an \( n \)-stage Generalized Cube MIN, the routing tag \( T \) for the submessages in the buffer place \( i \) generally consists of the setting of switches in stage \( n - 1 \) in time step \( i \), the setting of switches in stage \( n - 2 \) in time step \( (i + 1) \mod N \), and the setting of switches in the stage 0 in time step \( (i + n - 1) \mod N \). Hence, we can build a table similar to Table 4. The destinations for submessages in different buffer spaces can also be calculated before the beginning of the transmission by a translation operation, \( Tr(S,T) \), where \( 0 \leq S \leq N - 1 \) and \( T \) is the routing tag.

Now we explain how to place the submessages in an appropriate buffer space in the first pass. This can be done by first calculating the routing tag needed for each submessage to arrive at its destination. The routing tag is obtained by performing the translation operation on the current address and the destination address. For the example in Fig. 5, A is in node 002 and its destination is 012; therefore, the needed routing tag is \( 002 \oplus 012 = 012 \). After the first pass, five time steps have passed. From Table 4, we know the submessages beginning at the sixth time step have a routing tag \( 012 \), so A is stored in the first buffer space of the node 002. Other submessages are also placed in appropriate buffer spaces in the same way. Hence, \( A, K, G, \) and \( M \) are stored in the first buffer spaces of each node. \( F, P, D, \) and \( J \) are inserted in the second buffer spaces of each node. \( O, E, I, \) and \( C \) are placed in the third buffer spaces of each node. And \( L, B, N, \) and \( H \) are put in the fourth buffer spaces of each node. For an \( n \)-stage Generalized Cube MIN, each pass takes \( (N + \log_2 N - 1) \) time steps. We can also decide the buffer space to store each submessage using a similar way. Fig. 6 is the second pass of the permutation for the example in Fig. 4. After the second pass, each complete message arrives at its destination.

Observation 2. The distributing algorithm can finish a permutation within \( 2(N + \log_2 N - 1) \) time steps.

Explanation. To send a complete message through a Generalized Cube network with \( n \) stages, the total time is equal to \( N + \log_2 N - 1 \) time steps, \( N \) time steps for sending out \( N \) submessages and \( \log_2 N - 1 \) time steps for the last submessage to pass additional \( \log_2 N - 1 \) stages. Each permutation takes two passes to finish. Thus, the total time steps are as stated.

The distributing algorithm can also be explained in another way. Initially, node 0 has message \( m_0 \), node one has message \( m_1, \ldots, \) and node \( N - 1 \) has message \( m_{N-1} \). Each message \( m_i, i \leq 0 \leq N - 1 \), is split into \( N \) submessages, \( m_0, m_1, \ldots, m_{N(N-1)} \). Those \( N^2 \) submessages can be viewed as an \( N \times N \) matrix \( M \). Each node stores one row of the matrix. Different rows of the matrix are stored in different nodes. After the first pass, the matrix \( M \) is transposed. Then, the relative positions in each row, within the same node, are changed. Because the \( N \) submessages in each row originate from \( N \) different sources and have \( N \) different destinations, their final destinations are all in different rows. Thus, by appropriately arranging the relative positions in each row, the second pass can send each submessage to its final destination by transposing the arranged matrix again. Although the Generalized Cube Network is used here as the example, as long as a network has the property of distributing \( N \) submessages of each node to \( N \) nodes, the network can use the proposed...
algorithm to perform any permutation. In fact, the Generalized Cube Network is topologically equivalent to baseline [8], delta [13], indirect binary n-cube [14], omega [15], STARAN flip [16], SW-Banyan (S = F = 2) [17], etc. Therefore, switching patterns can also be found for those networks. For example, the shift family can also be used in distributing N submessages of each node to N nodes. The Generalized Cube Network [21], the indirect binary n-cube [14], the STARAN flip [16], and many other MINs have been demonstrated by researchers to be able to support the shift family. The routing tags of those MINs to support the shift family have also been derived. Therefore, their switching patterns for supporting the shift family can be found easily.

3.2 Related Issues

Two unidirectional, cascaded Generalized Cube networks can form a permutation pipeline (see Fig. 7). If more than one permutation is performed, the second permutation can be initiated as soon as the last submessage of the first permutation enters the first stage of the Generalized Cube in the first pass. Therefore, only q + 1 passes are required to complete q different permutations. For a bidirectional Generalized Cube network, a single permutation can be performed using a forward pass and then a reverse pass through the network. Furthermore, two permutations can be performed simultaneously on a bidirectional network with the two permutations making two counterflowing passes (see Fig. 8). Multiple messages can be transferred in parallel in a bidirectional Generalized Cube network; otherwise, they can be pipelined if two unidirectional networks are cascaded.

The Generalized Cube network can be partitioned into independent subnetworks by setting all the switches in a particular stage to the straight setting [20]. This prevents the cubei function for some i from being performed. Thus, each source/destination pair of addresses must correspond in bit position i, thereby isolating or partitioning the processors based on the values of their ith address bits. More than one stage’s switches may all be set straight, further subdividing the processors into independent groups. Partitioning can be enforced without affecting the ability to perform any permutation among the processors of a single partition in two passes. Partitioning does not need to be performed as a set of equal size partitions, but may include partitions of various sizes with powers of two.

4 Conclusions

In this study, we have presented a distributing algorithm for performing all permutations in two passes through a Generalized Cube MIN. By extension, the proposed algorithm is feasible for any network topologically equivalent to the Generalized Cube and for fault-tolerant extensions of the Generalized Cube topology. The complexity of the algorithm, O(N), is due to the settings of switches. Since the time for changing the states of a switching node can be very short compared to the time needed for transmitting a submessage, the overhead for setting switches is actually very small. In addition, the scheme is also suitable for hybrid electrooptical MINs.

Acknowledgments

The author would like to thank the National Science Council of the Republic of China for partial support of this manuscript under contract no. NSC-86-2213-E-110-022.

References

LAI: PERFORMING PERMUTATIONS ON INTERCONNECTION NETWORKS BY REGULARLY CHANGING SWITCH STATES


Wei Kuang Lai received the BS degree in electrical engineering from National Taiwan University in 1984 and the PhD degree in electrical engineering from Purdue University in 1992. He joined the faculty of the Department of Computer Science and Engineering, National Sun Yat-Sen University in August 1992 and is now a professor. His research interests are in high-speed networks and wireless networks.