Reconfiguration with Time Division Multiplexed MIN's for Multiprocessor Communications

Chunming Qiao and Rami Melhem

Abstract—**In this paper, time division multiplexed multistage interconnection networks (TDM–MIN's) are proposed for multiprocessor communications. Connections required by an application are partitioned into a number of subsets, called mappings, such that connections in each mapping can be established in an MIN without conflict. Switch settings for establishing connections in each mapping are determined and stored in shift registers. By repeatedly changing switch settings, connections in each mapping are established for a time slot in a round-robin fashion. Thus, all connections required by an application may be established in an MIN in a time division multiplexed way.**

TDM–MIN's can emulate a completely connected network using N time slots. It can also emulate regular networks such as rings, meshes, cube-connected-cycles (CCC), binary trees, and n-dimensional hypercubes using 2, 4, 3, 4, and n time slots, respectively. The problem of partitioning an arbitrary set of requests into a minimal number of mappings is NP-hard. Simple heuristic algorithms are presented and their performances are shown to be close to optimal. The flexibility of TDM–MIN's allows for the support of run-time requests through dynamic reconfigurations. The techniques are especially suitable for hybrid electro-optical systems with optical interconnects.

Index Terms— **Multistage interconnection networks (MIN's), embedding, mappings, Markov analysis, reconfiguration, partition of connection requests, time division multiplexing (TDM).**

I. Introduction and Background

CONNECTIVITY and implementation costs are two important factors in designing an interconnection network in parallel and distributed processing systems. A completely connected interconnection network can match the communication requirements of any application, but is too expensive to build. Reconfigurable interconnection networks are alternatives to complete connections. However, the control of reconfiguration is usually a performance bottleneck. This paper proposes an interconnection paradigm for reconfigurable networks that aims to match a relatively low control bandwidth with a high communication bandwidth. The proposed paradigm, which is especially suitable for hybrid electro-optical systems, is applicable to a large class of interconnection networks [22]. In this paper, however, it is discussed in the context of multistage interconnection networks (MIN's).

Manuscript received October 30, 1991; revised October 28, 1993. This work was supported by the U.S. Air Force Office of Scientific Research under Grant AFOSR–89–0469.

R. Melhem is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260.

C. Qiao is with the Department of Electrical and Computer Engineering, State University of New York, Buffalo, NY 14260.

IEEE Log Number 9215371.

MIN's as reconfigurable networks have been studied extensively for multiprocessing applications [2], [8], [13], [20], [30], [32], [36]. An MIN connects a set of input ports to a set of output ports through multiple stages of interconnected switches. A path from an input port to an output port can be established through proper switch settings. In multiprocessing environments, an MIN is used to connect processors to either memory modules or processors. In this paper, an MIN that connects processors to processors will be used to illustrate the application of the proposed interconnection paradigm.

In a circuit-switching MIN, only a limited number of connections (or circuits) can be established simultaneously. A connection that is released in order to accommodate a new connection may need to be established at a later time. Since setting up a connection takes time [14], [18], [19], [26], the process of repeatedly setting up or releasing connections introduces large control overhead. For this reason, the conventional circuit-switching method is not suitable for networks with high communication bandwidth. Especially, it is not suitable for hybrid electro-optical systems where the electronic control speed cannot match the high optical communication bandwidth.

The idea of the proposed paradigm is to repeatedly reconfigure a network through a sequence of configurations to establish the set of connections required by an application. Within each configuration, a subset of these connections is established. The switch settings for each configuration are stored in shift registers, so that the network can change configurations and establish all the required connections efficiently. This idea is an extension of the technique proposed in [34], in which an MIN emulates a completely connected network through the establishment of *all* possible connections from inputs to outputs. By establishing only the connections required by the application, however, the proposed paradigm improves communication efficiency for those applications that do not require all possible connections.

The proposed paradigm, called *Reconfiguration with Time Division Multiplexing* (RTDM) is introduced in Section II in the context of MIN's. The control of RTDM may be either static or dynamic. Static RTDM is applied when a set of connections to be established is known a priori. When this set of connections is regular (e.g., representing a common structure such as a mesh or a hypercube), static RTDM is also referred to as *embedding*. On the other hand, *dynamic* RTDM is applied when the connection requests are generated at run-time. Embeddings of several regular communication structures in time division multiplexed MIN's (TDM–MIN's)

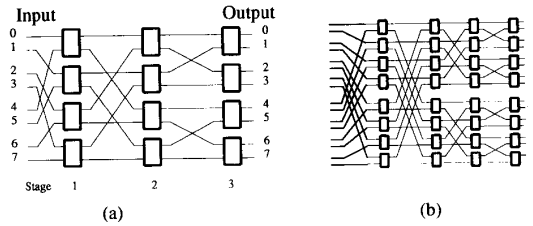


Fig. 1. MIN's with the generalized cube topology.

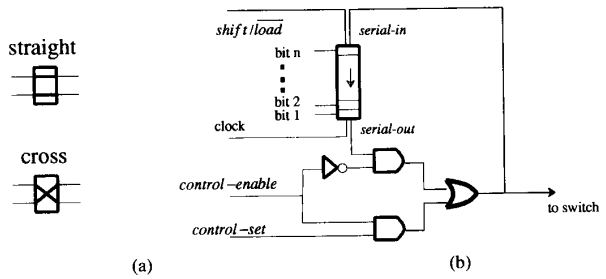


Fig. 2. The two states and a control circuit of a switch.

are presented in Section III. In Section IV, algorithms to perform static reconfiguration based on arbitrary communication structures are discussed. Section V describes dynamic reconfiguration strategies, and Section VI concludes the paper.

II. Time Division Multiplexed MIN's

MIN's under consideration in this paper are generalized cube networks [30], which are topologically equivalent to many blocking MIN's, such as *Data Manipulator, flip, n-cube, Omega, shuffle-exchange,* and *baseline* [2], [8], [13], [20], [32], [36]. Fig. 1(a) shows an example of an 8×8 MIN, with the generalized cube topology consisting of 2×2 switches. A 16×16 MIN can be constructed recursively from two 8×8 MIN's and an extra front stage, as shown in Fig. 1(b). In general, an $N \times N$ MIN with the generalized cube topology can be constructed recursively from two $\frac{N}{2} \times \frac{N}{2}$ MIN's and an extra front stage. In this paper, the stages in an $N \times N$ MIN are numbered from 1 to n from left to right (where $n = \log N$ and all log functions in this paper have base 2).

In an $N \times N$ MIN, there is a unique path between an input port and an output port. Along each path, there are n switches, one at each stage. In order to establish a path, each switch along the path has to be set properly to either a "straight" or "cross" state, both of which are shown in Fig. 2(a). Other switches can be in either state without affecting the path.

Let the set of input ports and the set of output ports of an $N \times N$ MIN be I and O, respectively, where $I = O = \{0, 1, \cdots, N-1\}$. A path in the MIN between $i \in I$ and $j \in O$ is denoted by $p_{i \to j} = (i, j) \in I \times O$. Define a mapping, M, to be a set of paths that can be established simultaneously without conflict. Clearly, a mapping can contain at most N paths. We refer to mappings that contain less than N paths as *partial* mappings. Since establishment of two or more paths at the same time may cause conflicts, not every set of paths is a mapping. We refer to the establishment of all of the paths in a mapping as the *realization* of the mapping.

Given a mapping, there is a way to set switches in an MIN to realize the mapping. Let $m = \frac{N}{2}$ be the number of switches per stage in the MIN and define a *switch setting* array to be an $m \times n$ array whose $[i, j]$th element corresponds to the ith switch at the jth stage in an MIN. Denote the switch-setting array of a mapping M by SS_M and its elements by $SS_M[i, j]$. The value of an element of SS_M is "0" or "1" if the corresponding switch has to be set to "straight" or "cross" to realize mapping M. The value of an element is "$\times$" (meaning do not care) if the corresponding switch can be in any state without affecting the realization of the mapping, in which case, the mapping must be a partial mapping. Two mappings, M_1 and M_2, are said to be not *compatible* with each other if there are some i and j such that the two elements, $SS_{M_1}[i, j]$ and $SS_{M_2}[i, j]$, are either "0" or "1," but not equal. That is, M_1 and M_2 contain some connections that cause conflicts in switch setting. Otherwise, M_1 and M_2 are said to be *compatible* with each other, in which case, the two mappings can be merged into one mapping, namely $M = M_1 \cup M_2$.

Given a set of paths $E \subseteq I \times O$, it may not be possible to establish all paths in E at the same time without conflicts. E, however, can be partitioned into several mappings, $E = M_1 \cup M_2 \cup \cdots \cup M_t$. Each mapping $M_i, i = 1, 2, \cdots, t$ may be realized for a fixed length of time, which we call a *time slot.* By doing so, every path in E is established once in a time slot, and E is said to be realized through TDM. Note that switch-setting arrays for different mappings are usually different. Therefore, the MIN has to change its switch setting after each time slot.

A TDM–MIN is an MIN that repeatedly realizes a sequence of mappings through time division multiplexing in a round-robin fashion. More specifically, a t-way TDM–MIN changes its switch setting in each time slot to realize one of the t mappings, $M_1, M_2, \cdots, M_t$, in a round-robin fashion. Without loss of generality, we assume that M_i is realized during the ith time slot ($1 \leq i \leq t$). We call the ordered sequence $[M_1, M_2, \cdots, M_t]$ a *configuration sequence* of the t-way TDM–MIN and the number of mappings in the sequence the *multiplexing cycle length* (MCL).

If the set of connections to be established is known a priori, a sequence of configurations that a network will go through can be predetermined. This is referred to as *static* reconfiguration. If there are run-time requests to establish connections, *dynamic* reconfiguration is needed, and the sequence of configurations will have to be changed in order to accommodate these requests. In either case, once a configuration sequence has been determined, the switch setting for each configuration in the sequence can be stored in shift registers for fast retrieval. More specifically, as shown in Fig. 2(b), each switch is assumed to have a shift register that stores the sequences of states to which the switch should be set. The kth bit of the shift register of the ith switch at stage j is either "0" or "1" if the corresponding element in SS_{M_k} is "$\times$." Otherwise, it should be equal to $SS_{M_k}[i, j]$. At the beginning of each time slot, the register shifts out one bit that is used to set the switch (assuming that the signal *control-enable* is low). Whenever there is a need to set the switch by using the external signal *control-set* (as is the case in dynamic reconfiguration), the signal *control-enable*

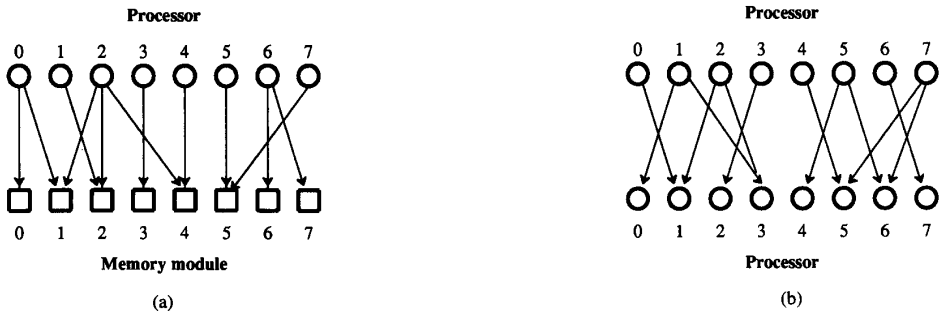


Fig. 3. Examples of connection request (CR) graphs.

is set to high. This external signal is then stored in the shift register, overwriting its original content.

A global clock may be used to synchronize all input and output ports, as well as the switches in the MIN. The input and output ports are informed of the configuration sequence, so that the time slot in which a connection is established is known to all input and output ports. As a result, for source (or destination) processors, routing decisions become as simple as choosing an appropriate time slot to transmit (or receive). No intermediate message buffering or address decoding is needed at the switches.

A. Determination of a Configuration Sequence

Communication requirements of an application can be represented by a bipartite graph, which we call a *connection request* (CR) graph.

Fig. 3 shows two examples of CR graphs, one for shared-memory systems and another for message-passing systems. Fig. 3(a) shows a CR graph based on processor-to-memory connections. A directed edge from processor i at the top to memory module j at the bottom means that processor i may address memory module j. On the other hand, Fig. 3(b) shows a CR graph based on interprocessor connections. A directed edge from a source processor i at the top to a destination processor j at the bottom means that processor i may send messages to processor j. An edge from processor i to itself is meaningless in CR graphs for interprocessor connections. Note that because of the dynamic nature of memory access requests in many applications, it is relatively difficult to construct CR graphs for shared-memory systems.

A node in a CR graph is either a *source node* or a *destination node*. Let source node $i, 0 \leq i \leq N-1$, use input port i of an $N \times N$ MIN, and let destination node $j, 0 \leq j \leq N-1$ use output port j of the same MIN. Therefore, an edge from source node i to destination node j in a CR graph requires the establishment of path $p_{i \rightarrow j}$ in the MIN. We use the same notation for a path to denote an edge, and use the terms "edge" and "path" interchangeably.

Denote the set of all edges in a CR graph by E, and the number of edges in the set by ε. As an example, the set E in the CR graph in Fig. 3(b), with $\varepsilon = 12$, is given in (1) below.

$$E = \{(0,1),(1,0),(1,3),(2,1),(2,3),(3,2),(4,5),(5,4),(5,6),(6,7),(7,5)(7,6)\}. \quad (1)$$

Given a CR graph, we call a configuration sequence $[M_1, M_2, \cdots, M_t]$ a *minimal connection* (MC) configuration sequences for the CR graph if it satisfies the following two conditions.

1) $E \subseteq \cup_{i=1}^{t} M_i$.
2) For any $i, j \in \{1, 2, \cdots, t\}$, M_i and M_j are not compatible.

The first condition states that any edge $(i,j) \in E$ is established in a mapping M_i, and the second condition states that no two mappings in the configuration sequence can be merged into one. We call a configuration sequence for a CR graph *optimal* if it has the *least* MCL among all other configuration sequences for the same graph. Note that if t is equal to the maximum degree of a node in a CR graph, then the configuration sequence is optimal, because only one-to-one connections are allowed in a mapping.

Based on the results presented in [4], the problem of determining whether a given permutation (or a subpermutation) can be partitioned into three or more conflict-free subsets in blocking MIN's is an NP-complete problem. Since a permutation is a special instance of a set of arbitrary paths, the problem of determining an optimal configuration sequence is more general and thus is an NP-hard problem. Nevertheless, optimal configuration sequences can be obtained for several regular communication structures, as discussed in the next section. In case optimal results cannot be obtained, heuristic algorithms can be used to find near optimal results (e.g., MC configuration sequences). Previously, the problem of realizing permutations in MIN's with multiple passes has been studied in [1], [25], [29], [37]. Related analytic models and heuristic algorithms have been developed in [5], [6], [15], [33], [38]. In order to efficiently partition a set of arbitrary connections, which may not be a permutation, into conflict-free subsets, different heuristic algorithms are needed. These algorithms and their performance evaluations are presented in Section IV.

III. Embeddings of Regular Communication Structures

The ability to embed regular communication structures efficiently is important, because there are many existing applications designed for them. Embedding communication structures in an MIN in the space domain has been studied previously. For example, the problem of satisfying a set of resource

allocation requests in rectangular SW banyans has been studied with the assumption that all output ports of the network are equivalent [10], [21]. Embedding of unidirectional binary trees in an MIN with broadcasting capabilities has been studied in [16]. In all of the above work, however, no time domain embedding is involved.

Finding a configuration sequence for a CR graph representing a regular communication structure can be regarded as *embedding* in the space and time domains. The MCL of a configuration sequence is a measure of the efficiency of the embedding. This measure is, in some sense, similar to the dilation cost of embeddings in the space domain. We also define *path utilization* (PU) to be the ratio of the number of connections required versus the number of connections that can be established in one multiplexing cycle. That is, $PU = \frac{e}{Nt}$. This measure is, in some sense, similar to the measure of expansion cost in space domain embeddings. In the following sections, embeddings of several common structures, such as completely-connected networks, rings, meshes, hypercubes, cube-connected-cycles, and binary trees in a TDM–MIN, are discussed.

B. Completely Connected Network

Because there are N^2 paths between every input port and every output port in a completely connected CR graph, and because at most N paths can be established in each mapping, an MC configuration sequence that embeds a completely connected network has at least N different mappings. We call a configuration sequence $[M_1, M_2, \cdots, M_N]$ a *completely connected* (CC) configuration sequence if

$$\bigcup_{i=1}^{N} M_i = I \times O. \tag{2}$$

Every path in a completely connected network is established in one of the N mappings of a CC configuration sequence. Such an embedding is clearly an optimal one with its multiplexing cycle length $t = N$ and path utilization $PU = 1$. There is more than one CC configuration sequence. As one example, define

$$M_{s(k)} = \{p_{i\to j} \mid j = (i+k) \bmod N \text{ for } i = 0, 1, \cdots, N-1\}, \tag{3}$$

and call it a *shift-k* mapping. The mapping can be realized with partial stage control if k is a power of 2, and with individual switch control otherwise [2], [13]. Therefore, the configuration sequence $[M_{s(0)}, M_{s(1)}, \cdots, M_{s(N-1)}]$ establishes paths from any input port to all N output ports, and thus is a CC configuration sequence. As another example, define

$$M_{f(k)}\{p_{i\to j} \mid j = x \text{ or } k \text{ or } i = 0, 1, \cdots, N-1\}, \tag{4}$$

and call it a *flip-k* mapping where *xor* is the bitwise *Exclusive–OR* operation. The *flip-k* mapping can be realized by individual stage control as in [3]. Therefore, the configuration sequence $[M_{f(0)}, M_{f(1)}, \cdots, M_{f(N-1)}]$ is also a CC configuration sequence. Note that CC configuration sequences are functionally equivalent in terms of their multiplexing cycle lengths and path utilizations. However, the CC configuration sequence with *flip-k* mappings may be chosen for the purpose of embedding a completely connected network in the space and time domains because of its control simplicity. It is also worth noting that in the case of processor-to-processor interconnection, *identity* mappings, such as $M_{s(0)}$ or $M_{f(0)}$, that establish no paths other than those from a node to itself can be deleted from CC configuration sequences.

C. Other Regular Networks

Since any CR graph is a subgraph of a completely connected graph, it can be embedded in any CC configuration sequence of a TDM–MIN. This, however, requires an N-way TDM–MIN, and thus may be inefficient in terms of both the multiplexing cycle length and the path utilization. An alternative is to find an MC configuration sequence of length $t < N$ which embeds the CR graph. In other words, a t-way TDM–MIN instead of an N-way TDM–MIN, for some $t < N$, can be used to increase the embedding efficiency. The smaller the t is, the more efficient it is to use such an MC configuration sequence. In the remainder of this section, we present optimal embeddings of several regular communication structures with N nodes into $N \times N$ MIN's.

Ring: In a bidirectional ring network, processor i is connected to processor $(i \pm 1) \bmod N$. Therefore, the set of edges in the CR graph is

$$E = \{p_{i\to j} \mid j = (i \pm 1) \bmod N, i = 0, 1, \cdots, N-1\}. \tag{5}$$

That is, each source or destination node in the CR graph has two outgoing or two incoming edges, respectively. According to (3), the mapping $M_{s(1)}$ establishes a path from i to $(i+1) \bmod N$, and the mapping $M_{s(N-1)}$ establishes a path from i to $(i-1) \bmod N$. Therefore, the configuration sequence $[M_{s(1)}, M_{s(N-1)}]$ is an MC configuration sequence for the bidirectional ring structure. It is optimal with $t = 2$ and $PU = 1$. Note that an embedding for unidirectional rings is straightforward, and that embeddings for augmented rings (such as *chordal rings*) can be similarly obtained.

Mesh: We first consider a wraparound $m \times m$ mesh where $N = m \times m$. The set of edges in its CR graph is

$$E = \{p_{i\to j} \mid j = (i \pm 1) \bmod N\} \bigcup \{p_{i\to j} \mid j = (i \pm m) \bmod N\}. \tag{6}$$

Therefore, the configuration sequence $[M_{s(0)}, M_{s(N-1)}, M_{s(m)}, M_{s(N-m)}]$, is an MC configuration for the mesh. Obviously, the above embedding is optimal with $t = 4$ and $PU = 1$.

Note that edges of an ordinary mesh without wraparound links is a subset of E in (6). In fact, the number of (directed) edges in such a mesh is $4(N - m + 1)$; therefore, $PU = \frac{(N-m-1)}{N}$ if the same configuration sequence is used. Nevertheless, because some processors are connected to four other processors, the above embedding, which has $t = 4$, is optimal.

Binary Hypercube: Let the processor number in a binary hypercube of $N = 2^n$ nodes be represented by their binary addresses. Let the binary address representation of i be $a_{n-1} \cdots a_d \cdots a_0$. Define the function $Ngh(d, i)$ to be

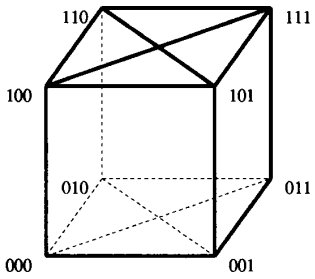


Fig. 4. An example of an enhanced cube.

the neighbor of node i along the dth dimension. That is, $Ngh(d,i) = j = a_{n-1}\cdots\overline{a_d}\cdots a_0$. Because each processor in the hypercube is connected to n others, one in each dimension, the set of edges E consists of n subsets. Each subset E_d contains edges that span the dth dimension. More specifically, we have

$$E = \bigcup_{d=0}^{n-1} E_d, \tag{7}$$

where

$$E_d = \{p_{i\to j} \mid j = Ngh(d,i)\}. \tag{8}$$

Since the function $Ngh(d,i)$ is equivalent to the *flip-k* function in (4) when $k = 2^d$, the mapping $M_{f(2^d)}$ establishes all paths in E_d. Therefore, an MC configuration sequence for the hypercube is $[M_{f(2^0)},\cdots,M_{f(2^d)},\cdots,M_{f2^{(n-1)}}]$, which is optimal with $t = n = \log N$ and $PU = 1$.

From the CC configuration sequence with *flip-k* mappings, it is clear that *any* augmented hypercube can be embedded by adding appropriate mappings into the above configuration sequence. For example, Fig. 4 shows an enhanced cube, as described in [35]. Its optimal configuration sequence can be obtained by adding mapping $M_{f(3)}$ into the sequence $[M_{f(1)}, M_{f(2)}, M_{f(4)}]$. Configuration sequences for other variations of hypercubes [31] can be similarly obtained.

Cube-Connected-Cycle (CCC): Consider a CCC of 2^n nodes, and let r be the smallest integer for which $r + 2^r \geq n$. Fig. 5 shows a CCC of 32 nodes with $n = 5$ and $r = 2$. In a CCC, each processor has an n-bit address k, which may be expressed as a pair $(l,\ p)$ of integers represented with $(n-r)$ and r bits, respectively, such that $k = l \times 2^r + p$. Each processor has three bidirectional ports: F, B and L (mnemonic for *Forward, Backward* and *Lateral*). Connection between ports of processors are determined by the processor address $(l,\ p)$, such that

$F(l,p)$ is connected to $B(l,(p+1) \bmod 2^r)$;
$B(l,p)$ is connected to $F(l,(p-1) \bmod 2^r)$;
$L(l,p)$ is connected to $L(l+\sigma 2^p,p)$.

Let $bit_p(l)$ be the pth least significant bit of the binary representation of l, which gives $\sigma = 1 - 2bit_p(l)$. In words, the processors in a CCC are grouped into 2^{n-r} cycles, with each cycle consisting of 2^r processors, circularly connected by F–B and B–F connections. The cycles are connected as an $(n-r)$-cube by L-L (lateral) connections.

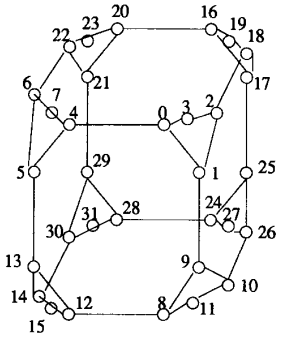


Fig. 5. A CCC of 32 nodes.

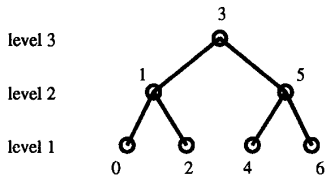


Fig. 6. In-order labeling of a two-level tree.

The F–B and B–F connections in each cycle can be embedded in a $2^r \times 2^r$ MIN similar to the ring embedding described earlier. By partitioning an $N \times N$ MIN into 2^{n-r} such smaller MIN's [30], all F–B and B–F connections in the CCC can be embedded in two time slots, one for all F–B connections and another for all B–F connections. For L–L connections, we note that two processors connected through their *lateral* ports have the same address bits, except for the $(p+r)$th least insignificant bit. According to Theorem 2, provided by Lawrie in [13], it is straightforward to conclude that all *lateral* connections can be established in one mapping. Therefore, a CCC can be embedded *optimally* in a TDM–MIN with $t = 3$.

Binary Tree: Consider a $(2^n - 1)$-node complete binary tree with in-order labeling of the nodes. More specifically, let the leaves of the tree be at level 1, and assign to the root, which is at level n, the label $2^{n-1} - 1$. Call a subtree with root at level L an L-level tree. Assume that the root of a d-level subtree $(1 \leq d < n)$ is assigned the label $R(d)$; then the root of its left subtree, with $d-1$ levels, is assigned the label $R(d) - 2^{d-2}$, and the root of its right subtree is assigned the label $R(d) + 2^{d-2}$. Fig. 6 shows a three-level complete binary tree with in-order labeling.

Assume that edges between a parent node and its children in the tree are bidirectional. The maximum degree of a node in the tree is 3. This suggests that there may exist an embedding of the tree into an $N \times N$ TDM–MIN (where $N = 2^n$), which requires only three mappings. Although the existence of such an embedding is not warranted, it is shown by a lemma in the appendix that with in-order labeling of the tree nodes, any embedding needs at least four mappings.

No matter how the tree nodes are labeled, four mappings are minimal if the children-to-parents connections in the tree are to be embedded separately from the parents-to-children

TABLE I
SUMMARY OF THE EMBEDDING RESULTS

Structure	Number of nodes	MCL	Optimality
Ring	N	2	Yes
Mesh	$N = m^2$	4	Yes
Hypercube	$N = 2^n$	n	Yes
Cube-Connect-Circle	$N = 2^n$	3	Yes
Complete Binary Tree	$N = 2^n - 1$	4	?

connections. More specifically, partition the tree into two trees, T_{cp} and T_{pc}, with unidirectional edges for children-to-parent connections and for parent-to-children connections, respectively. Because the maximum degree of a node in either T_{cp} or T_{pc} is 2, any embedding of each of these two trees requires at least two mappings. Therefore, if edges in T_{cp} will not be established in the same time slots as edges in T_{pc}, at least four mappings will be needed, no matter how the tree nodes are labeled. Below we demonstrate an embedding in which edges in T_{cp} and T_{pc} are established in separate mappings with in-order labeling of the tree nodes. The embedding uses four mappings, and is therefore optimal with $t = 4$ and $PU = \frac{(N-2)}{N}$.

Consider first the edges in T_{cp}. In order to establish them in two mappings, edges in T_{cp} are further decomposed into two subsets, one containing all of the edges from a left child to its parent, and the another containing all of the edges from a right child to its parent. As shown in the proof of the following theorem, which is given in the appendix, the edges in each of the two subsets of T_{cp} can be established in one mapping. Thus, we have Theorem 1.

Theorem 1: All edges in tree T_{cp} can be established in two mappings.

Similarly, we can have the following theorem, which is also proved in the appendix.

Theorem 2: All edges in T_{pc} can be established in two mappings.

Table I summarizes the embedding results presented in this section.

IV. STATIC RECONFIGURATION BASED ON ARBITRARY CR GRAPHS

In the preceding section, we have demonstrated several optimal embeddings of regular structures in the TDM–MIN. As mentioned earlier, the problem of finding an optimal configuration sequence for an arbitrary set of requests is NP-hard. In this section, we present several heuristic algorithms and analyze their performances. The average MCL, denoted by t_{av}, is used to indicate how efficient an algorithm is in determining a TDM configuration sequence for a given fixed-size set of arbitrary requests. In particular, we use the normalized MCL, which is $\frac{t_{av}}{N}$, as a performance measure. Clearly, the smaller the ratio is, the better an algorithm performs.

A. Selecting an MC Configuration

For arbitrary CR graphs, an MC configuration sequence can always be obtained by selecting a subset of mappings from a CC configuration sequence. Such an MC configuration sequence can often improve performance by reducing the multiplexing cycle length to t time slots, for some $t < N$. Note that, given a specific path, there is only one mapping in a CC configuration sequence that establishes that path. The mapping can usually be determined by either a simple arithmetic operation or a table look-up. For example, if the CC configuration sequence consists of flip-k mappings, the mapping that establishes the path $p_{i\to j}$ is $M_{f(k)}$, where $k = i \ xor \ j$.

Given a CR graph containing a set of edges E and CC configuration sequence $[M_1, M_2, \cdots, M_N]$, an MC configuration sequence that established the edges in the CR graph can be found by using the selection algorithm below. We use the symbol "[ ]" to denote an empty MC configuration sequence with no mappings and the operation "||" to denote the addition of a mapping to an MC configuration sequence.

Selection Algorithm:

1. Set MC = []
2. For each edge $p_{i\to j} \in E$ repeat
   2.1. Determine the mapping M_k such that $p_{i\to j} \in M_k$
   2.2. If $M_k \notin$ MC then MC=MC $\| M_k$

End of the algorithm.

For example, consider the CR graph in Fig. 3(b) and the CC configuration sequence consisting of flip-k mappings. The MC configuration sequence selected by the algorithm is $[M_{f(1)}, M_{f(2)}, M_{f(3)}]$. Since $t = 3$ and $N = 8$, a three-way rather than an eight-way TDM–MIN may be used for the application to improve the efficiency. Note that because the maximum degree of a node in the graph is 2, this configuration sequence may not be optimal. In fact, an optimal MC configuration sequence with $t = 2$ is obtained in the next section.

Probabilistic analysis of the average multiplexing cycle length of MC configuration sequences for random CR graphs can be carried out as follows. Assume that S out of N source nodes are each connected randomly to D out of N destination nodes. The probability that an edge is established in any mapping of a given CC configuration sequence is $\frac{1}{N}$. Because edges that originate at the same source node must be established in different mappings, exactly D mappings are needed to establish paths from a source node to D destination nodes. Therefore, after selecting D mappings from the given CC configuration sequence, the probability that any mapping has *not* been selected is $p = 1 - \frac{D}{N}$. Because each source node randomly and independently selects D mappings, the probability that a mapping in the CC configuration sequence has *not* been selected after all S source nodes have selected their mappings is $P = p^s$. Denote by $C(n, r)$ the number of ways to choose r out of n distinct objects. That is,

$$C(n,r) = \binom{n}{r} = \frac{n!}{r! \times (n-r)!}. \tag{9}$$

The probability that exactly i mappings have been selected for an MC configuration sequence is thus

$$\text{Prob}(i) = C(N,i) \times P^{N-i} \times (1-P)^i$$
$$\text{where} \quad P = (1 - \frac{D}{N})^s. \tag{10}$$

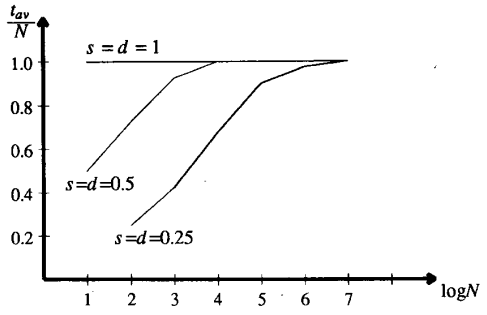


Fig. 7. Performance of the Selection Algorithm.

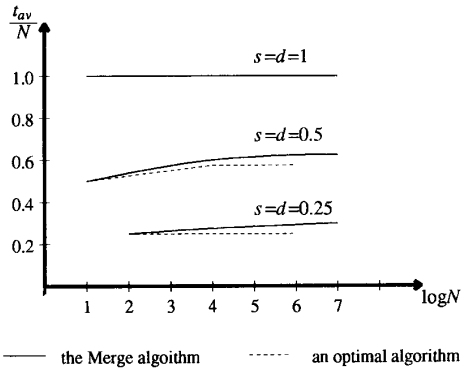


Fig. 8. Performance of the Merge Algorithm.

Therefore, the average (expected) number of mappings selected, that is, the expected MCL of an MC configuration sequence, is

$$t_{\text{av}} = \sum_{i=d}^{N} i \times \text{Prob}(i). \tag{11}$$

Clearly, $\max(S, D) \leq t_{\text{av}} \leq N$. Define $s = \frac{S}{N}$ and $d = \frac{D}{N}$. The normalized communication load of an application can be approximated by $s \times d$. The heavier the load is, the larger the ratio of $\frac{t_{\text{av}}}{N}$ will become. Fig. 7 shows calculated values of the normalized MCL versus load (when $s = d$) for different network size N. It can be seen that the Selection algorithm generates an MC configuration sequence that improves over a CC configuration sequence when the network size is small and the communication load is low.

In order to improve the performance in large networks with medium to high loads, the following algorithm can be used.

The Merge Algorithm:

1. Run the Selection algorithm to determine an MC configuration sequence.
2. For each mapping $M_k \in$ MC, repeat step 3
3) If every $p_{i\to j} \in M_k$ is such that $\{P_{i\to j}\}$ is compatible with an M_l currently in MC where $l \neq k$
   3.1. For every $p_{i\to j} \in M_k$
   $M_l = M_l \cup \{p_{i\to j}\}$ if $\{p_{i\to j}\}$ is compatible with M_l
   3.2. Remove M_k from MC

⟨ *End of the algorithm*⟩

More specifically, the above algorithm examines each mapping initially selected to see if it can be deleted from the configuration sequence by *migrating* its paths to other mappings in the sequence. The performance of the Merge algorithm is determined through simulations, and the results are shown in Fig. 8. The minimal MCL values are also plotted in the figure. These minimal values are obtained by using an exhaustive search algorithm. for $N > 64$, however, exhaustive search is not feasible, because its time complexity is exponential to the number of requests. From Fig. 8, it can be seen that the Merge algorithm improves over the Selection algorithm and achieves near-optimal results under simulated communication loads. Note that though the Selection algorithm is simple, it can be shown that the complexity of the Merge algorithm is as high as $O(\varepsilon^2 N \log N)$, where ε is the number of requests.

B. Composing an MC Configuration

In this section, we present another heuristic algorithm with time complexity smaller than the Merge algorithm. The performance of the algorithm, as shown, is comparable to that of the Merge algorithm. The algorithm, called *Composition* algorithm, does not restrict the choice of mappings to just N mappings as does the Selection algorithm. Instead, it constructs mappings based on the set of connection requests in the CR graph in a greedy fashion. That is, starting with an empty set of paths, a mapping is composed by including as many required paths in E as possible.

The Composition Algorithm:

Set MC = [] and $k = 1$. Repeat until E is empty

1. Reset mapping $M_k = \phi$ and all elements of SS_{M_k} to "×"
2. For each edge $p_{i\to j} \in E$
   If $\{p_{i\to j}\}$ is compatible with M_k
   2.1. $M_k = M_k \cup \{p_{i\to j}\}$ and update SS_{M_k} accordingly
   2.2. delete $p_{i\to j}$ from the set E.
3. MC = MC $||M_k$ and $k = k + 1$

⟨*End of the algorithm*⟩

For example, an MC configuration sequence that is composed by the algorithm for the CR graph in Fig. 3(b) is $[M_1, M_2]$, where

$$M_1 = \{(0,1),(1,0),(2,3),(3,2),(4,5),(5,4),(6,7),(7,6)\}$$
$$M_2 = \{(1,3),(2,1),(5,6),(7,5)\} \tag{12}$$

The switch-setting arrays corresponding to these two mappings are

$$SS_{M_1} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad SS_{M_2} = \begin{bmatrix} \times & 1 & 1 \\ 0 & 1 & 0 \\ 0 & \times & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{13}$$

In this example, the MC configuration sequence comprised by the algorithm is optimal with $t = 2$, which improves over the MC configuration sequence generated by the Selection algorithm. As with the other two heuristic algorithms, however, optimal results cannot be guaranteed.

An analytic model has been developed to evaluate the performance of the Composition algorithm. It is assumed that the source and the destination of a request are randomly selected from the N inputs and outputs of the network,

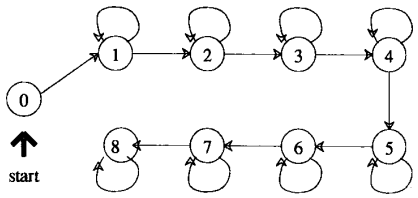


Fig. 9. An example of the Markov chain for $N = 8$.

respectively, with an equal probability. Let $t_{av}(\varepsilon)$ be the average MCL required for a set of ε such random requests. Denote by $P(i|\ \varepsilon)$ the probability that i out of ε requested connections can be established in a mapping where $1 \le i \le \min(N, \varepsilon)$. After i requests are satisfied in the first mapping, the remaining $(\varepsilon - i)$ requests, which are also random, require an average MCL of $t_{av}(\varepsilon - i)$. Therefore, the average MCL for a set of ε requests can be calculated recursively by using the following equations.

$$\begin{aligned} t_{av}(\varepsilon) &= \sum_{t=1}^{\min(N,\varepsilon)} P(i|\varepsilon) \times [1 + t_{av}(\varepsilon - i)] \\ &= 1 + \sum_{t=1}^{\min(N,\varepsilon)} P(i \mid \varepsilon) \times t_{av}(\varepsilon - i) \\ t_{av}(0) &= 0 \end{aligned} \tag{14}$$

The probability $P(i \mid \varepsilon)$ in the above equation can be obtained by using Markov process analysis. Fig. 9 shows a Markov chain for $N = 8$ in which a state transition represents the processing of a new request, and state i $(0 \le i \le N)$ means that exactly i connections are established in the network. The model starts at state 0. After a request is processed (and thereafter satisfied), it moves to state 1. In general, if a new request is satisfied when the model is in state i, the model moves to state $i + 1$. Otherwise, it stays in state i.

Let the transition probability from state i to itself and to state $i + 1$ be denoted by π_{ii} and $\pi_{i(i+1)}$, respectively. The transition matrix for the above Markov chain is

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \pi_{11} & \pi_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \pi_{22} & \pi_{23} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \pi_{33} & \pi_{34} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \pi_{44} & \pi_{45} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \pi_{55} & \pi_{56} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \pi_{66} & \pi_{67} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \pi_{77} & \pi_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We note that by definition, the transition probability π_{ii} is equal to the probability that a new request cannot be satisfied, because of conflict between the new request and the i existing connections. On the other hand, $\pi_{i(i+1)}(= 1 - \pi_{ii})$ is the probability that the new request does not conflict with i existing connections. Therefore, $\pi_{00} = 0$ and $\pi_{01} = 1$, because state 0 is the initial state; and $\pi_{88} = 1$, because state N is an absorbing state. In other words, at most N connections can be satisfied in the network simultaneously. Denote the εth power of T by T^{ε} and its element at the ith row and the jth column by $T^{\varepsilon}[i, j]$, where $0 \le i, j \le N$. By definition, we have

$$P(i \mid \varepsilon) = T^{\varepsilon}[0, i] \tag{15}$$

Therefore, if the transition probabilities of the Markov chain are known, the average MCL can be obtained from equations 14 and 15. To calculate $\pi_{i(i+1)}$ (and subsequently π_{ii}), we let SW be the switch that the new request will use at the first stage of the network. There are three possible cases, namely, the following.

1) Neither of the two inputs of SW has been used by any existing connection, so the new request does not conflict with other connections at SW. In other words, the request passes the first stage.
2) Only one input of SW is used by an existing connection. If the new request uses this same input (case (2a)), then it cannot be satisfied. Otherwise, if the new request uses the other "free" input (case (2b)), then it passes the first stage with probability 0.5.
3) Both inputs of SW are already used; therefore, the new request cannot be satisfied.

Since the new request is randomly generated from any of the N sources, the probability that it may need any of the N inputs is equal. Let $q_1(N, i)$ be the probability that case 1) occurs, and let $q_2(N, i)$ be the probability that case 2b) occurs. Using the notation in (12), these two probabilities can be calculated by the following formulas:

$$q_1(N, i) = \frac{C(N-2, i)}{C(N, i)} = \frac{(N-i)(N-i-1)}{N(N-1)} \tag{16}$$

$$q_2(N, i) = \frac{C(N-2, i-1)}{C(N, i)} = \frac{i(N-i)}{N(N-1)}. \tag{17}$$

After the new request passes through the first stage, it has to pass through a subnetwork A (of size $\frac{N}{2}$) in order to be satisfied (see Fig. 10). This subnetwork will contain some j out of the i existing connections, and the other subnetwork, B, will contain the remaining $i - j$ connections (see Fig. 10). If case 1) occurs, the value of j is such that $\max\left(0, i - \frac{N}{2}\right) \le j \le \min\left(i, \frac{N}{2} - 1\right)$, where the max and min functions are used to ensure that the number of existing connections is at most $\frac{N}{2} - 1$ in subnetwork A and is at most $\frac{N}{2}$ in subnetwork B. On the other hand, if case 2b) occurs, the connection that shares the switch SW will go through subnetwork B, as shown in Fig. 10. Therefore, $\max\left(0, i - 1 - \frac{N}{2}\right) \le j \le \min\left(i - 1, \frac{N}{2} - 1\right)$.

Let $\Phi_1\left(\frac{N}{2}, i, j\right)$ and $\Phi_2\left(\frac{N}{2}, i, j\right)$ be the probabilities that j out of i existing connections go through subnetwork A if case 1) and 2b) occur, respectively. These probabilities are given by

$$\Phi_1\left(\frac{N}{2}, i, j\right) = \frac{C\left(\frac{N}{2} - 1, j\right) \times C\left(\frac{N}{2} - 1, i - j\right)}{C(N-2, i)}, \tag{18}$$

and

$$\Phi_1\left(\frac{N}{2}, i, j\right) = \frac{C\left(\frac{N}{2} - 1, j\right) \times C\left(\frac{N}{2} - 1, i - 1 - j\right)}{C(N-2, i-1)} \tag{19}$$

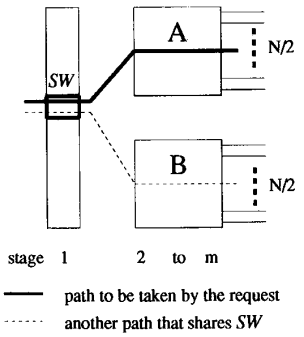


Fig. 10. Illustration of the two subnetworks A and B for Case (2b).

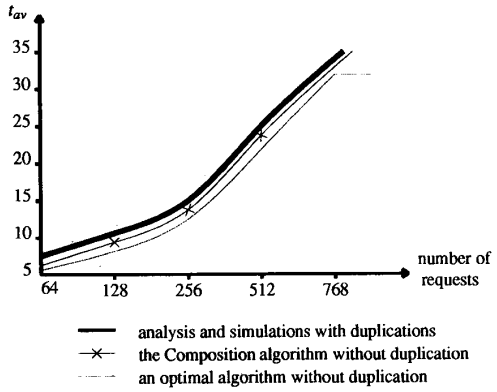


Fig. 11. The average MCL vs. the number of requests in a 32×32 network.

Clearly, $\Phi_1(N/2,i,j)$ and $\Phi_2(N/2,i,j)$ should be 0 if j is not within the ranges specified by the max and min functions for cases 1) and 2b), respectively.

Having derived these probabilities, we may now calculate the probability that the new request will be satisfied in a network of size N, given that i connections already exist in the network. From the above discussion, this probability is equal to the probability that the new request passes the first stage and then passes a subnetwork of size $\frac{N}{2}$, in which some j connections are already established (where the bounds of j are as discussed earlier). Specifically, denote by $\Pi(n,r)$ the probability that a new request will pass a network of size n in which r connections are already established. For $N > 2$,

$$\Pi(N,i) = q_1(N,i) \times \sum_j \Phi_1\left(\frac{N}{2},i,j\right) \times \Pi\left(\frac{N}{2},j\right) + 0.5 \times q_2(N,i) \times \sum_j \Phi_2\left(\frac{N}{2},i,j\right) \times \Pi\left(\frac{N}{2},j\right). \tag{20}$$

Note that the term $\Pi(\frac{N}{2},j)$ in the above equation may be calculated by using a similar recursive equation. Both cases 2a) and 3), however, are not possible at the second and later stages, because a request that passes a stage will not conflict with an existing connection at an *input* port of the next stage. Therefore, when calculating $\Pi(n,r)$ for $n \le \frac{N}{2}$, the terms q_1 and q_2 have to be replaced by q_1' and q_2', respectively, which are obtained from the following formulas

$$q_1'(n,r) = \frac{C(n-2,r)}{C(n-1,r)} = \frac{n-r-1}{n-1} \text{ and}$$
$$q_2'(n,r) = \frac{C(n-2,r-1)}{C(n-1,r)} = \frac{r}{n-1}. \tag{21}$$

Hence, the recursive equation to calculate $\Pi(n,r)$ for $n \le \frac{N}{2}$ can be written as follows.

$$\Pi(n,r) = q_1'(n,r) \times \sum_j \Phi_1\left(\frac{n}{2},r,j\right) \times \Pi(\frac{n}{2},j) + 0.5 \times q_2'(n,r) \times \sum_j \Phi_2\left(\frac{n}{2},r,j\right) \times \Pi\left(\frac{n}{2},j\right)$$
$$\Pi(2,0) = 1$$
$$\Pi(2,1) = 0.5 \tag{22}$$

To summarize, equations (16) through (20) can be used to determine $\Pi(N,i)$. Note that by definition, $\Pi(N,i)$ is equal to the transition probability $\pi_{i(i+1)}$ of the Markov chain in Fig. 9; hence, these equations can be used to determine the transition matrix. Afterward, the average MCL for a set of ε requests can be determined by using (14) and (15).

Note that in the above analysis, it is assumed that requests are randomly selected such that duplicates are possible. This assumption is necessary to preserve the randomness of the set of requests considered for all mappings to be composed. Otherwise, for example, after the first i requests are included in one mapping, the remaining $(\varepsilon - i)$ requests cannot be considered random. In addition, as shown in the next section, duplicate requests are meaningful in the presence of nonuniform bandwidth requirements.

If duplicates are not allowed, the probabilities $q_1(N,i)$ and $q_2(N,i)$ will become larger, resulting in a smaller value of t_{av} for the same number of requests. That is, the analysis will yield an upper bound on the average MCL needed for a set of requests that does not contain duplicates.

Fig. 11 shows the average MCL versus the number of random requests in a network of size 32. The analytic results closely agree with the simulation results when duplicates are allowed. Simulation studies have also been conducted to determine the performance of the Composition algorithm when duplicates are not allowed. As can be seen from Fig. 11, the analytic model generates tight upper bounds on the average MCL for a set of requests without duplicates. From this figure, it is also clear that the results of the Composition algorithm are close to those obtained by an optimal exhaustive search algorithm. When the number of requests is close to N^2 (e.g., ≥ 768 when $N = 32$), however, the average MCL (except that obtained by the exhaustive search algorithm) could exceed 32, as shown in the figure, because the network is a blocking network and the order in which requests are considered affects the way that connections are established.

In order to compare the Composition algorithm with the Selection and Merge algorithms, another set of simulation runs has been done. A random number generator is used to generate D distinct destinations for each of the S sources, which are also determined randomly. Fig. 12 shows the simulation results

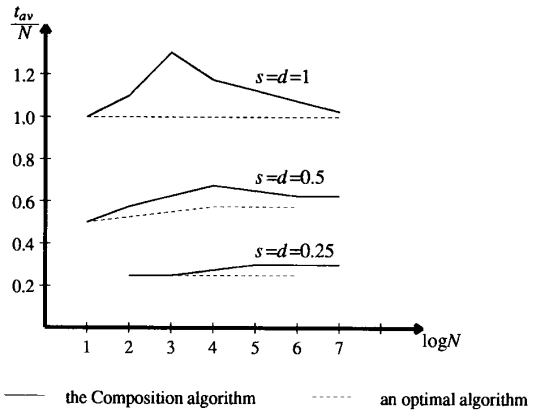


Fig. 12. Performance of the Composition Algorithm.

for different system sizes under different load conditions where $s = \frac{S}{N}$ and $d = \frac{D}{N}$. It can be seen by comparing Fig. 12 with Figs. 7 and 8 that under low or medium load conditions, the performance of the Composition algorithm is somewhat between those of the Selection algorithm and the Merge algorithm. The Composition algorithm has a time complexity of $O(\varepsilon N \log N)$, however, which is less than that of the Merge algorithm. We note that when the load becomes relatively high, the Selection algorithm is preferred because of its simplicity and near-optimal performance.

C. Dealing with Nonuniform Bandwidth Requirements

So far, it was assumed that all connection requests have a uniform bandwidth requirement. It is quite possible that different amount of information (measured in bytes to be transferred, for example) is to be carried on different connections. For some applications, the amount of information to be carried on each connection can be determined by using compile time analysis. In such cases, the communication efficiency may be improved by taking into consideration the nonuniform requirements. In what follows, an example is discussed and some optimization techniques are illustrated.

Given a set of requests E, we may normalize the amount of information to be carried on each requested connection so that the minimum is of a unit weight and all others are approximated by integer numbers. Let the configuration sequence that embeds E prior to optimization be $[M_1, M_2, \cdots, M_t]$. Assume, without loss of generality, that a connection of a unit weight needs to be established for one time slot in order to complete the information transfer. Given that each connection is established in just one mapping in the configuration sequence, the MIN has to go through the sequence as many times as the maximum weight of all connections. If W is that maximum weight, the total number of time slots needed for the execution of the application is thus tW.

Let $p_{i\to j}$ be the connection carrying the maximal weight W, and let all other connections carry a unit weight. It can be seen that certain amount of bandwidth on all connections other than $p_{i\to j}$ is wasted if no optimization is done. More specifically, the amount of bandwidth wasted on these connections in the above example is proportional to $t(W-1)$.

The first step of optimization is to duplicate each request with w copies if the weight carried by the requested connection is w. Each of these duplicated requests is to be processed individually, so that w duplicated connections are established in w mappings. Static reconfiguration algorithms, discussed in a previous section, can then be applied. In the above example, assume that $p_{i\to j}$ is the connection carrying the maximum weight W, and that it can be established in mapping M_k. By using the Composition algorithm, for example, M_k will be generated W times, one for each duplicate of $p_{i\to j}$. This will result in the configuration sequence $[M_1, \cdots M_k, \cdots, M_k \cdots, M_t]$, in which M_k occurs W times. The total number of time slots needed for execution is reduced to $(W + t - 1)$, which is less than tW for any $t, W > 1$.

Note that as a result of the optimization, the MCL will be increased. Although the increase could be as big as the sum of all of the weights, it is conceivable that a smaller MCL can be achieved, because some duplicated connections may be established in mappings existing in the sequence prior to the optimization, and therefore no new mappings are needed. In addition, a new mapping can include several duplicates of different connections. When all is said and done, the communication efficiency is improved by the optimization, because the total execution time is reduced.

From the above discussions, it can be seen that an optimized configuration sequence may contain more than one copy of a mapping. To generalize the notation for such optimized sequences, let $M_1, M_2, \cdots, M_t$ be different mappings in the sequence, and let f be a positive integer function. The function f is such that for $1 \le i \le t, f(i)$ is the number of times mapping M_i appears in the sequence. This results in the MCL of $t' = \sum_{i=1}^{t} f(i)$. Note that for configuration sequences that are not optimized for weighted connection, $f(i) = 1$ for all $1 \le i \le t$; therefore, $t' = t$.

Hardware considerations may limit each switch to have a shift register of K bits. We call a TDM–MIN with this limit a K-way *limited* TDM–MIN. A K-way limited TDM–MIN can act as a t'-way TDM–MIN for any $t' \le K$. If $t' > K$, however, we can partition the configuration sequence into several partial ones. More specifically, let CS be such a configuration sequence, and let $u = \lceil \frac{t'}{K} \rceil$. Configuration sequence CS can be partitioned into u partial configuration sequences, $CS_1, CS_2, \cdots, CS_u$, such that

$$CS = CS_1 || CS_2 || \cdots || CS_u.$$

Each CS_i has an MCL less than or equal to K, and thus can be implemented by shifting one bit of the shift registers in each time slot. The contents of the shift registers may need to be reloaded after K shifting operations, which may be costly at run-time. Therefore, in the presence of nonuniform bandwidth requirements and limits on the size of shift registers, trade-offs have to be made when performing the above optimization.

Replication of an MIN is proposed in [12] for improving performance, as well as reliability, of an MIN. A K-way TDM–MIN can be regarded as a time domain equivalence of the replication approach. One can configure a K-way limited TDM–MIN to a TDM–MIN with MCL of k for any $k \le K$, based on different applications. Therefore, in terms of

hardware and control complexity, a K-way limited TDM–MIN is more flexible and more efficient than a K-replicated MIN.

V. Dynamic Reconfiguration

Static reconfiguration works well if all paths are required to be established from the beginning to the end of execution of an application. For some applications, however, connection requests are generated at run-time. These run-time requests include requests for establishing new paths and releasing existing ones. Fixing a configuration sequence throughout the execution will be inefficient in the presence of these run-time requests. Therefore, different configuration sequences have to be determined at run-time with *dynamic reconfiguration.*

A. Reconfiguration with Centralized Control

With centralized control, dynamic reconfiguration may be done either periodically or incrementally. With the periodic approach, requests are buffered and processed at selected instances. At each instance, a snapshot of the CR graph is constructed based on all paths that need to be established. Static reconfiguration algorithms can then be executed to determine a suitable configuration sequence. With the incremental approach, requests are processed as they are generated. In addition, either a variable or a fixed MCL can be used in determining a configuration sequence. For example, if a request is to establish a path $p_{i\rightarrow j}$, the current configuration sequence is examined to find a mapping M_k that is compatible with $\{p_{i\rightarrow j}\}$. If successful, path $p_{i\rightarrow j}$ is added to mapping M_k, and reconfiguration based on the request is completed.

If, however, the current configuration sequence does not contain any mapping that is compatible with $\{p_{i\rightarrow j}\}$, then the request will be blocked if a fixed MCL is used. Alternatively, some existing paths may be preempted to make room for the request. Various replacement policies may be used, depending on performance criteria. On the other hand, if a variable MCL is allowed, a new mapping that establishes $p_{i\rightarrow j}$ can be added to the configuration sequence. In either case, one may migrate paths in an existing mapping into other mappings, so that the mapping becomes compatible with $\{p_{i\rightarrow j}\}$. In this way, the preemptions of existing paths or the addition of a new mapping may be avoided. Requests to release paths can be processed similarly.

In [24], it is shown that dynamic reconfiguration overhead can be amortized over a sequence of configurations. In addition, dynamic reconfiguration with TDM results in fast and fair response to run-time connection requests, and improves overall network performance.

B. Dynamic Reconfiguration with Distributed Control

Centralized control of dynamically reconfigured TDM–MIN's enables the determination of efficient configuration sequences, but tends to create a bottleneck in the central control unit. In what follows, dynamic reconfiguration in an MIN with distributed control and global synchronization is discussed. To facilitate implementations with distributed control, a fixed MCL, K, is used. If a variable MCL is used, it would be difficult to inform each processor of either the increment or decrement of the MCL in a timely way without a centralized control mechanism.

One way to distribute control is to have a separate control network overlaying the MIN used for data transfer. The separation can be either *logical* or *physical.* With logical separation, the data and control networks share the same physical links and switches that are multiplexed to create two virtual networks.

Instead of letting a control unit process all requests and determine the switch settings, requests are processed in the control network in a packet-switching fashion. In order to determine which time slot (if any) is available to establish the requested connection, each processor sends out a *reservation* packet to its desired destination. On its way to the destination, the packet collects information on the time slots that have not been reserved for other connections. If the packet reaches the destination, it is sent back as an acknowledgment that contains a single value $i, 1 \leq i \leq K$, which is a reserved time slot for establishing the requested connection.

The availability of a time slot for a requested connection depends on the availability of the same time slot along the connection. More specifically, let a connection that passes through n switches $s_1, s_2, \cdots, s_n$ be represented by the sequence $\langle l(s_1), r(s_1), r(s_2), \cdots, r(s_n)\rangle$, where $l(s_i)$ and $r(s_i)$ are the input and output ports of a switch, s_i, used by the connection, respectively. This implies that port $l(s_1)$ is connected to the source and port $r(s_n)$ is connected to the destination. Each port of a switch in the control network maintains a list of available time slots for establishing connections. Denote such a list for a port X by AVAL (X) Every list is initialized to $\{1, 2, \cdots, K\}$ to make all K time slots available. A time slot ts can be reserved successfully for the connection if all of the ports specified in the sequence have ts as one of their available time slots.

As mentioned earlier, each reservation packet collects information on the available time slots on the way to its destination. It does so by maintaining a list initialized to $\{1, 2, \cdots, K\}$ and *intersecting* the list with those maintained by each port that it visits. If the resulting list becomes empty, the reservation fails. Otherwise, one of the available time slots will be selected by the destination, and the source will be acknowledged. Once a time slot is reserved successfully for a connection, it is deleted from the lists maintained by all ports visited by the reservation packet.

Since a time slot may not be reserved until the reservation packet gets back from its destination, the lists maintained by ports visited by the packet shall not be modified by other reservation packets. To ensure this, mutual exclusion operations on these lists can be performed by each reservation packet. Without loss of generality, it is assumed that "lock" and "unlock" are used at each switch port. Initially, all ports are "unlocked" to permit access. After a port is "locked" by a reservation packet, other packets cannot go through that port and are buffered at the switch. When a packet reaches its destination, or fails to maintain a nonempty list along the way to its destination, it reverses its propagation direction and unlocks ports visited along its way back to the source.

When a connection needs to be released, the source processor sends out a *cancellation* packet that contains the time slot during which the connection is established. The cancellation packet makes its way to its destination and releases the time slot at all the ports that it visits. In the following algorithm, the procedure *Establish()* will be used to send out a reservation packet, denoted by R–P, and the procedure *release()* will be used to send out a cancellation packet, denoted by C–P. A reservation packet maintains a list, AVAL (R–P), of available time slots for establishing the requested connection while a cancellation packet maintains a time slot $\{ts\}$ to be released.

The Reservation/Cancellation Algorithm
(with distributed control)
Establish $(p_i \rightarrow s = < l(s_1), r(s_1), r(s_2), \cdots, r(s_n) >)$
1. The source node connected to $l(s_1)$ generates a reservation packet R-P
   Lock port $l(s_1)$ and set AVAL (R-P) = AVAL ($l(s_1)$)
2. For $i = 1$ to n do
   2.1. Lock Port $r(s_i)$, and set AVAL (R-P) = AVAL $(R - P) \bigcap$ AVAL $(r(s_i))$
   2.2. If AVAL $(R - P) = \phi$ then
      2.2.1. For $j = i$ down to 1 do unlock port $r(s_j)$
      2.2.2. The source node realizes that the connection is not established
      2.2.3. Exit this procedure
3. Choose a time slot $ts \in$ AVAL(R-P)
4. For $i = n$ downto 1 do
   4.1. AVAL $(r(s_i))$ = AVAL $(r(s_i)) - \{ts\}$ and unlock port $r(s_i)$
   4.2. Record the state of switch s_i during time slot ts in the mapping generator
5. AVAL $(l(s_1))$ = AVAL $(l(s_1)) - \{ts\}$ and unlock $l(s_1)$

Release $(p_{i \rightarrow j} = \langle l(s_1), r(s_1), r(s_2), \cdots, r(s_n) \rangle)$
1. The source node connected to $l(s_1)$ generates a cancellation packet C-R with AVAL $(C - R) = \{ts\}$
   Lock port $l(s_1)$. Set AVAL $(l(s_1))$ = AVAL $(l(s_1)) \cup \{ts\}$.
   and unlock $l(s_1)$
2. For $i = 1$ to n do
   2.1. Lock port $r(s_i)$
   2.2. AVAL $(r(s_i))$ = AVAL $(r(s_i)) \cup \{ts\}$
   2.3. Unlock port $r(s_i)$

⟨*End of theAlgorithm*⟩.

VI. Concluding Remarks

Reconfiguration with TDM are suitable for MIN's with either electronic or optical interconnects. It is especially suitable for the latter, because the technique matches hybrid electro-optical technology in many ways. First, 2×2 Lithium Niobate switches, which are optical switches controlled electronically, are feasible for implementing blocking networks of moderate sizes. A large number of processors can be connected to each network port, thus generating several connection requests at a time. Such requests can be processed efficiently by using TDM. Second, time division multiplexing techniques used to reconfigure MIN's are effective in using the high optical bandwidths of communication links [23]. In addition, reconfiguration with TDM does not require buffering and address decoding to route messages. This eliminates the needs for optical delay loops and conversions between optical and electronic signals [27], [28], which are costly in multiprocessor systems. Last, the synchronous feature of switch settings in a TDM–MIN, together with the unique properties of optical signal propagation, namely, unidirectional propagation and predictable delay per unit length, makes message pipelining between stages feasible [11], [17], [22].

In this paper, efficient embeddings of several common structures into TDM–MIN's are presented. The design and analysis of heuristic algorithms for partitioning a set of arbitrary connections into conflict-free subsets are discussed. In addition, dynamic reconfiguration strategies, along with an algorithm with distributed control is proposed. It is shown that static reconfiguration of TDM–MIN's provides architectural flexibilities without run-time control overhead, thus achieving high bandwidth utilization. In [24], it is shown that dynamic reconfiguration of TDM–MIN's results in fast and fair response to run-time requests and improves overall network performances.

Finally, we note that application of the RTDM paradigm is not restricted to MIN's. In fact, it is shown in [22] that RTDM may be applied to any network that can be reconfigured to establish different subsets of input to output connections. Such networks includes busses, crossbars, and wavelength division multiplexed (WDM) systems [7], [9].

Appendix
Embedding a Binary Tree in a TDM–MIN

This appendix contains a lemma on the lower bound of the number of mappings needed to embed a binary tree with in-order labeling of the tree nodes. It also contains the induction proofs of Theorem 1 and 2. Readers are advised to refer to the paper for the notations and terms used in this appendix.

Lemma: Given an L-level binary tree with in-order labeling of the tree nodes, $L \geq 3$, the number of mappings required to establish all of its edges has a lower bound of 4.

Proof: Consider the tree with $L = 3$ as given in Fig. 6. It contains children-to-parents connections (1, 3) and (5, 3), and parents-to-children connections (1, 0) and (1, 2). Let mappings that establish these edges be M_1, M_2, M_3, and M_4, respectively.

Clearly, M_1 is not compatible with M_2, because output port 3 is used in both mappings. M_1 is not compatible with M_3 or M_4, because input port 1 is used in all three mappings. In addition, input ports 1 and 5 share the same switch at the first stage (see Fig. 1), and M_2 requires that $SS_{M_2}[1, 1] = 1$; but M_3 and M_4 require that $SS_{M_3}[1, 1] = 0$ and $SS_{M_4}[1, 1] = 0$, respectively. Therefore, M_2 is not compatible with M_3 and M_4. Finally, M_3 is not compatible with M_4, because input port 1 is used in both of them. Note that the relation "compatible" is symmetric; therefore, none of the above four mappings are compatible with each other. This means that at least four

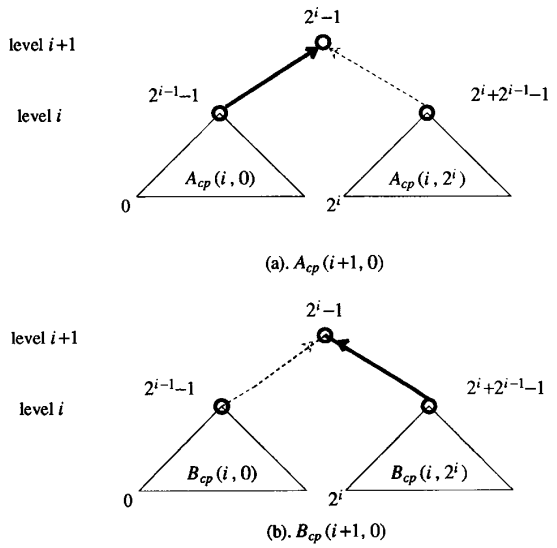


Fig. 13. Recursive constructions of A_cp and B_cp.

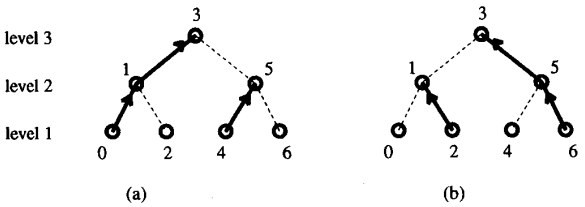


Fig. 14. Example of A_cp and B_cp.

mappings are needed to embed a binary tree having the tree in Fig. 6 as a subtree; thus, the lemma is proven.

Proof of Theorem 1: Denote the set of children-to-parents connections in an L-level tree by $T_{\rm cp}(L,b)$ if the leftmost leaf of the tree is numbered b. In order to prove that all of the edges in $T_{\rm cp}$ can be established in two mappings, partition the set $T_{\rm cp}(l,b)$ into two subsets, $A_{\rm cp}(L,b)$ and $B_{\rm cp}(l,b)$, such that $T_{\rm cp}(L,b) = A_{\rm cp}(L,b) \cup B_{\rm cp}(L,b)$. These two subsets are recursively defined with respect to edges in the left subtree and the right subtree of $T_{\rm cp}$, as follows.

$$A_{\rm cp}(i+1,b) = A_{\rm cp}(i,b) \cup A_{\rm cp}(i,b+2^i) \cup (2^{i-1}+b-1, 2^i+b-1) \quad \text{(A1)}$$

$$B_{\rm cp}(i+1,b) = B_{\rm cp}(i,b) \cup B_{\rm cp}(i,b+2^i) \cup (2^i+2^{i-1}+b-1, 2^i+b-1), \quad \text{(A2)}$$

where $2 \le i \le L$, $A_{\rm cp}(2,b) = (b, b+1)$, and $B_{\rm cp}(2,b) = (b+2, b+1)$.

Fig. 13 illustrates recursive construction procedures outlined in (A1) when $b = 0$. Bold edges are included in the subsets, and dotted edges are not. Fig. 14 shows an example of $A_{\rm cp}(3,0)$ and $B_{\rm cp}(3,0)$.

Denote a $2^{i+1} \times 2^{i+1}$ MIN with ports numbered from b to $2^{i+1}+b-1$ by $MIN(i+1,b)$. According to the topology of a generalized cube network, such an MIN can be constructed from two sub-MIN's, $MIN(i,b)$ and $MIN(i,b+2^i)$, as illustrated in Figs. 15 and 16. Given that a set E of paths are

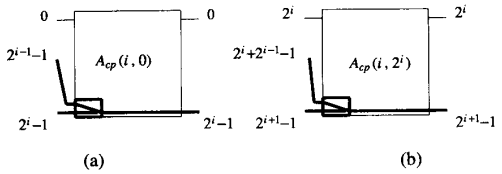


Fig. 15. Embedding of A_cp. (a) $A_cp(i+1,0)$ in $MIN(i,0)$. (b) $A_cp(i,2^i)$ in $MIN(i,2^i)$.

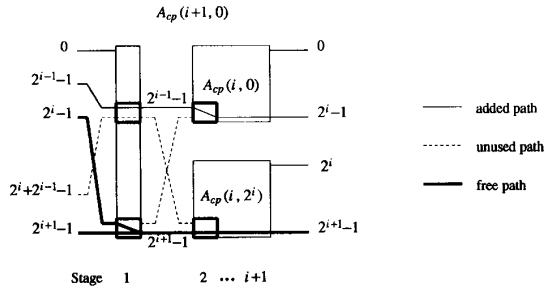


Fig. 16. Embedding of $A_cp(i+1,0)$ in $MIN(i+1,0)$.

already established in an MIN, a path $p \notin E$ is called *free* if it can be established in the MIN without conflicting with any path in E.

We first prove that all edges in $A_{\rm cp}(i,b)$ for any i can be established in one mapping of $MIN(i,b)$ by induction on i.

Induction Hypotheses:

1) All edges in $A_{\rm cp}(i,b)$ can be established in one mapping in $MIN(i,b)$.
2) After establishing all edges in $A_{\rm cp}(i,b)$ in $MIN(i,b)$, the two paths $(2^{i-1}+b-1, 2^i+b-1)$ and $(2^i+b-1, 2^i+b-1)$ are free.

Induction Basis: When $i \le 2$, the above hypotheses can be verified.

Induction Steps: Assuming that the hypotheses are true for $A_{\rm cp}(i,b)$, we want to prove that the hypotheses are also true for $A_{\rm cp}(i+1,b)$.

In Fig. 15(a) and 15(b), free paths are shown in bold lines after all edges in $A_{\rm cp}(i,0)$ are established in $MIN(i,0)$ and all edges in $A_{\rm cp}(i,2^i)$ are established in $MIN(i,2^i)$, respectively. Fig. 16 shows $MIN(i+1,0)$ constructed from these two MIN's and an extra front stage. By setting all switches at the first stage of $MIN(i+1,0)$, except the last one at the bottom, to "straight," all edges in both $A_{\rm cp}(i,0)$ and $A_{\rm cp}(i,2^i)$ can be established without conflicts. In addition, edge $(2^{i-1}-1, 2^i-1)$ in $A_{\rm cp}(i+1,0)$ can be established in $MIN(i+1,0)$ by using the upper free path in $MIN(i,0)$. Therefore, according to (A1), all edges in $A_{\rm cp}(i+1,0)$ are now established in one mapping.

Since neither input port 2^i-1 nor input port $2^{i+1}-1$ is connected to any output port according to $A_{\rm cp}(i+1,0)$, the last switch at the bottom of the first stage can be in either the straight or cross state without affecting any existing path. With the lower free path in $MIN(i,2^i)$, any one of the two input ports 2^i-1 and $2^{i+1}-1$ can be connected to output port $2^{i+1}-1$. That is, the two paths represented by bold lines in Fig.

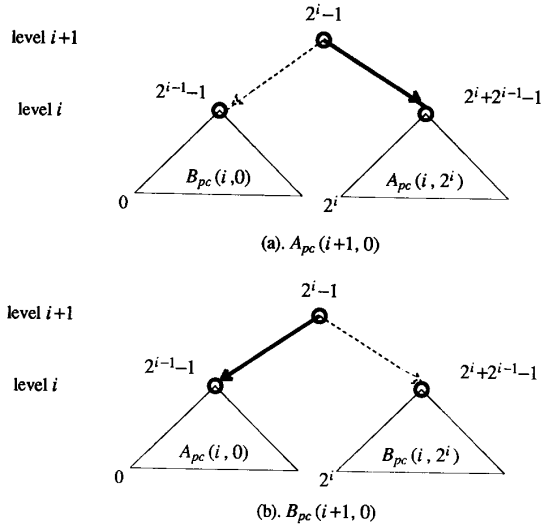


Fig. 17. Recursive constructions of A_pc and B_pc.

16 are free in $MIN(i+1,0)$. This proves that the hypotheses are true for $A_{\rm cp}(i,0)$ for any i.

Note that embedding $A(i+1,b)$ in $MIN(i+1,b)$ is not different from embedding $A(i+1,0)$ in $MIN(i+1,0)$, except that all input and output ports will be offset by b. Therefore, the hypotheses are also true for $A(i+1,b)$. That is, we have proved by induction that all edges in $A(i,b)$ can be established in one mapping of $MIN(i,b)$ for any i.

A similar proof can be used to show that all edges in $B_{\rm cp}(i+1,b)$ can be established in one mapping. Therefore, all edges in $T_{\rm cp}(i+1,0)$ can be established in two mappings for any i, and Theorem 1 holds. □

Proof of Theorem 2: Denote the set of parent-to-children edges in an L-level tree by $T_{\rm pc}(L,b)$ if the leftmost leaf of the tree is numbered b. In order to prove that all of the edges in $T_{\rm pc}$ can be established in two mappings, partition the set $T_{\rm pc}(L,b)$ into two subsets. This partition, however, is somewhat different from the partition used in the proof of Theorem 1. More specifically, $T_{\rm pc}(L,b)$ is partitioned into two subsets, $A_{\rm pc}(L,b)$ and $B_{\rm pc}(L,b)$, which are recursively defined as in (A2). Note that in the proof of Theorem 1, the recursive construction of subset A is independent of that of subset B (see (A1)), which is not the case here.

$$A_{\rm pc}(i+1,b) = B_{\rm pc}(i,b) \cup A_{\rm pc}(i,b+2^i) \cup (2^i+b-1, 2^i+2^{i-1}+b-1) \quad \text{(A3)}$$

$$B_{\rm pc}(i+1,0) = A_{\rm pc}(i,b) \cup B_{\rm pc}(i,b+2^i) \cup (2^i+b-1, 2^{i-1}+b-1) \quad \text{(A4)}$$

where $2 \le i \le L$, $A_{\rm pc}(2,b) = (b+1,b)$, and $B_{\rm pc}(2,b) = (b+1,b+2)$.

Fig. 17 illustrates the recursive construction procedures outlined in (A2) when $b = 0$.

The proof that all edges in $A_{\rm pc}(i,b)$ can be established in one mapping, and that all edges in $B_{\rm pc}(i,b)$ also can, is by induction on i. The induction hypotheses are as follows.

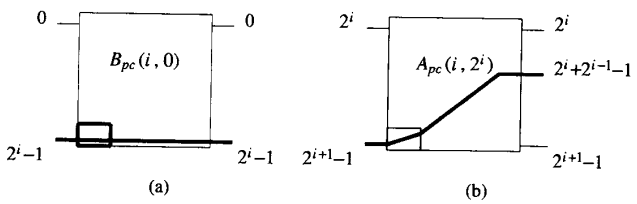


Fig. 18. Embedding of A_pc. (a) $B_pc(i,0)$ in $MIN(i,0)$. (b) $A_pc(i,2^i)$ in $MIN(i,2^i)$.

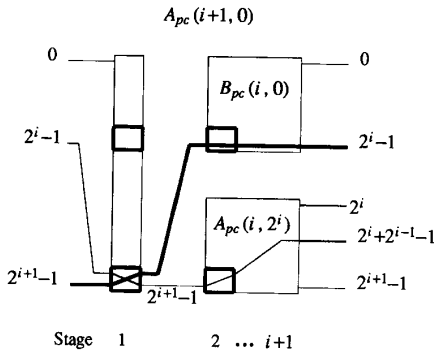


Fig. 19. Embedding of $A_pc(i+1,0)$ in $MIN(i+1,0)$.

1) All edges in $A_{\rm pc}(i,b)$ can be established in $MIN(i,b)$ in one mapping, with path $(b+2^i-1, b+2^{i-1}-1)$ free.
2) All edges in $B_{\rm pc}(i,b)$ can be established in $MIN(i,b)$ in one mapping, with path $(b+2^i-1, b+2^i-1)$ free.

The above hypotheses can be verified when $i \le 2$. The hold line in Fig. 18(a) shows the free path in the $MIN(i,0)$, in which all edges in $B_{\rm pc}(i,0)$ are established. The bold line in Fig. 18(b) shows the free path in the $MIN(i,2^i)$, in which all edges in $A_{\rm pc}(i,2^i)$ are established.

Fig. 19 shows $_{MIN}(i+1,0)$, with the MIN in Fig. 20(a) on top of the MIN in Fig. 18(b) and an extra front stage. By setting all of the switches of at the first stage $MIN(i+1,0)$, except the last one at the bottom, to "straight," all connections in both $B_{\rm pc}(i,0)$ and $A_{\rm pc}(i,2^i)$ can be established. In addition, by using the free path in the MIN of Fig. 18(b), edge $(2^i-1, 2^i+2^{i-1}-1)$ can be established, along with all edges in $B_{\rm pc}(i,0)$ and $A_{\rm pc}(i,2^i)$. That is, all edges in $A_{\rm pc}(i+1,0)$ can be established in one mapping. In addition, the path, denoted by the bold line in Fig. 19, is free. Therefore, induction hypothesis 1) is proved.

Induction hypothesis 2) can be proven in a similar manner. Fig. 20 shows $_{MIN}(i+1,0)$ with a sub-MIN similar to the one in Fig. 20(b) at the top, and a sub-MIN similar to the one in Fig. 18(a) at the bottom. By setting all of the switches of the first stage $MIN(i+1,0)$ to "straight," all connections in both $A_{\rm pc}(i,0)$ and $B_{\rm pc}(i,2^i)$ can be established in the upper and lower sub-MIN's, respectively. In addition, by using the free path in the upper sub-MIN, connection $(2^i-1, 2^{i-1}-1)$ can also be established. That is, all connections in $B_{\rm pc}(i+1,0)$ can be established in one mapping. Because the path, denoted by the bold line in Fig. 20, is free, induction hypothesis 2) is thus true.

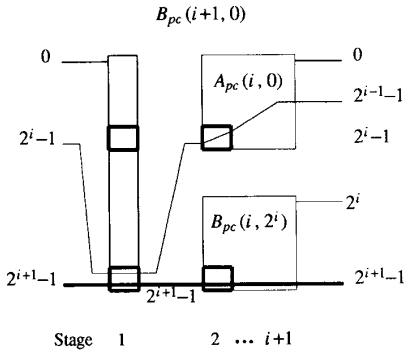


Fig. 20. Embedding of $B_pc(i+1,0)$ in $MIN(i+1,0)$.

Having proved the above two induction hypotheses, it is clear that all edges in $T_{\mathrm{pc}}(i,0)$ can be established in two mappings, and thus Theorem 2 holds. □

REFERENCES

[1] M. Abidi and D. Agrawal, "On conflict-free permutations in multistage interconnection networks," *J. Digital Syst.*, vol. V, no. 2, pp. 115–134, 1980.

[2] K.E. Batcher, "The flip network in STARAN," *1976 Int. Conf. Parallel Processing*, pp. 65–71.

[3] ——, "The multi-dimensional access in STARAN," *IEEE Trans. Comput.*, Special Issue on Parallel Processing, vol. C-27, no. 2, pp. 174–177, Feb. 1977.

[4] P. Bernhard and D. Rosenkrantz, "The complexity of routing through an omega network," *Proc 25th Ann. Allerton Conf. Commun., Control and Computing*, pp. 1027–1036, Sept. 1987. (Also appeared in *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, pp. 503–507, 1991.

[5] P. Bernhard, "Bounds on the performance of message routing heuristics," *Proc. 3rd Symp. Parallel Distrib. Processing*, pp. 856–863, Dec. 1991.

[6] J. Deogun and Z. Fang, "A heuristic algorithm for conflict resolution problem in multistage interconnection networks," *Proc. 1987 Int. Conf. Parallel Processing*, 1987, pp. 475–478.

[7] P. Dowd, "Random access protocols for high speed interprocessor communication based on a passive star topology," *J. Lightwave Technol.*, vol. LT-5, no. 12, pp. 1782–1793, Dec. 1987.

[8] T. Y. Feng, "Data manipulating functions in parallel processors and their implementations," *IEEE Trans. Comput.*, vol. C-23, no. 3, pp. 309–318, Mar. 1974.

[9] A. Ganz and Z. Koren, "WDM passive star protocols and performance analysis," *Proc. IEEE INFOCOM'91,*, 1991, pp. 9A.2.1–9A.2.10.

[10] G.R. Goke and G.J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *1st Ann. Symp. Comput. Architecture*, 1973, pp. 21–28.

[11] Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, "Array processors with pipelined optical busses," *J. Parallel Distrib. Computing*, vol. 12, pp. 269–282, 1991.

[12] C. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Trans. Comput.*, vol. C-32, no. 12, pp. 1091–1098, Dec. 1983.

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

[14] J. Lenfant, "Parallel permutations of data: A Benes network control algorithm for frequently used permutations," *IEEE Trans. Comput.*, vol. C-27, no. 7, pp. 637–647, July 1978.

[15] G. Lev, N. Pippenger, and L.G. Valiant, "A fast parallel algorithm for routing in permutation networks," *IEEE Trans. Comput.*, vol. C-30, no. 2, pp. 93–100, 1981.

[16] W. Lin and C.-L. Wu, "Configuring computation tree topologies on a distributed computing system," *Proc. Int. Conf. Parallel Processing*, 1983, pp. 114–116.

[17] R. Melhem, D. Chiarulli, and S. Levitan, "Space multiplexing of waveguides in optically inter-connected multiprocessor systems," *Comput. J.*, vol. 32, no. 4, pp. 362–369, 1989.

[18] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permutation network," *IEEE Trans. Comput.*, vol. C-31, no. 2, pp. 148–154, Feb. 1982.

[19] D. Opferman and N. Taou-Wu, "On a class of rearrangeable switching networks," *Bell Syst. Tech. J.*, vol. 50, pp. 1579–1600, May 1971.

[20] M.C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, no. 5, pp. 458–473, May 1987.

[21] U. Premkumar and J. Browne, "Resource allocation in rectangular SW Banyan," *9th Ann. Int. Symp. Comput. Architecture*, pp. 326–333, 1982.

[22] C. Qiao, "A high speed interconnection paradigm for multiprocessors and its applications to optical interconnection networks," Ph.D. dissertation (in preparation), Department of Comput. Sci., Univ. of Pittsburgh.

[23] C. Qiao and R. Melhem, "Time-division optical communications in multiprocessor arrays," *IEEE Trans. Comput.,*, vol. 42, pp. 577–590, May 1993.

[24] C. Qiao, R. Melhem, D. Chiarulli, and S. Levitan, "Dynamic reconfiguration of optically interconnected networks with time division multiplexing," *J. Parallel Distrib. Computing*, to appear.

[25] C. Raghavendra and A. Varma, "Fault-tolerant multiprocessors with redundant-path interconnection networks," *IEEE Trans. Comput.*, vol. C-35, no. 4, pp. 307–316, Apr. 1986.

[26] H. Ramanujam, "Decomposition of permutation networks," *IEEE Trans. Comput.*, vol. 22, no. 7, pp. 639–643, July 1973.

[27] D. Sarrazin, H. Jordan, and V. Heuring, "Digital fiber optic delay line memory," *Digital Optical Computing II, SPIE*, vol. 1215, pp. 366–375, Jan. 1990.

[28] J. Sauer, "A multi-Gb/s optical interconnect," *SPIE Proc., Digital Computing II*, vol. 1215, pp. 198–207, 1990.

[29] A. Shimer and S. Ruhman, "Toward a generalization of two- and three-pass multistage blocking interconnection networks," *Proc. Int. Conf. Parallel Processing*, 1980, pp. 337–346.

[30] H.J. Siegel and S.D. Smith, "Study of multistage SIMD interconnection networks," *5th Ann. Symp. Comput. Architecture*, 1978, pp. 223–229.

[31] A. Somani and S. Choi, "On embedding permutations in hypercubes," *Proc. 1991 Distrib. Memory Computing Conf.* 1991, pp. 622–629.

[32] H.S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, Feb. 1971.

[33] S. Thanawastien and V. Nelson, "Interference analysis of shuffle/exchange networks," *IEEE Trans. Comput.*, vol. C-30, no. 8, pp. 545–556, Aug. 1981.

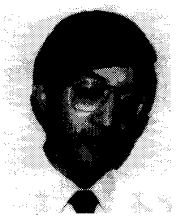
[34] R.A. Thompson, "Photonic time-multiplexed permutation switching using the dilated slipped banyan network," *Proc. OSA Topical Meeting on Photon. Switching 1989*, 1991, pp. 64–66.

[35] N. Tzeng and S. Wei, "Enhanced hypercubes," *IEEE Trans. Comput.*, vol. 40, pp. 284–294, Mar. 1991.

[36] C.-L Wu and T.Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-29, no. 8, pp. 694–702, Aug. 1980.

[37] ——, "The universality of the shuffle-exchange network," *IEEE Trans. Comput.*, vol. C-30, no. 5, pp. 324–332, May 1981.

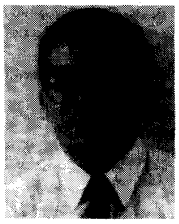
[38] A. Youssef, "Off-line permutation scheduling on circuit-switched fixed routing networks," *Proc. 4th Symp. Frontiers of Massively Parallel Computation*, 1992, pp. 389–396.



Chunming Qiao received the B.S.E.E. degree in computer science and engineering from the University of Science and Technology of China in 1985, and the M.S. and Ph.D. degrees in computer science from the University of Pittsburgh in 1990 and 1993, respectively.

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, State University of New York, Buffalo, NY. His research interests include parallel computer architectures, multiprocessor interconnection networks, photonic switching, and optical communications networks.

Dr. Qiao received the Andrew Mellon Predoctoral Award during 1992–93. He served in the Program Committee for the 27th Annual Simulation Symposium. He is a member of the International Society for Optical Engineering (SPIE) and the IEEE Computer Society.



Rami G. Melhem received a B.E. degree in electrical engineering from Cairo University, Egypt, in 1976; an M.A. degree in mathematics and an M.S. degree in computer science from the University of Pittsburgh in 1981; and a Ph.D. degree in computer science from the University of Pittsburgh in December 1983.

Since 1989, he has been an Associate Professor of computer science at the University of Pittsburgh. Previously, he was an Assistant Professor at Purdue University and at the University of Pittsburgh. He has published numerous papers in the areas of systolic architectures, parallel computing, fault-tolerant processor arrays, and optical computing.

Dr. Melhem has served in program committees for several conferences, and he is on the Editorial Board of the IEEE TRANSACTIONS ON COMPUTERS. He is a member of the IEEE Computer Society, the Association for Computing Machinery, and the International Society for Optical Engineering.