The Journal of Supercomputing, 1, 13–42 (1987)

An Introduction to the Multistage Cube Family of Interconnection Networks

HOWARD JAY SIEGEL*, WILLIAM TSUN-YUK HSU**, MENKAE JENG
PASM Parallel Processing Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907, USA

Abstract. The interconnection network in large-scale parallel/distributed supercomputer systems is a crucial component. Three networks are overviewed here. Multistage cube networks represent an important family of networks, which includes the omega, n-cube, multistage shuffle-exchange, delta, baseline, SW-banyan, and Generalized Cube. This family has been used or proposed for use in such systems as STARAN, PASM, Ultracomputer, the BBN Butterfly, the IBM RP3, and data-flow machines. The multistage cube topology, distributed routing control, and ability to be partitioned into independent subnetworks are examined. The Extra Stage Cube (ESC), a single-fault-tolerant multistage cube network, is described. The structure, control, and partitionability of the ESC, and how it functions when multiple faults occur, are presented. The Dynamic Redundancy (DR) network, a fault-tolerant multistage cube network that supports the incorporation of spare processors for fault tolerance, is discussed. Its structure, control, and partitionability into single-fault-tolerant subnetworks are explained.

1. Introduction

There is a myriad of tasks that require the computational power made possible by parallel processing. The demand for fast computation is usually due to a desire for real-time response and/or the need to process immense data sets. These types of tasks include weather forecasting, map making, aerodynamic simulations, chemical reaction simulations, seismic data processing, air traffic control, satellite-collected imagery analysis, missile guidance, ballistic missile defense, robot vision, and speech understanding. In all of these situations, systems comprising a multitude of tightly coupled, cooperating processors working together to perform a single overall task can supply the support essential to meet the computational performance goals. This article overviews one type of interconnection network that can provide communications among the processors of such systems: the multistage cube.

The task of interconnecting N processors and N memory modules, where N may be in the range 2^6–2^{16}, is a nontrivial one. The interconnection scheme must

This research was supported by the Air Force Office of Scientific Research under grant F49620-86-K-0006, the Rome Air Development Center under grant F30602-83-K-0119, and the Purdue Research Foundation David Ross Grant 1985/86 no. 0857.
* currently with the Supercomputing Research Center, 4380 Forbes Blvd., Lanham, MD 20706 (as of June 1, 1987).
** currently with Computer Science Department, University of Illinois, Urbana-Champaign, IL 61801.

provide fast and flexible communications without unreasonable cost. A single shared bus is not sufficient, since it is often desirable to allow all processors to send data to other processors simultaneously (e.g., from processor i to processor $i+1$, $0 \leq i < N-1$). Ideally, one would like to be able to link directly each processor to every other processor so that the system is completely connected. Unfortunately, this is highly impractical for large N, since it requires $N-1$ lines for each processor. For example, if $N = 2^{10}$, then $2^{10} * (2^{10} - 1) = 1{,}047{,}552$ links would be needed. An alternative interconnection scheme that allows all processors to communicate simultaneously is the crossbar switch. The difficulty with crossbar switches is that N^2 crosspoint switches are needed [Thurber 1979]. Thus, the cost of the network grows with N^2, which, given current technology, makes it infeasible for large systems.

In order to solve the problem of providing fast, efficient communications at a reasonable cost, many different networks between the extremes of the single bus and the completely connected scheme have been proposed in the literature. There is no single network that is generally considered "best." The cost-effectiveness of a particular network design depends on such factors as the computational tasks for which it will be used, the desired speed of interprocessor data transfers, the actual hardware implementation of the network, the number of processors in the system, and any cost constraints on the construction. One network approach that appears to be appealing for a variety of parallel systems is the multistage cube. The advantages of the multistage cube network approach include hardware complexity proportional to $N \log_2 N$, distributed control schemes, partitionability, and availability of multiple simultaneous paths. The multistage cube network family is the topic of this article. Other approaches are described by Broomell and Heath [1983], Feng [1981], Masson et al. [1979], Siegel et al. [1979], and Thurber and Masson [1979].

Multistage cube networks have been used or proposed for use in a number of parallel supercomputers, including STARAN [Batcher 1977], DISP [Filip 1982], PASM [Siegel et al. 1981 and 1984], PUMPS [Briggs et al. 1982], the Ballistic Missile Defense Agency test bed [McDonald and Williams 1978; Siegel and McMillen 1981], Ultracomputer [Gottlieb et al. 1983], the BBN Butterfly [Crowther et al. 1985], the Burroughs Flow Model Processor for the Numerical Aerodynamic Simulator [Barnes and Lundstrom 1981], the IBM Research Parallel Processor Prototype (RP3) [Pfister et al. 1985], and data-flow machines [Dennis et al. 1980]. Multistage cube-type networks have also been called baseline [Wu and Feng 1980], delta ($a = b = 2$) [Patel 1981], Generalized Cube (Siegel and McMillen 1981; Siegel and Smith 1978], indirect binary n-cube [Pease 1977], omega [Lawrie 1975], multistage shuffle-exchange [Thanawastien and Nelson 1981], flip [Batcher 1976], and SW-banyan ($S = F = 2$) [Goke and Lipovski, 1973]. The Generalized Cube is representative of these networks in that they are topologically equivalent to it [Siegel 1979 and 1985; Siegel and Smith 1978; Wu and Feng 1980]. This article describes various properties of the Generalized Cube network.

Different approaches to making multistage networks fault tolerant are discussed

by Adams et al. [1987] and Jeng and Siegel [1986a]. Here we overview two particular fault-tolerant multistage cube variations: the Extra Stage Cube and the Dynamic Redundancy networks. In a reconfigurable machine, in order to attempt to optimize the performace of the system, the operating system must be able to control efficiently the network path establishment and network partitioning [Delp et al. 1985]. The issues of network control, partitioning, and fault recovery are addressed in the following sections.

In section 2, basic parallel machine organizations are defined. Section 3 overviews the Generalized Cube multistage network and its properties. The Extra Stage Cube and Dynamic Redundancy networks are described in sections 4 and 5, respectively.

2. Parallel Machine Organizations

One approach to the design of supercomputer systems is to use a large number of homogeneous processors to execute a task in a cooperative manner. This section describes some basic models for such parallel systems.

The acronym SIMD stands for *single instruction stream—multiple data stream* [Flynn 1966]. Typically, SIMD machine is a computer system consisting of a control unit, N processors, N memory modules, and an interconnection network. The control unit broadcasts instructions to the processors, and all active processors execute the same instruction at the same time. Thus, there is a single instruction stream. Each active processor executes the instruction on data in its own associated memory module. Thus, there are multiple data streams. The interconnection network, sometimes referred to as an alignment or permutation network, provides for communications among the processors and memory modules. These machines are designed to exploit the parallelism of tasks such as matrix operations and problem domains such as image processing, where the same operation is performed on many different matrix or image elements. General information about SIMD systems can be found in textbooks such as those by Baer [1980], Hockney and Jesshope [1981], Hwang and Briggs [1984], Stone [1980], and Thurber [1976]. Examples of SIMD machines that have been constructed include STARAN [Batcher 1974], MPP [Batcher 1980 and 1982], and the Connection Machine [Hillis 1985].

A multiple-SIMD machine is a parallel processing system that can be dynamically reconfigured to operate as one or more independent virtual SIMD machines of various sizes. A multiple-SIMD system consists of N processors, N memory modules, an interconnection network, and Q control units, where $Q < N$. Each of the multiple control units can be connected to some unique subset of the processors, creating independent virtual SIMD machines of various sizes. An example of a proposed system capable of operating in the multiple-SIMD mode is MAP [Nutt 1977a and b].

The acronym MIMD stands for *multiple instruction stream—multiple data stream*

[Flynn 1966]. Typically, an MIMD machine consists of N processors, N memory modules, and an interconnection network. Each of the N processors follows its own program. Thus, there are multiple instruction streams. Each processor fetches its own data on which to operate. Thus, there are multiple data streams, as in an SIMD system. The interconnection network provides communications among the processors and memory modules. While in an SIMD system all active processors use the interconnection network at the same time (i.e., synchronously), in an MIMD system, since each processor is executing its own program, inputs to the network arrive independently (i.e., asynchronously). Examples of large MIMD systems that have been constructed are Cm* [Jones et al. 1977; Swan et al. 1977a and b], the BBN Butterfly [Crowther et al. 1985], and the Intel iPSC cube [Intel Corporation 1985]. More information about MIMD systems and their use can be found in textbooks such as those by Baer [1980], Hwang and Briggs [1984], and Stone [1980].

A partitionable SIMD/MIMD machine is a parallel processing system that can be dynamically reconfigured to operate as one or more independent virtual SIMD and/or MIMD machines of various sizes. A partitionable SIMD/MIMD machine has the same components as a multiple-SIMD machine except that each processor can follow its own instructions (MIMD operation) in addition to being capable of accepting an instruction stream from a control unit (SIMD operation). Like multiple-SIMD machines, the N processors, N memory modules, and interconnection network of a partitionable SIMD/MIMD system can be partitioned to form independent virtual machines. In this case, each virtual machine can be an SIMD machine or an MIMD machine. Examples of proposed partitionable SIMD/MIMD systems are PASM [Siegel et al. 1978, 1981, 1984] and TRAC [Kapur et al. 1980; Premkumar 1980; Sejnowski et al. 1980].

The multistage cube networks discussed here can support partitionable SIMD/MIMD systems as well as the SIMD, multiple-SIMD, and MIMD modes of parallelism. This flexibility is important since it permits the construction of a partitionable SIMD/MIMD system with properties that include the following [Siegel et al. 1981]:

1. Fault tolerance. If a single processor fails, only those virtual machines (partitions) that include the failed processor are affected. The rest of the system can continue to function.
2. Fault detection. For situations in which high reliability is needed, three partitions can run the same program on the same data and compare results.
3. Multiple simultaneous users. Since there can be multiple independent virtual parallel machines, there can be multiple simultaneous users of the system, each executing a different parallel program.
4. Program development. Rather than trying to debug a parallel program on, for example, 1024 processors, a user can debug the program on a smaller-size virtual parallel machine of 32 or 64 processors.
5. Efficiency. If a task requires only $N/2$ of N available processors, the other $N/2$ can be used for another task.
6. Subtask parallelism. Two or more independent parallel subtasks that are part

of the same job can be executed in parallel, sharing results if necessary.

7. Mode selection. The same set of processors can switch between the SIMD and MIMD modes of parallelism when performing a task, depending on which is more desirable at the time.

Given a multistage network, there are two basic multiprocessor system configurations [Siegel 1985]. One is called the *PE-to-PE configuration*, in which each processing element, or *PE* (formed by a processor and a local memory), is attached to both an input port and an output of the network (i.e., PE j is connected to input port j and output port j). The inter-PE communication provided by the network is unidirectional from the input side to the output side. The other configuration is called the *processor-to-memory configuration*, in which processors are attached to one side of the network while memories are attached to the other side. The communication through the network is bidirectional. Moving data between two processors cannot be done by one transfer; data must be moved through shared memories. The PE-to-PE system configuration will be used in the following discussions; however, the results presented will also be applicable to the processor-to-memory configuration.

3. Generalized Cube

3.1. Network Definition

The *Generalized Cube* network topology is shown in figure 1. Assume the network has N inputs and N outputs. In figure 1, $N = 8$. The Generalized Cube topology has $m = \log_2 N$ stages, where each stage consists of a set of N lines (*links*) connected to $N/2$ interchange boxes. Each *interchange box* is a two-input, two-output device. The labels of the links entering the upper and lower inputs of an

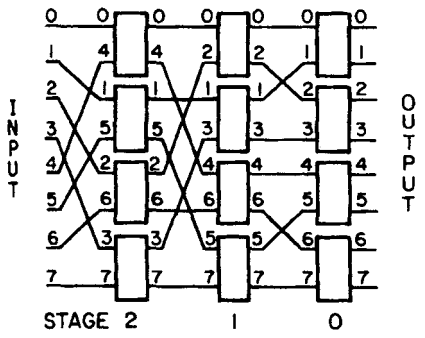

Figure 1. Generalized Cube network topology for $N = 8$.

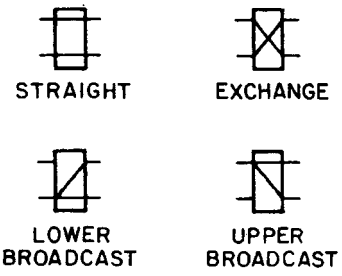

Figure 2. The four legitimate states of an interchange box.

interchange box are used as the labels for the upper and lower outputs, respectively. The labels are the integers from 0 to $N-1$.

The connections in this network are based on the Cube interconnection functions [Siegel 1977 and 1985]. Let $P = p_{m-1} \ldots p_1 p_0$ be the binary representation of an arbitrary link label. Then the m Cube interconnection functions can be defined as:

$$\text{cube}_i(p_{m-1} \ldots p_1 p_0) = p_{m-1} \ldots p_{i+1} \bar{p}_i p_{i-1} \ldots p_1 p_0$$

where $0 \leq i < m$, $0 \leq P < N$, and $\bar{p}_i$ denotes the complement of p_i. This means that the cube_i interconnection function connects link P to link $\text{cube}_i(P)$, where $\text{cube}_i(P)$ is the link whose label differs from P in just the ith bit position. Stage i of the Generalized Cube network topology contains the cube_i interconnection function; that is, it pairs links that differ in the ith bit position.

Each interchange box can be set to one of four legitimate states. Let the upper input and output links be labeled J and the lower input and output links be labeled K. The four legitimate states (see figure 2) are (1) *straight*—input J to output J, input K to output K; (2) *exchange*—input J to output K, input K to output J; (3) *lower broadcast*—input K to outputs J and K; and (4) *upper broadcast*—input J to outputs J and K [Lawrie 1975]. The straight and exchange settings produce a one-to-one connection at the box, while the lower and upper broadcasts produce a one-to-two connection at the box. A *two-function* interchange box can be in either the straight or exchange state and a *four-function* interchange box can be in any of the four states [Siegel and Smith 1978].

The name *Generalized Cube network* will be used to refer to the network consisting of the Generalized Cube topology and four-function interchange boxes. Furthermore, each interchange box will be controlled independently through the use of routing tags, as will be discussed. The properties of the Generalized Cube make it well-suited for SIMD, MIMD, multiple-SIMD and partitionable SIMD/MIMD operations.

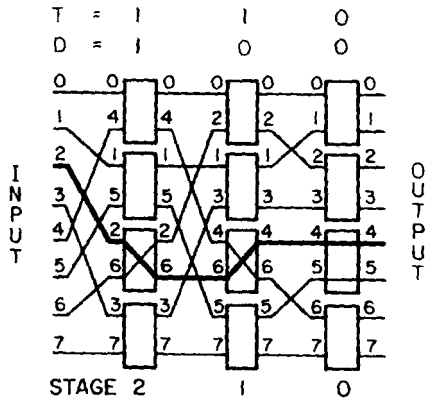

Figure 3. The path from input 2 to output 4 in the Generalized Cube network for $N = 8$. "T" is the exclusive-or tag. "D" is the destination address tag.

3.2. *One-to-One Connections*

One-to-one connections in the Generalized Cube network involve just the straight and exchange box settings. Recall that the two input/output links to an interchange box in stage i differ only in the ith bit position; that is, J and K in the box description in the previous subsection differ only in the ith bit position. In general, to go from a source $S = s_{m-1} \ldots s_1 s_0$ to a destination $D = d_{m-1} \ldots d_1 d_0$, the stage i box in the path from S to D must be set to exchange (cube$_i$) if $s_i \neq d_i$ and to straight if $s_i = d_i$. For example, if $S = 2$ nd $D = 4$, then $s_2 s_1 s_0 = 010$ and $d_2 d_1 d_0 = 100$, so $s_2 \neq d_2$, $s_1 \neq d_1$, and $s_0 = d_0$. This is shown in figure 3. Thus, stage i determines d_i: it is $\bar{s}_i$ if the box is set to exchange and it is s_i if the box is set to straight.

Because stage i determines the value of d_i (i.e., if it will be $\bar{s}_i$ or s_i), d_{m-1} is first determined, then d_{m-2}, then d_{m-3}, etc. The link a message from S to D travels on is therefore $s_{m-1} \ldots s_1 s_0$ before stage $m-1$, $d_{m-1} s_{m-2} \ldots s_1 s_0$ after stage $m-1$, $d_{m-1} d_{m-2} s_{m-3} \ldots s_1 s_0$ after stage $m-2$, $\ldots$, $d_{m-1} \ldots d_{i+1} d_i s_{i-1} \ldots s_1 s_0$ after stage i, $\ldots$, and finally $d_{m-1} \ldots d_1 d_0$ after stage 0. For the example in figure 3, where $N = 8$ ($m = 3$), $S = 2$, and $D = 4$, the links traversed are $s_2 s_1 s_0 = 010$ at the input, $d_2 s_1 s_0 = 110$ after stage 2, $d_2 d_1 s_0 = 100$ after stage 1, and $d_2 d_1 d_0 = 100$ after stage 0. In general, the stage i output link used in the path from S to D is $d_{m-1} \ldots d_{i+1} d_i s_{i-1} \ldots s_1 s_0$.

A property of the Generalized Cube network that follows from the above discussion is that there is only one path from a given source to a given destination. For example, there is only one way to go from input 2 to output 4 in figure 3. Only stage i can change the ith bit of the link label in the path from S to D. After

stage 0, this link label must be D. Therefore, a message from S must take the output link $d_{m-1}s_{m-2}\ldots s_1 s_0$ at stage $m-1$, the output link $d_{m-1}d_{m-2}s_{m-3}\ldots s_1 s_0$ at stage $m-2$, etc. This is the only path available from S to D.

A routing tag scheme can be used for controlling the Generalized Cube network. Using tags allows network control to be distributed among the processors. The routing tags for one-to-one data transfers (not involving broadcasting) consist of m bits. When a message is to be transmitted through the network, a routing tag is added as the first item in the message, i.e., the header. Each interchange box that the message enters examines the routing tag to determine how it should be set and then sets itself. The data portion of the message follows the routing tag portion through the network.

The use of routing tags is appropriate for operating the network in either the circuit-switched mode or the packet-switched mode. In the *circuit-switched mode*, once a path is established by the routing tag, the interchange boxes in the path remain in their specified state until the path is released. Thus, there is a complete circuit established, from input port to output port, for that path. The data are then sent directly from the source to the destination over this circuit. In the *packet-swtiched mode*, the routing tag and data to be transmitted are collected together into a "packet." As in the circuit-switched case, the routing tag sets the state of the interchange box. However, in contrast to the circuit-switched case, a complete path (circuit) from the source to the destination is not established. Instead, the packet makes its way from stage to stage, releasing links and interchange boxes immediately after using them. In this way, only one interchange box is used at a time for each message. This differs from circuit switching, where m boxes, one from each stage, are used simultaneously for the entire duration of the message transmission. Circuit switching must be used in networks constructed from combinational logic where there are no buffers in the interchange boxes for storing data. Packet switching uses data buffers in each interchange box to store packets as they move through the network. Details of the communications protocol and interchange box physical implementation are discussed by McMillen et al. [1981], and McMillen and Siegel [1980], and Siegel and McMillen [1981].

Exclusive-or tags. For one-to-one (nonbroadcast) connections, an m-bit routing tag can be computed from the input port number and desired output port number. Let $S = s_{m-1}\ldots s_1 s_0$ be the source network address (input port number) and $D = d_{m-1}\ldots d_1 d_0$ be the destination network address (output port number). Then the routing tag $T = t_{m-1}\ldots t_1 t_0 = S \oplus D$ (where "$\oplus$" means bitwise "exclusive-or") [Siegel and McMillen 1981]. This is called the *exclusive-or routing tag* scheme. An interchange box in the network at stage i need only examine t_i. If $t_i = 1$, an exchange is performed (i.e., cube$_i$ is performed), since $t_i = 1$ implies $d_i = \bar{s}_i$. If $t_i = 0$, the straight connection is used, since $t_i = 0$ implies $d_i = s_i$. If $N = 8$, $S = 2 = 010$, and $D = 4 = 100$, then $T = 110$. The corresponding stage settings are exchange, exchange, and straight. This is shown in figure 3.

Because the exclusive-or operation is commutative (i.e., $S \oplus D = D \oplus S$), the incoming routing tag is the same as the return tag. In other words, the same tag used to route data from S to D can be used to route data from D to S. This can be used to "handshake" (confirm receipt of the data).

If each network destination D knows its own address (output port number), then from the tag it can compute the source address S (input port number) that sent it data. Specifically, $T \oplus D = S$.

Destination tags. As an alternative to the exclusive-or routing tag method, destination tags can be used [Lawrie 1975]. In the *destination tag* scheme, the destination $D = d_{n-1} \ldots d_1 d_0$ is the tag. An interchange box in the network at stage i need only examine d_i. If $d_i = 0$, the upper output of the box is taken. If $d_i = 1$, the lower output of the box is taken. As an example, consider the path from source 2 to destination 4 shown in figure 3. The lower output of the stage 2 box is taken ($d_2 = 1$), the upper output of the stage 1 box is taken ($d_1 = 0$), and the upper output of the stage 0 box is taken ($d_0 = 0$). This same destination to tag ($D = 4$) would be used to route any input to output 4.

This method differs from the exclusive-or method in that the source address is not used in its calculation and that the output link (upper or lower) is explicitly specified, not the box state (straight or exchange). If the tag arrives on the upper input of a stage i box, then if $d_i = 0$ the box is set to straight and if $d_i = 1$ it is set to exchange. Analogously, if the tag arrives on the lower input box, then if $d_i = 0$ the box is set to exchange and if $d_i = 1$ the box is set to straight.

The reason the destination tag scheme works is because the upper output link of a stage i box always has a 0 in the ith bit position of its label, while the lower output always has a 1. Thus, taking the upper output link means that the ith bit of the network output reached will be a 0, while taking the lower output link means it will be a 1.

This scheme has an advantage over the exclusive-or method in that a destination can compare the destination tag that arrives with its own address to determine whether the message arrived at the correct network output (if it did not, the network must be faulty). It has the disadvantage that it cannot be used to determine the source, as the exclusive-or scheme can. By using m additional bits, both methods can have both the capability to determine whether the data arrived at the proper destination and to identify its source. This can be done by sending the source address along with the destination tag or sending the destination address along with the exclusive-or tag.

Network conflicts. In MIMD mode, for both schemes of routing tags, since each source generates its own tag, it is possible that a conflict can occur in the network; for example, the tag on the upper input link of a box indicates that the box should be set to exchange while the tag on the lower input indicates it should be set to straight. When a situation like this arises, one message must wait until the other has com-

pleted its transmission. Both requests cannot be accommodated simultaneously.

Permutations. Permutation connections, where each input is connected to a single distinct output, are also possible. Since each connection in a permutation is one–to–one, only the straight and exchange box settings are used. Permutations are a special case of one-to-one connections where all N PEs are forming one-to-one connections simultaneously. Such connections are used when processing in SIMD mode. An example of the Generalized Cube network set to connect input j to output $j+1$ modulo N, $0 \le j < N$, for $N = 8$ is shown in figure 4.

As in MIMD mode, network conflicts can occur if a permutation is attempted that results in two box inputs needing to connect to the same box output. For $N = 8$, for example, a permutation that includes connecting input 0 to output 1 and input 4 to output 2 will result in a conflict in the stage 2 box with input labels 0 and 4. Since each of the $Nm/2$ boxes can be set to straight or exchange, the Generalized Cube can do $2^{Nm/2}$ of the $N!$ different permutations. In general, $2^{Nm/2} \ll N!$, for example, for $N = 8$, $2^{Nm/2} = 4096$, while $N! = 40{,}320$. However, it can do most permutations that are important in SIMD processing [Lawrie 1975; Pease 1977].

In the Generalized Cube network, any permutation physically performable by the network can be generated by routing tags. Consider the "+1" permutation shown for $N = 8$ in figure 4. For example, the path from $S = 000$ to $D = 001$ is generated by $T = S \oplus D = 001$ (straight, straight, exchange), and the path from $S = 001$ to $D = 010$ is generated by $T = S \oplus D = 011$ (straight, exchange, exchange). Thus, exclusive-or routing tags can be used for permutations as well as one-to-one connections. Destination tags could also be used.

3.3. *Broadcast Connections*

When the lower and/or upper broadcast states of interchange boxes are used in a

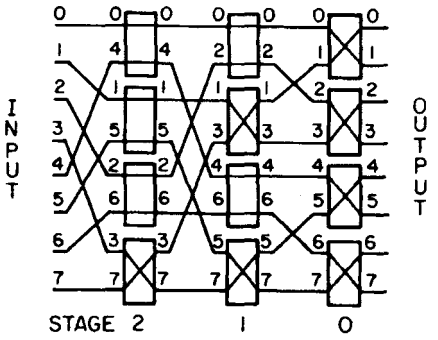

Figure 4. The Generalized Cube for $N = 8$ set to do the permutation input j to output $j+1$ modulo N, $0 \le j < N$.

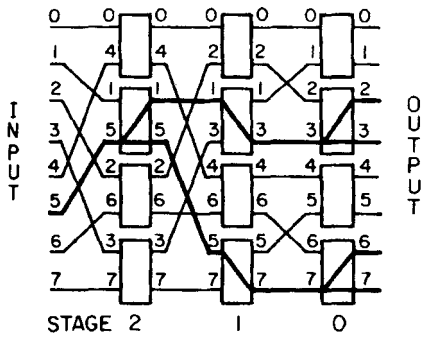

Figure 5. The broadcast path from input 5 to outputs 2, 3, 6, and 7 in the Generalized Cube network for $N = 8$.

path, a broadcast (one-to-many) connection is performed. An example is shown in figure 5, where input 5 broadcasts to outputs 2, 3, 6, and 7. The path from S to a set of destinations D^* is the combination of the paths from S to each destination D in D^*.

A broadcast routing tag scheme that consists of an m-bit broadcast mask along with the m-bit routing tag can be used to specify a variety of broadcast connections (but not all those physically possible). Details are in Siegel [1985] and Siegel and McMillen [1981].

3.4. Partitioning the Generalized Cube Network

The *partitionability* of an interconnection network is the ability to divide the network into independent subnetworks of different sizes. Each subnetwork of size N' must have all of the interconnection capabilities of a complete network of that same type built to be of size N'. A partitionable network can be characterized by any limitations on the way in which it can be subdivided. The partitionability of an interconnection network is an important attribute to consider when choosing a network for a reconfigurable system. The rules for partitioning the Generalized Cube network are from Siegel [1980 and 1985], Siegel and Smith [1978], and Smith and Siegel [1978]. These rules assume a PE-to-PE configuration, where PE i is connected to both input i and output i, so both input i and output i must belong to the same partition. Such partitions will also support the processor-to-memory configuration. (There are some partitionings that will support the processor-to-memory configuration but not the PE-to-PE.)

The key to partitioning the Generalized Cube network so that each subnetwork is independent is the choice of the input/output ports that belong to the subnetworks. The requirement is that the physical addresses of all of the input/output ports in a subnetwork of size 2^s agree (have the same values) in $m - s$ of their bit

positions. Furthermore, the interchange boxes used by this subnetwork are set to straight in the $m-s$ stages corresponding to the $m-s$ bit positions in which the addresses agree. The other s stages are used by the subnetwork to form a Generalized Cube network of size 2^s.

First consider partitioning a Generalized Cube of size N into two independent subnetworks, each of size $N/2$. There are m ways to do this, each being based on a different bit position of the input/output port addresses. One way is to force all boxes in stage $m-1$ to the straight state. This would form two subnetworks, one consisting of those input/output ports with a 0 in the high-order bit position of their addresses and the other consisting of those ports with a 1 in the high-order bit position. These two groups could communicate with each other only by using the exchange setting in stage $m-1$. By forcing this stage to straight, the subnetworks are independent and have full use of the rest of the network (stages $m-2$ to 0). This is shown for $N=8$ in figure 6.

Each subnetwork has the properties of a Generalized Cube. Therefore, each subnetwork can be further subdivided. Referring to figure 6, subnetwork B can be further divided using the middle bit position, forming two smaller subgroups of two processors each, as shown in figure 7. Thus, a size $N/2$ subnetwork can be divided into two size $N/4$ subnetworks by setting all the stage i boxes in the size $N/2$ subnetwork to straight, for any i, $0 \le i < m$, as long as stage i was not used to create the size $N/2$ subnetworks (as stage $m-1$ was above). This process of dividing subnetworks into independent halves can be repeated to create any size subnetwork from one to $N/2$. The only constraints are that the size of each subnetwork must be a power of 2, the physical addresses of the input/output ports of a subnetwork of size 2^s must all agree in any fixed set of $m-s$-bit positions, and each input/output port can belong to at most one subnetwork.

The routing tag control described in the previous subsection can be used within partitions, in combination with an m-bit partitioning mask. For example, consider using the partitioning mask with the exclusive-or tag scheme. The mask is set to 0

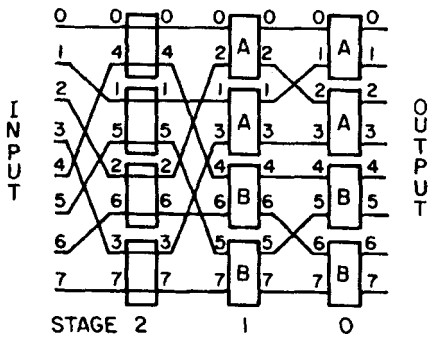

Figure 6. Partitioning the Generalized Cube for $N \times 8$ into two subnetworks of size four (A and B) based on the high-order bit position.

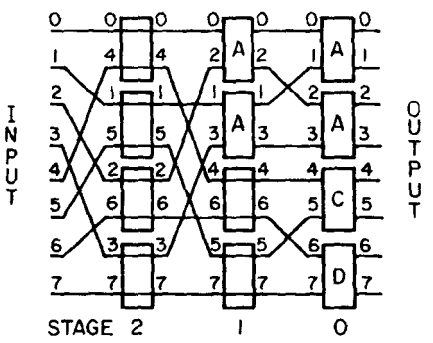

Figure 7. Partitioning the Generalized Cube for $N = 8$ into one subnetwork of size four (A) and two subnetworks of size two (C and D).

in all the bits that correspond to the stages which are forced to straight in that partition, and all other bits are set to 1. The routing tag is logically ANDed with the mask to force corresponding stages to straight.

4. The Extra Stage Cube Network

4.1. Network Definition

The main problem with the Generalized Cube network topology is that there is

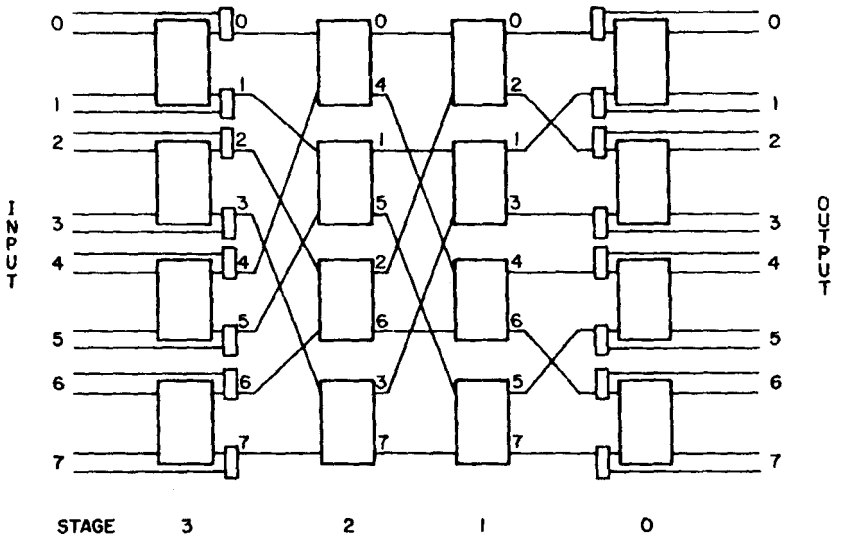

Figure 8. The Extra Stage Cube (ESC) network for $N = 8$.

only one path from a given network input to a given output. Thus, if there is a fault on that path no communication is possible between that input and output.

This section describes the *Extra Stage Cube* (ESC) network, a single-fault-tolerant network derived from the Generalized Cube network, capable of operating in SIMD, multiple-SIMD, MIMD, and partitionable SIMD/MIMD environments [Adams and Siegel 1982 and 1984; Siegel 1985]. The ESC network is formed from the Generalized Cube by adding one extra stage at the input and hardware to allow the bypass, when desired, of the extra stage (stage m) or the output stage (stage 0). Stage m implements the cube_0 interconnection function (pairs links that differ in the 0-th bit position), so there are now two ways of performing the cube_0 data transfer. This results in an additional path being available from each source to each destination. An ESC network for $N = 8$ PEs is shown in figure 8.

A stage is *enabled* when its interchange boxes can be used to provide cube_0 interconnection capability. It is *disabled* when its interchange boxes are being bypassed. Enabling and disabling of stages m and 0 is accomplished by having dual input/output ports, and multiplexers and demultiplexers to select between the input/output lines, as shown in figure 9. At stage m, a multiplexer selects between two sets of identical processor output lines, one of which bypasses the stage m box and the other of which routes through the box. At stage 0, a demultiplexer gives the option of bypassing the box or routing data through it. If dual input/output ports for the processors are not available, it is possible to replace them with demultiplexers at the inputs of the network and multiplexers at the outputs of the network, as described by Adams and Siegel [1984] and Siegel [1985]. Stage m and 0 enabling and disabling is performed by a system control unit. In the fault-free state, the network will be set so that stage m is disabled

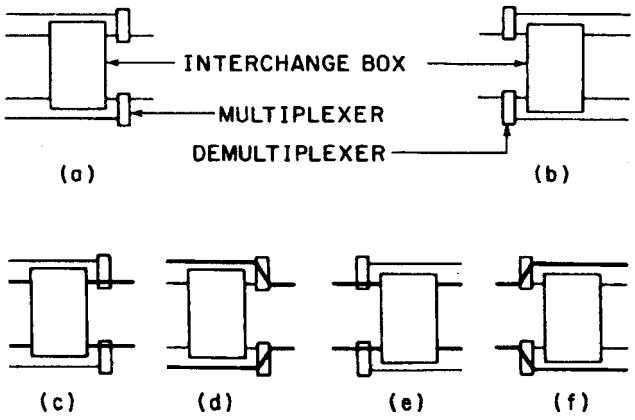

Figure 9. (a) Details of stage m interchange box. (b) Details of stage 0 interchange box. (c) Stage m enabled. (d) Stage m disabled. (e) Stage 0 enabled. (f) Stage 0 disabled.

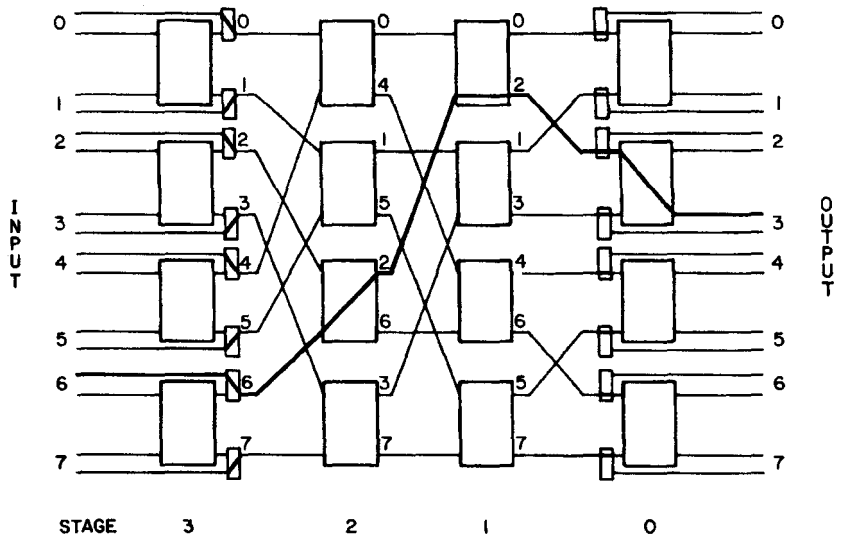

Figure 10. The path from input 6 to output 3 in the ESC network for $N = 8$, when stage 3 is disabled and stage 0 is enabled.

(bypassed) and stage 0 is enabled, as shown in figure 10. The resulting structure is that of the Generalized Cube network.

In the fault model to be used, failures may occur in network interchange boxes, links, or network input/output lines. Once a fault has been detected and located in the ESC network, the faulty component is considered unusable until it is repaired or replaced. It should also be noted that a failure in a stage m multiplexer or stage 0 demultiplexer has the effect of a link fault, which the ESC network can tolerate.

Techniques such as test patterns [Davis et al. 1985; Feng and Wu 1981; Feng and Zhang 1985], dynamic parity checking [Siegel and McMillen 1981], and handshaking signals for fault detection and location have been described for use in the Generalized Cube network topology. This section is not concerned with fault detection and location, but rather with how to recover from a fault once it is located.

4.2. Single-Fault Tolerance

If the fault is a box in stage m or 0, the stage can be disabled. The remaining m stages are sufficient to provide one path between any source and destination (i.e., all m cube functions are still available). This is demonstrated in figures 10 and 11 for $N = 8$ and $S = 6$.

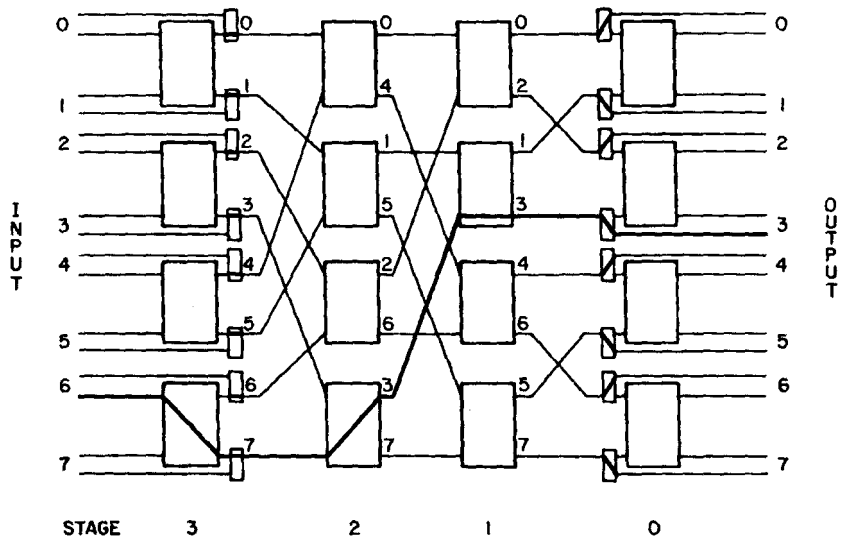

Figure 11. The path from input 6 to output 3 in the ESC network for $N = 8$, when stage 3 is enabled and stage 0 is disabled.

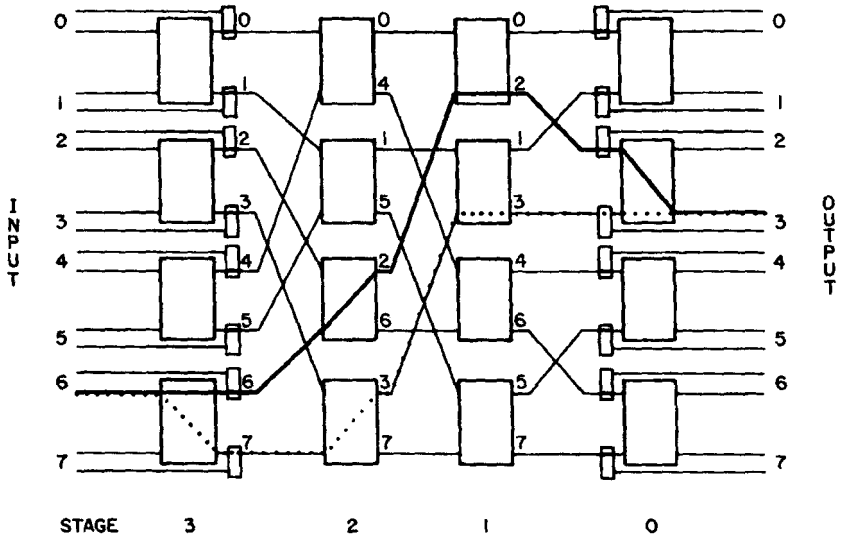

Figure 12. The paths from input 6 to output 3 in the ESC network for $N = 8$, when both stages 3 and 0 are enabled.

If the fault is in a link in a stage i box, $1 \le i < m$, both stages m and 0 are enabled, creating exactly two potential paths between any source and any destination. Stage m of the ESC network allows access to two distinct stage $m-1$ inputs, S and $\text{cube}_0(S)$. Stage $m-1$ to 0 of the ESC network form a Generalized Cube network topology, so each of the two stage $m-1$ inputs has a single path to the destination and these paths are distinct, except for the stage m and 0 boxes and network input/output ports. This is because the links on the path from S will have s_0 as their low-order bit, and the links from $\text{cube}_0(S)$ will have $\bar{s}_0$, as shown in figure 12 for $S = 6$. Distinct paths allow continued one-to-one communication between a given source and destination because if a single fault occurs, at least one path remains fully functional.

If an input line connecting a processor to a stage m multiplexer fails, the stage m boxes are enabled and their input lines are used. If the fault is on an input line to a stage m interchange box, that line is currently unused and the system continues to ignore the faulty line. If an output line from a stage 0 box to a processor is faulty, the network is reconfigured as if stage 0 is faulty. If the fault is on an output line from a demultiplexer to a processor, that line is currently unused and the system continues to ignore the faulty line.

Broadcasting in the ESC is an extension of the ideas for one-to-one paths and will not be discussed here; see Adams and Siegel [1982] and Siegel [1985] for information about this.

Finding fault-free paths. The ESC network path from S to D corresponding to the Generalized Cube network path is called the *primary path*. This path either bypasses stage m (stage m is disabled) or uses the straight setting in stage m. The other path from S to D is called the *secondary path*. It uses the exchange setting in stage m (stage m is enabled). This is exemplified in figure 12, where the solid line is the primary path and the dotted line is the secondary path. For the source/destination pair $S = s_{m-1} \ldots s_1 s_0$ and $D = d_{m-1} \ldots d_1 d_0$, the primary path uses the stage i output labeled $d_{m-1} \ldots d_{i+1} d_i s_{i-1} \ldots s_1 s_0$ and the secondary path uses $d_{m-1} \ldots d_{i+1} d_i s_{i-1} \ldots s_1 \bar{s}_0$, for $0 \le i < m$ (see figure 12). This follows from the discussion about the Generalized Cube in section 3.

A *fault label* specifying the fault location is sent to each processor when a fault has been detected and located in a link or a stage i interchange box, $1 \le i < m$. If the output link from a stage i box fails, each processor receives the fault label (i, j). If a box in stage i with outputs j and k fails, the pair of fault labels (i, j) and (i, k) is sent to each processor. Since j and k will differ only in the ith bit position, a single faulty link number with an "X" ("don't care") in the ith position could be used.

For one-to-one connections, a source S can check to see if the primary path to its intended destination D contains a faulty link or faulty stage i box, $1 \le i < m$. If the faulty stage number is i, $1 \le i < m$, the source forms $d_{m-1} \ldots d_{i+1} d_i s_{i-1} \ldots s_1 s_0$, which is the number of the stage i output the primary path would use to reach its destination. (If there is a faulty stage m output link, then $s_{m-1} \ldots s_1 s_0$ is

used.) The source then compares this with the faulty link number (or box number). If there is a match, then the primary path is faulty; if not, it is fault free. If the primary path is fault free, it will be used. If faulty, the secondary path will be fault free and thus usable (since only single faults are being considered in this subsection). For example, in figure 12, if the stage 2 box with output links 2 and 6 is faulty, the fault label generated would be (2, X10). Source 6 ($S = 6$) forms the address $d_2 s_1 s_0 = 010$, which matches X10, so the secondary path would be used since the primary path is faulty.

If there is no strong preference to using the primary versus secondary path, the test to check for a faulty link or faulty stage i box, $1 \le i < m$, can be reduced to just comparing on a single bit position. If the low-order bit of the source address and faulty link number agree, then the primary path *may* be faulty, so the secondary path can be used. For example, assume the stage 1 output link 2 is faulty in figure 12. Then the low-order bit of link number 2 and source 2 is compared, found to be equal, and so the secondary path is used since the primary path may be blocked (which it is). Since the secondary path uses links whose low-order bit is $\bar{s}_0$, it will be fault free. This simplified procedure may result in the unnecessary use of secondary paths; however, in most applications it does not matter whether the primary or secondary path is used.

For a fault in stage 0, no fault label will be given, only notice that a stage 0 fault exists so that stage m can be used instead of stage 0 to implement cube_0 if needed. This is because stage 0 will be disabled. Stage m faults require system maintenance but no labels need be issued, as the stage will be disabled, and the network will function as if it is fault free (stage m disabled, stage 0 enabled).

Routing tags. In the ESC network with one fault, any one-to-one connection performable on the Generalized Cube network with the exclusive-or routing tag T can be performed using the $(m+1)$-bit routing tag T^* (t^*_m controls stage m) obtained from the following rules.

1. If the fault is in stage 0, use $T^* = t_0 t_{m-1} \ldots t_1 t_0^*$, where t_0^* is arbitrary (recall that stage m will be enabled and stage 0 will be disabled). Stage m functionally replaces stage 0.
2. If the fault is in a link or in a box in stages $m-1$ to 1 and the primary path is fault free, use $T^* = 0 t_{m-1} \ldots t_1 t_0$. If the primary path is faulty, use the secondary path $T^* = 1 t_{m-1} \ldots t_1 \bar{t}_0$ (since) $t^*_m = 1$ connects S to input $\text{cube}_0(S)$ of stage $m-1$, bits $m-1$ to 0 of T^* must be $\text{cube}_0(S) \oplus D = t_{m-1} \ldots t_1 \bar{t}_0$).
3. If there is no fault or the fault is in a stage m box, use $T^* = t^*_m t_{m-1} \ldots t_1 t_0$, where t^*_m is arbitrary (recall that stage m will be disabled and stage 0 enabled).

4.3. Multiple-Fault Tolerance

The previous subsection establishes the capability of the ESC network to tolerate

a single fault in the sense that any one-to-one connection possible in the fault-free ESC network remains possible; that is, the ESC retains its *fault-free interconnection capability*. When there are multiple faults, the ESC network retains fault-free interconnection capability if the primary and secondary paths are not both faulty.

First, consider the situation where there are no faults in stage m or 0 boxes (i.e., the multiple faults are due to faulty links or faulty stage i boxes, $1 \le i < m$). As faults are detected and located, a system control unit can determine whether fault-free interconnection capability is retained. Let $A = (i, a_{m-1} \ldots a_1 a_0)$ and $B = (j, b_{m-1} \ldots b_1 b_0)$, where $1 \le j \le i < m$, be two fault labels. If $a_{m-1} \ldots a_{i+1} a_i \neq b_{m-1} \ldots b_{i+1} b_i$, or if $a_{j-1} \ldots a_1 \bar{a}_0 \neq b_{j-1} \ldots b_1 b_0$, then there will be at least one fault-free path between any source and destination [Adams and Siegel 1982; Siegel 1985]. This is called the *fault-free interconnection capability criterion*. If A corresponds to a faulty stage m output link, then the fault-free interconnection capability criterion is $a_{j-1} \ldots a_1 \bar{a}_0 \neq b_{j-1} \ldots b_1 b_0$. The source/destination pairs S/D which can no longer communicate when the fault-free interconnection capability criterion is not met due to multiple faults in the links and/or stage i boxes, $1 \le i < m$, are of the form $s_{i-1} \ldots s_2 s_1 = a_{i-1} \ldots a_2 a_1$; $d_{m-1} \ldots d_{j+1} d_j = b_{m-1} \ldots b_{j+1} b_j$; and $s_{m-1} \ldots s_{i+1} s_i$, s_0, and $d_{j-1} \ldots d_1 d_0$ are arbitary [Adams and Siegel 1982; Siegel 1985].

Thus, if the multiple faults occur in the links and/or in stage i boxes, $1 \le i < m$, the fault label(s) of any new fault(s) is compared with all existing fault label(s). If each pair of fault labels meets the fault-free interconnection capability criterion, then the network retains its fault-free interconnection capability. If any pair of fault labels does not meet the criterion, complete fault-free interconnection capability does not exist, and the S/D pairs affected can be determined.

With multiple stage 0 faults only or multiple stage m box faults only, the stage is simply disabled, as for a single fault; fault-free interconnection capability still exists. If faults exist in boxes in both stages m and 0, they are both disabled, and the network is incapable of performing "cube$_0$" (it cannot connect S and D if $s_0 \neq d_0$); thus, fault-free interconnection capability is lost.

If there are faults in either stage m or stage 0 boxes (but not both), and faults in other parts of the network, then fault-free interconnection capability no longer exists. This is because with only one of stages m and 0 enabled there exists only one path between any source and any destination. Therefore, any faulty box in stage i, $1 \le i < m$, or any faulty link will block the only path available for certain source/destination pairs.

An enhancement to the ESC called *box bypassing* improves the chances that the ESC will tolerate multiple faults [Adams and Siegel 1984]. Instead of bypassing an entire stage in case of a fault in a stage m or stage 0 box, individual interchange boxes are bypassed, so there will still be two possible paths for each of the processors not using the disabled box. For example, in figure 12, if the stage 0 box with links 0 and 1 is faulty, only that box is bypassed, and there are still two paths from 6 to 3. However, if the entire stage 0 is bypassed, the primary path from 6 to 3 is lost. (The secondary path from 6 to 3 is still available, as

shown in figure 11.) The situation is similar for faults in stage m.

If fault-free interconnection capability exists, full network operations continue by sending the additional fault label(s) to each source. Each source checks its primary path against all network faults to decide whether that path is fault free. This may degrade system performance somewhat, but not at all significantly.

4.4. *Partitioning*

The ways in which the Generalized Cube network can be partitioned were discussed in the previous section. The ESC network can be partitioned in a similar manner, with the property that each subnetwork has the attributes of the ESC network, including fault tolerance. The only constraint is that the partitioning cannot be based on the 0-th bit position of the input/output port addresses (i.e., the PEs cannot be partitioned into even and odd subnetworks). This is because, in order to be able to use either the primary or secondary path and yet remain within a partition, both sources S and $\text{cube}_0(S)$ must be in the same partition. Since the addresses of the interchange box outputs and links of a primary path and a secondary path differ only in the 0-th bit position, both paths will be in the same partition (i.e., they will agree in the bit position(s) upon which the partitioning is based). Thus, the fault-tolerant routing scheme of the ESC network is compatible with network partitioning.

For example, in figure 13, the ESC network for $N = 8$ is shown partitioned with

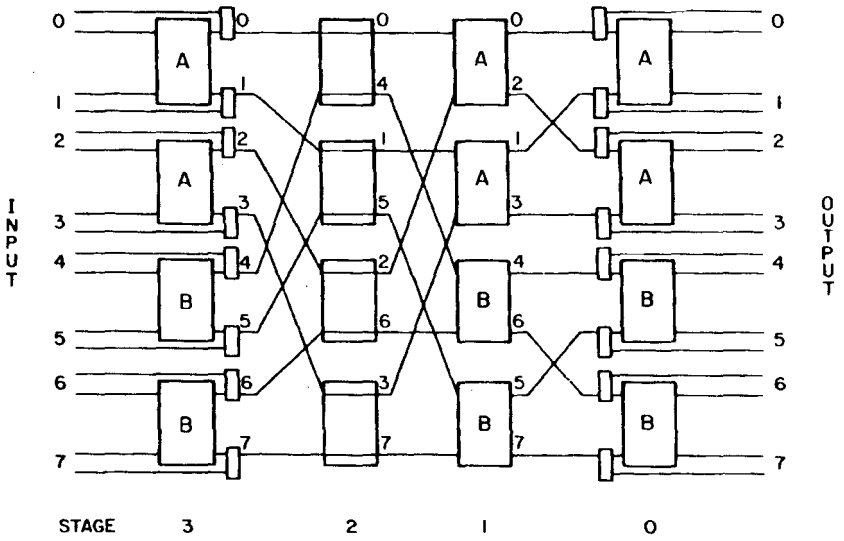

Figure 13. The ESC network for $N = 8$ partitioned into two independent subnetworks of size four, based on the high-order bit position. The labels A and B denote the two subnetworks.

respect to stage 2. The two subnetworks are indicated by the labels A and B. Subnetwork A consists of ports 0, 1, 2, and 3. These port addresses agree in the high-order bit position (it is 0). Subnetwork B contains ports 4, 5, 6, and 7, all of which agree in the high-order bit position (it is 1).

As described for the Generalized Cube network, partitioning can be controlled by logically ANDing tags with partitioning masks to force to 0 those tag positions corresponding to interchange boxes that should be set to straight. Partitioning is always possible if fault-free interconnection capability exists (i.e., a faulty box is avoided, whether or not it is forced to straight).

4.5. SIMD Operation

Consider performing a permutation on the ESC that can be done on the Generalized Cube in one pass. If the ESC is fault free, it can obviously perform any such permutation in one pass. If there is a fault in a box or a link in stage i, $1 \leq i < m$, there are no longer N paths through the network, so two passes are required. One path involves all the processors that are using their primary paths, and the other path involves the processors that are using their secondary paths. If the fault is in a stage 0 box, stage 0 is disabled, and stage m is used for cube_0 in its place. Since the order of performing the cube_i connections affects the permutations that can be performed, two passes are used: In the first pass, stage m is set to straight and data is permuted through stages $m-1, m-2, \ldots, 1$, and, in the second pass, stage m is set as stage 0 would be and all other stages are set to straight. If the fault is in stage m, stage m is disabled and the ESC is equivalent to a Generalized Cube. Hence, in this case only one pass is necessary.

5. Dynamic Redundancy Network

5.1 Motivation

For systems that contain a large number of PEs, the possibility of PE failures cannot be neglected. One way to tolerate PE failures is the use of *dynamic redundancy*. Using dynamic redundancy to tolerate faulty PEs means that a system contains spare PEs and uses them to replace faulty PEs, when detected. The ESC network discussed above does not have extra I/O ports for spare PEs; hence, it cannot be used directly for such systems.

In the following, a fault-tolerant multistage interconnection network is presented that can be used directly to support multiprocessor systems containing spare PEs. We call this the *Dynamic Redundancy* (DR) network [Jeng and Siegel 1986a and b]. It can provide all of the functionality of the Generalized Cube multistage network.

The DR network can tolerate any single switch or link failure (including input

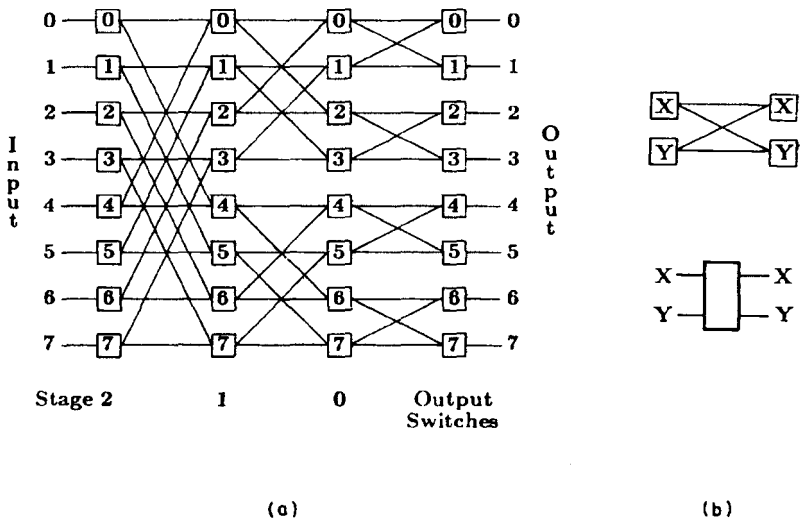

Figure 14. (a) Graphical interpretation of the multistage cube network for $N = 8$. (b) Relationship between graphical interpretation and interchange box representation.

and output stage switches and the links between PEs and I/O ports of the network) and provide the necessary capabilities for the system to recover from any single PE failure. It will be discussed later how, when operating in a partitionable environment, that is, an environment where the system can be partitioned into different virtual machines, the DR network can provide single-fault tolerance for each subsystem. As in the ESC section, we are not concerned with fault detection and location here, but with fault recovery.

A PE that participates in the execution of tasks will be called a *functioning* PE; otherwise it is a *spare* PE. A spare PE will become functioning when a faulty functioning PE is detected and isolated. A DR network that begins with N functioning PEs and σ spare PEs will be discussed.

5.2. *Structure of the DR Network*

The design of the DR network is based on the interconnection graph of the generalized cube multistage network. Figure 14 shows an equivalent SW-banyan graph of multistage cube network with $N = 8$ [Malek and Myre 1981; Siegel 1985]. Consider each node as a switch and each edge as a link. Using this representation of the cube, each stage has N switches and $2N$ links. Recall that the basic concept underlying the multistage cube topology is that, at stage i, PEs whose network input port labels differ in the ith bit position can exchange data (i.e., stage i implements cube$_i$). Using the representation in figure 1, at stage i, link j and link

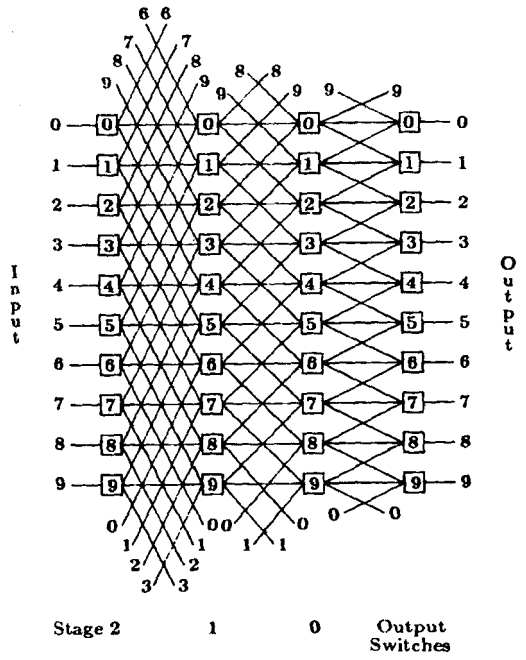

Figure 15. A DR network for $N = 8$ and $\sigma = 2$.

$\text{cube}_i(j)$ can exchange data. Using the representation in figure 14, at stage i, switch j and switch $\text{cube}_i(j)$ can exchange data; that is, switch j in stage i is connected to switch $\text{cube}_i(j)$ in stage $i-1$, $0 \le j < N$, $0 \le i < m$.

Since the DR interconnection network is to be used in multiprocessor systems which contain N functioning PEs and σ spare PEs, it must contain $N+\sigma$ I/O ports. Like the Generalized Cube, the DR network contains m stages, where $N = 2^m$, as shown in figure 15 for $N = 8$ and $\sigma = 2$. Stages are ordered from $m-1$ to 0 from the input side to the output side of the network. Each stage has $N+\sigma$ switches followed by $3(N+\sigma)$ links. In addition, there are $N+\sigma$ output switches. Let the PEs of the multiprocessor system be numbered from 0 to $N+\sigma-1$. PE j of the system is connected to the input of switch j of stage $m-1$ and to the network output switch j, where $0 \le j \le N+\sigma-1$. Each switching element j at stage i of the network has three output links to stage $i-1$. The first output link is connected to switch $(j-2^i) \bmod (N+\sigma)$ of stage $i-1$, the second to switch j of stage $i-1$, and the third to switch $(j+2^i) \bmod (N+\sigma)$ of stage $i-1$. The network output nodes are the output ports of the network.

The graph representation of a DR network is similar to that of an ADM network [McMillen and Siegel 1985], except that the ADM network has N I/O ports and uses modulo N instead of modulo $(N+\sigma)$ in the connection functions.

Because of the spare I/O ports, the interconnection and fault-tolerance capabilities of the DR network will be quite different from those of ADM.

5.3. *Control of the DR Network*

In a multiprocessor system containing $N+\sigma$ PEs, since only N PEs are functioning at a time, only N of the $N+\sigma$ network I/O ports are needed at a time to provide the inter-PE communication for the system. Hence, the capability of a subset of the network is of interest here rather than the whole network.

When there are no faults, PEs 0 to $N-1$ are used as the functioning PEs, and switches numbered from 0 to $N-1$ and their associated links emulate the multistage cube. At stage i, DR switches j and $\text{cube}_i(j)$ emulate the stage i interchange box with inputs j and $\text{cube}_i(j)$; assuming $j < \text{cube}_i(j)$, setting switch j to $+2^i$ and switch $\text{cube}_i(j)$ to -2^i emulates "exchange," and setting switches j and $\text{cube}_i(j)$ to straight emulates "straight." (Upper and lower broadcasts are defined similarly.)

If PE j or switch j (any stage) or a link attached to switch j (any stage) is found to be faulty, the system is reconfigured so that the PE and switches physically numbered P are numbered

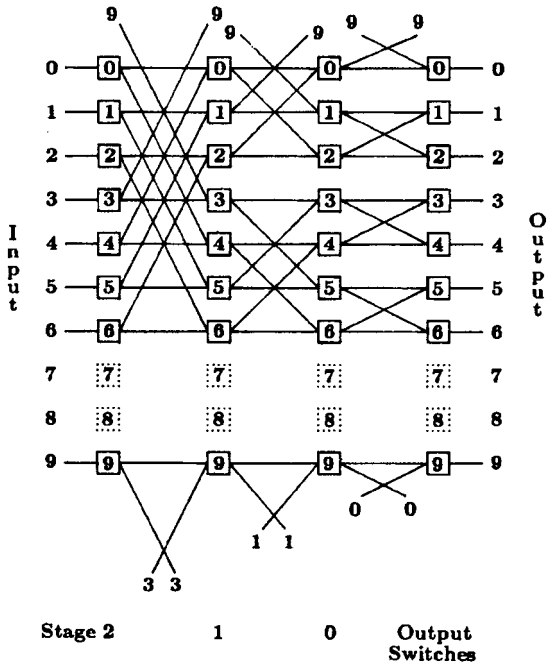

Figure 16. Reconfiguration of a DR network with $N=8$ and $\sigma=2$ when PE 7 is faulty. The *solid lines* show the cube subgraph of the DR network.

$$P-(j+\sigma) \bmod N+\sigma$$

Then PEs logically numbered 0 to $N-1$ are used as the functioning PEs, and switches logically numbered 0 to $N-1$ and their associated links emulate the multistage cube, in the same manner as described for the fault-free case above (but based on the logical numbering). An example of reconfiguration is shown in figure 16, where each logical number can be obtained by subtracting 9 from the corresponding physical number given, subtraction modulo 10.

Routing tags for the DR are based on the logical numbers for the PEs and switches, where the logical numbers in the faulty situation are as defined above, and for the no-fault situation physical PE and switch i is logically numbered i, $0 \le i < N$. Given a source PE with logical address S and a destination PE with logical address D, then the *exclusive-or routing tag* T can be derived by taking the bitwise exclusive-or of S and D (as done for the Generalized Cube network). When a switch in stage i receives a message, it examines t_i. If $t_i = 0$, then the straight link is used. If $t_i = 1$, then a switch has to decide to use the $+2^i$ link or the -2^i link. Let $W = w_{m-1} \ldots w_1 w_0$ represent the logical address of the switch. If $w_i = 0$, switch W will use $+2^i$ link; otherwise it will use -2^i link. This will emulate a stage i exchange, that is, connect W to $\text{cube}_i(W)$.

This approach necessitates adding a one-bit flag in each switch to store the ith bit of the logical address of the switch, w_i. These flags can be set by special messages sent from PEs during system initialization and after each reconfiguration due to a fault. This allows the same routing tags used in the multistage cube network to be used directly to control the DR network.

There is an alternative method that does not require each switch to store its logical number, but does require sending the logical address of the source PE, an additional m bits, in the tag. A connection path from a logical source PE S to a logical destination PE D will use the switch with logical address $d_{m-1} \ldots d_{i+1} s_i \ldots s_0$ at stage i (this was shown for the Generalized Cube in section 3). Thus, $w_i = s_i$, and each switch can use s_i in place of w_i.

The network can also be controlled by using destination tags. A *destination tag* used for the DR network will contain the source PE logical address S and the destination PE logical address D. A switch in stage i can examine s_i and d_i to select a link. The switch will select link -2^i when $d_i < s_i$, the straight link when $d_i = s_i$, or link $+2^i$ when $d_i > s_i$. Instead of sending S, w_i could be used in place of s_i.

Broadcasting on the DR network is an extension of this approach. It is discussed by Jeng and Siegel [1986a].

5.4. *Partitionability of the DR Network*

The partitionability of the DR network depends on the value of σ. When σ is even, the DR network can be partitioned into two independent subnetworks by

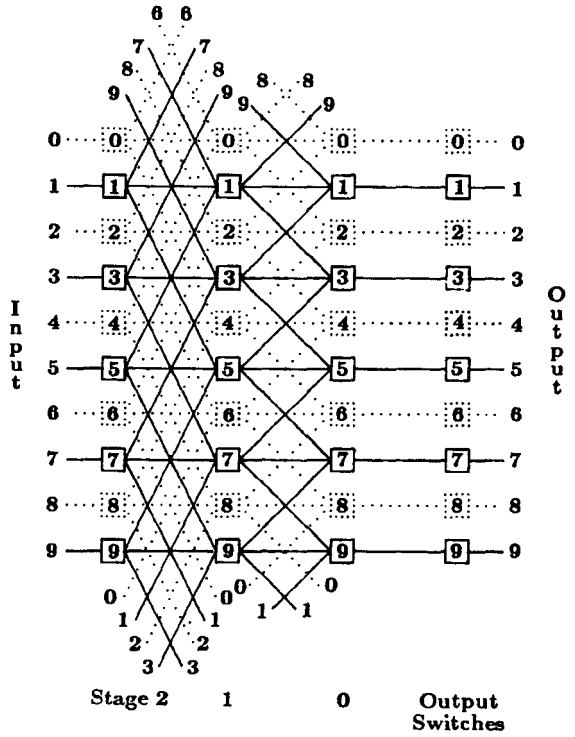

Figure 17. Partitioning a DR network with $N = 8$ and $\sigma = 2$ into two independent subnetworks. The *solid lines* show the subnetwork containing only odd-number switches. The *dotted lines* show the subnetwork containing only even-number switches.

setting all switches in stage 0 to straight. The theory underlying this is similar to that for partitioning the ADM network, as discussed by Siegel [1980 and 1985]. An example of partitioning the DR network with $N = 8$ and $\sigma = 2$ is shown in figure 17. Each subnetwork is a DR network and can be partitioned. The limit is that a DR network of size $N + \sigma$, where $N = 2^m$ and $\sigma = 2^q$, can be partitioned into σ subnetworks of size $N' + \sigma'$, where $N' = 2^{m-q}$ and $\sigma' = 1$. The physical addresses of all the switches in a subnetwork of size $2^{m-r} + 2^{q-r}$ must agree in their lower-order r bit positions, and partitions can be of different sizes.

The partitionability of the DR network can be used to support systems with multiple-SIMD, partitionable SIMD/MIMD, and multiple virtual MIMD capabilities; each subsystem (partition) can use spare PEs to enhance system reliability. Control and reconfiguration of each subnetwork for each subsystem will be the same as discussed above for a complete DR network.

6. Summary

Properties of the Generalized Cube network, a representative of the multistage cube family, were overviewed. Advantages of the Generalized Cube include hardware complexity proportional to $N\log_2 N$, distributed control using routing tags, the capability for one PE to broadcast to all or a subset of the others, support for up to N simultaneous transfers, partitionability into independent subnetworks, adaptability for use in SIMD or MIMD mode, and options for a variety of implementation methods. The Extra Stage Cube (ESC) network has all the advantages of the Generalized Cube; plus it is single-fault tolerant (although it requires two passes for permutations), robust for two faults (when box bypassing is used, there is approximately a 90% probability that fault-free interconnection capability is retained for $N = 1024$), and analyzable to determine whether multiple faults will cause degradation. Since the Dynamic Redundancy (DR) network emulates the Generalized Cube, it has all the advantages of the Generalized Cube, plus it is single-fault tolerant (including permutations), can tolerate single faults in σ partitions (if the network size is $N + \sigma$), and supports PE fault tolerance through dynamic redundancy (allows a spare PE to replace a faulty one).

The purpose of this article was to introduce the features of the multistage cube family of networks that make them attractive for use in future parallel supercomputers, taking into consideration the aspect of fault tolerance, which is so very important in large-scale parallel systems. Further information is available in the references cited.

Acknowledgments

Figures 1–7 are from Siegel [1985]. All figures were drawn by Sharon Katz. A preliminary version of this material appeared in the *New Frontiers in Computer Architecture Conference Proceedings*, March 1986.

References

Adams III, G. B., and Siegel, H. J. 1982. The extra stage cube: a fault-tolerant interconnection network for supersystems. *IEEE Trans. Comput.* C-31 (May), 443–454.

Adams III, G. B., and Agrawal, D. P., and Siegel, H. J. 1987. A survey and comparison of fault-tolerant multistage interconnection networks. *Computer* (in press).

Adams III, G. B., and Siegel, H. J. 1984. Modifications to improve the fault tolerance of the extra stage cube interconnection network. In *1984 International Conference on Parallel Processing* (August), pp. 169–173.

Barnes, G. H., and Lundstrom, S. F. 1981. Design and validation of a connection network for many-processor multiprocessor systems. *Computer*, 14 (December), 31–41.

Batcher, K. E. 1974. STARAN parallel processor system hardware. In *AFIPS 1974 National Computer Conference* (May), pp. 405–410.

Batcher, K. E. 1976. The flip network in STARAN. In *1976 International Conference on Parallel Processing* (August), pp. 65–71.

Batcher, K. E. 1977. STARAN series E. In *1977 International Conference on Parallel Processing* (August), pp. 140–143.

Batcher, K. E. 1980. Design of a massively parallel processor. *IEEE Trans. Comput.*, C-29 (September), 836–844.

Batcher, K. E. 1982. Bit serial parallel processing systems. *IEEE Trans. Comput.*, C-31 (May), 377–384.

Briggs, F. A., Fu, K.-S., Hwang, K., and Wah, B. W. 1982. PUMPS architecture for pattern analysis and image database management. *IEEE Trans. Comput.*, C-31 (October), 969–982.

Broomell, G., and Heath, J. R. 1983. Classification categories and historical development of circuit switching topologies. *ACM Comput. Surveys*, 15 (June), 95–133.

Crowther, W., Goodhue, J., Starr, E., Thomas, R., Williken, W., and Blackadar, T. 1985. Performance measurements on a 128-node Butterfly parallel processor. In *1985 International Conference on Parallel Processing* (August), pp. 531–540.

Davis IV, N. J., Hsu, W. T.-Y., and Siegel, H. J. 1985. Fault location techniques for distributed control interconnection networks. *IEEE Trans. Comput.*, (October), 902–910.

Delp, E. J., Siegel, H. J., Whinston, A., and Jamieson, L. H. 1985. An intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture. In *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management* (November), pp. 217–224.

Dennis, J. B., Boughton, G. A., and Leung, C. K. C. 1980. Building blocks for data flow prototypes. In *Seventh Annual Symposium on Computer Architecture* (May), pp. 1–8.

Feng, T. Y. 1981. A survey of interconnection networks. *Computer*, 14 (December), 12–27.

Feng, T. Y., and Wu, C.-L. 1981. Fault-diagnosis for a class of multistage interconnection networks. *IEEE Trans. Comput.*, C-30 (October), 743–758.

Feng, T. Y., and Zhang, Q. 1985. Fault diagnosis of multistage interconnection networks with four valid states. In *Fifth International Conference on Distributed Computing Systems* (May), pp. 218–226.

Filip, A. E. 1982. A distributed signal processing architecture. In *Third International Conference on Distributed Computing Systems* (October), pp. 49–55.

Flynn, M. J. 1966. Very high-speed computing systems. *Proc. IEEE*, 54 (December), 1901–1909.

Goke, G. R., and Lipovski, G. J. 1973. Banyan networks for partitioning multiprocessor systems. In *First Annual Symposium on Computer Architecture* (December), pp. 21–28.

Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K. P., Rudolph, L., and Snir, M. 1983. The NYU Ultracomputer—designing an MIMD shared-memory parallel computer. *IEEE Trans. Comput.*, C-32 (February), 175–189.

Hillis, W. D. 1985. *The Connection Machine*. MIT Press, Cambridge, MA.

Hockney, R. W., and Jesshope, C. R. 1981. *Parallel Computers*. Adam Hilger, Bristol, England.

Hwang, K., and Briggs, F. A. 1984. *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, NY.

Intel Corporation. 1985. *A New Direction in Scientific Computing*. Order 28009-001, Intel Corporation.

Jeng, M., and Siegel, H. J. 1986a. A fault-tolerant multistage interconnection network for multiprocessor systems using dynamic redundancy. In *Sixth International Conference on Distributed Computing Systems* (June), pp. 70–77.

Jeng, M., and Siegel, H. J. 1986b. Implementation approach and reliability estimation of dynamic redundancy networks. In *1986 Real-Time Systems Symposium* (December), pp. 79–88.

Jones, A. K., Chansler, R. J., Jr., Durham, I., Feiler, P., and Schwans, K. 1977. Software management of Cm*—a distributed multiprocessor. In *AFIPS 1977 National Computer Conference* (June), pp. 657–663.

Kapur, R. N., Premkumar, U. V., and Lipovski, G. J., 1980. Organization of the TRAC processor-memory subsystem. In *AFIPS 1980 National Computer Conference* (June), pp. 623–629.

Lawrie, D. H. 1975. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, C-24

(December), 1145–1155.

Malek, M., and Myre, W. W. 1981. A description method for interconnection networks. *IEEE Technical Committee Distrib. Processing Quart.*, (February), 1–6.

Masson, G. M., Gingher, G. C., and Nakamura, S. 1979. A sampler of circuit switching networks. *Computer*, 12 (June), 32–48.

McMillen, R. J., and Siegel, H. J. 1980. The hybrid cube network. In *Distributed Data Acquisition, Computing, and Control Symposium* (December), pp. 11–22.

McMillen, R. J., and Siegel, H. J. 1985. Evaluation of cube and data manipulator networks. *J. Parallel Distrib. Comput.*, 2 (February), 79–107.

McMillen, R. J., Adams III, G. B., and Siegel, H. J. 1981. Performance and implementation of 4 × 4 switching nodes in an interconnection network for PASM. In *1981 International Conference on Parallel Processing* (August), pp. 229–233.

McDonald, W. C., and Williams, J. M. 1978. The advanced data processing test bed. In *IEEE Computer Society Second International Computer Software and Applications Conference* (March), pp. 346–351.

Nutt, G. J. 1977a. Microprocessor implementation of a parallel processor. In *Fourth Annual Symposium on Computer Architecture* (March), pp. 147–152.

Nutt, G. J. 1977b. A parallel processor operating system comparison. *IEEE Trans. Software Eng.*, SE-3 (November), 467–475.

Patel, J. H. 1981. Performance of processor-memory interconnections for multiprocessors. *IEEE Trans. Comput.*, C-30 (October), 771–780.

Pease III, M. C. 1977. The indirect binary n-cube microprocessor array. *IEEE Trans. Comput.*, C-26 (May), 458–473.

Pfister, G. F., Brantley, W. C., George, D. A., Harvey, S. L., Kleinfelder, W. J., McAuliffe, K. P., Melton, E. A., Norton, V. A., and Weiss, J. 1985. The IBM Research Parallel Processor Prototype (RP3): introduction and architecture. In *1985 International Conference on Parallel Processing* (August), pp. 764–771.

Premkumar, U. V., Kapur, R. N., Malek, M., Lipovski, G. J., and Horne, P. 1980. Design and implementation of the Banyan interconnection network in TRAC. In *AFIPS 1980 National Computer Conference* (June), pp. 643–653.

Sejnowski, M. C., Upchurch, E. T., Kapur, R. N., Charlu, D. P. S., and Lipovski, G. J. 1980. An overview of the Texas Reconfigurable Array Computer. In *AFIPS 1980 National Computer Conference* (June), pp. 631–641.

Siegel, H. J. 1977. Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks. *IEEE Trans. Comput.*, C-26 (February), 153–161.

Siegel, H. J. 1979. Interconnection networks for SIMD machines. *Computer*, 12 (June), 57–65.

Siegel, H. J. 1980. The theory underlying the partitioning of permutation networks. *IEEE Trans. Comput.*, C-29 (September), 791–801.

Siegel, H. J. 1985. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*. Lexington Books, D. C. Heath and Company, Lexington, MA.

Siegel, H. J. and McMillen, R. J. 1981. The multistage cube: a versatile interconnection network. *Computer*, 14 (December), 65–76.

Siegel, H. J., and Smith, S. D. 1978. Study of multistage SIMD interconnection networks. In *Fifth Annual Symposium on Computer Architecture* (April), pp. 223–229.

Siegel, H. J., McMillen, R. J., and Mueller, P. T. Jr. 1979. A survey of interconnection methods for reconfigurable parallel processing systems. In *AFIPS 1979 National Computer Conference* (June), pp. 529–542.

Siegel, H. J., Mueller, P. T., Jr., and Smalley, H. E., Jr 1978. Control of a partitionable multimicroprocessor system. In *1978 International Conference on Parallel Processing* (August), pp. 9–17.

Siegel, H. J., Schwederski, T., Davis IV, N. J., and Kuehn, J. T. 1984. PASM: a reconfigurable parallel system for image processing. In *Workshop on Algorithm-guided Parallel Architectures for Automatic Target Recognition* (July), 263–291. (Also appears in the ACM SIGARCH newsletter: *Comput. Architect. News*, 12, 4 (September), 7–19.)

Siegel, H. J., Siegel, L. J., Kemmerer, F. C., Mueller, P. T., Jr., Smalley, H. E., Jr., and Smith, S. D. 1981. PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Trans. Comput.*, C-30 (December), 934–947.

Smith, S. D., and Siegel, H. J. 1978. Recirculating, pipelines, and multistage SIMD interconnection networks. In *1978 International Conference on Parallel Processing* (August), pp. 206–214.

Stone, H. S. 1980. Parallel computers. In H. S. Stone (ed.), *Introduction to Computer Architecture*, 2nd ed. Science Research Associates, Chicago, IL, 363–425.

Swan, R. J., Bechtolsheim, A., Lai, K. W., and Ousterhout, J. K. 1977a. The implementation of the Cm* multimicroprocessor. In *AFIPS 1977 National Computer Conference* (June), pp. 645–655.

Swan, R. J., Fuller, S., and Siewiorek, D. P. 1977b. Cm*: a modular multimicroprocessor. In *AFIPS 1977 National Computer Conference* (June), pp. 637–644.

Thanawastien, S., and Nelson, V. P. 1981. Interference analysis of shuffle/exchange networks. *IEEE Trans. Comput.* C-30 (August), 545–556.

Thurber, K. J. 1976. *Large Scale Computer Architecture: Parallel and Associative Processors*. Hayden Book Company, Rochelle Park, NJ.

Thurber, K. J. 1979. Parallel processor architectures—part 1: general purpose systems. *Comput. Design*, 18 (January), 89–97.

Thurber, K. J., and Masson, G. M. 1979. *Distributed-Processor Communication Architecture*. Lexington Books, D. C. Heath and Company, Lexington, MA.

Wu, C.-L., and Feng, T. Y. 1980. On a class of multistage interconnection networks. *IEEE Trans. Comput.*, C-29 (August), 694–702.