Using the Multistage Cube Network Topology in Parallel Supercomputers

HOWARD JAY SIEGEL, SENIOR MEMBER, IEEE, WAYNE G. NATION, CLYDE P. KRUSKAL, MEMBER, IEEE, AND LEONARD M. NAPOLITANO, JR.

A variety of approaches to designing the interconnection network to support communications among the processors and memories of supercomputers employing large-scale parallel processing have been proposed and/or implemented. These approaches are often based on the multistage cube topology. This topology is the subject of much ongoing research and study because of the ways in which the multistage cube can be used. The attributes of the topology that make it useful are described. These include O(N $\log_2$ N) cost for an N input/output network, decentralized control, a variety of implementation options, good data permuting capability to support single instruction stream/multiple data stream (SIMD) parallelism, good throughput to support multiple instruction stream/multiple data stream (MIMD) parallelism, and ability to be partitioned into independent subnetworks to support reconfigurable systems. Examples of existing systems that use multistage cube networks are overviewed. The multistage cube topology can be converted into a single-stage network by associating with each switch in the network a processor (and a memory). Properties of systems that use the multistage cube network in this way are also examined.

I. Introduction

One very important approach to the design of supercomputers is the use of large-scale parallel processing systems, i.e., computer systems with 2^6–2^{16} processors working together to solve a problem. The domain of problems that require the use of supercomputers are those that have a "need for speed" due to some combination of the complexity of the algorithms needed to compute the solution to the problem, the size of the data set to be processed, and the time constraint on when a solution to the problem must be attained. Examples of such problem domains are aerodynamic simulations, air traffic control, ballistic missile defense, biomedical signal processing, chemical reaction simulations, map making, missile guidance, robot vision, satellite-collected imagery analysis, seismic data processing, speech understanding, and weather forecasting.

A critical component of any large-scale parallel processing system is the interconnection network that provides a means for communication among the system's processors and memories. Attributes of the multistage cube topology that have made it an effective basis for interconnection networks and the subject of much ongoing research are overviewed in this paper. The goal is to survey a variety of features of the multistage cube topology that make it attractive for use in supercomputers employing large-scale parallel processing. Appropriate references are listed to allow the interested reader to probe in more depth.

A. The Problem

Designing a vehicle for providing communications among N processors and *N* memory modules in a large-scale parallel processing system, where *N* may be in the range 2^6–2^{16}, is a difficult task. The interconnection scheme must provide the needed communications and be of reasonable cost. One extreme, a single shared bus, would become a bottleneck for *N* in this range if the processors communicate frequently. The other extreme is to link directly each processor to every other processor so that the system is completely connected, requiring $N-1$ unidirectional lines for each processor, for a total of $N(N-1)$ links. This is unreasonable for large *N*. The crossbar network can emulate a completely connected system; however, it uses N^2 crosspoint switches [1], and so, given current technology, it is also infeasible for large *N*.

A great number of networks between these extremes have been proposed in the literature. The cost-effectiveness of a particular network design depends on such factors as the computational tasks for which it will be used (which impacts the frequency of communication, interprocessor communication patterns, and message lengths), the desired speed of interprocessor data transfers, the actual implementation of the network (both hardware and associated software protocols), the number and characteristics of the processors and memories in the system, and constraints on the development and construction costs. Because of the wide range of values these factors can have, there is no one

Manuscript received December 28, 1988; revised April 4, 1989. This work supported in part by the U.S. Air Force Office of Scientific Research under grant F49620-86-K-0006, by Purdue University, and by the U.S. Department of Energy under contract DE-AC04-76DP00789.

H. J. Siegel and W. G. Nation are with the Parallel Processing Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

C. P. Kruskal is with the Department of Computer Science, University of Maryland, College Park, MD 20742.

L. M. Napolitano Jr. is with the Imaging Technology Division 8435, Sandia National Laboratories, P.O. Box 969, Livermore, CA 94551-0969.

IEEE Log Number 8933024.

0018-9219/89/1200-1932$01.00 © 1989 IEEE

network that is considered "best" for all possible situations. A variety of network designs are described in [2]–[9].

B. Appeal of the Multistage Cube Network Topology

Some of the approaches to parallel processing system network design that have been discussed in the literature are based on the multistage cube network topology. The multistage cube network topology has been used or proposed for use in systems such as the Ballistic Missile Defense Agency test bed [10], [11]; BBN Butterfly Plus [12]; BBN GP 1000 [13]; BBN TC2000 [14]; the Burroughs flow model processor for the numerical aerodynamic simulator [15]; Cedar [16]; dataflow machines [17]; DISP [18]; IBM RP3 [19]; PASM [20], [21]; PUMPS [22]; STARAN [23]; and the NYU Ultracomputer [24]. Networks topologically equivalent to the multistage cube have been called different names [6], [25]–[27], including baseline [27], bidelta [28], butterfly [29], delta ($a=b=2$) [30], flip [31], Generalized Cube [11], [26], indirect binary n-cube [32], multistage shuffle-exchange [33], omega [34], and SW-banyan ($S=F=2$) [35] networks. Fault-tolerant versions of the multistage cube that have been proposed, such as the Dynamic Redundancy [36] and Extra Stage Cube [37] networks, are overviewed in [38]. Variations on the multistage cube network that use the same topology to produce a "single-stage" network with $KP(\log_2 P)$ processors, for some fixed K and P, include the CBAR [39], HCSN [40], Lambda [41], LSSN [42], and MAN-YO [43] networks. The Generalized Cube topology is used in this paper to represent this class of topologically equivalent networks.

The advantages of the multistage cube network approach include the number of components being proportional to $N \log_2 N$, efficient distributed control schemes, partitionability, availability of multiple simultaneous paths, and ability to employ a variety of different implementation techniques. It is these advantages and good overall network performance in both the SIMD and MIMD modes of parallelism that make the multistage cube topology appealing.

C. Overview

Section II presents different architectural models of parallel processing systems and provides some basic terminology. The multistage cube topology is defined and properties such as routing tag control and partitionability are described in section III. In section IV, the different modes of operation that the network switches can use are explored. The way in which the network can be used to support SIMD (synchronous) parallelism and MIMD (asynchronous) parallelism are examined in sections V and VI, respectively. Various existing machines employing the multistage cube are overviewed in section VII. Section VIII considers using the multistage cube topology as a single-stage network, where each networks switch is associated with a processor and memory. Section IX discusses the optimality of the multistage cube network and contains some concluding remarks.

II. Architectural Models of Parallel Machines

There is a variety of ways to organize the processors, memories, and interconnection network in a large-scale parallel processing system. In this section, models of a few of the basic structures are briefly introduced. More information is available in textbooks such as [6] and [44]–[50].

A. SIMD Machines

A model of a "single instruction stream–multiple data stream (SIMD) machine" consists of a control unit, N processors, N memory modules, and an interconnection network [51]. 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. Examples of SIMD machines that have been constructed include STARAN [52], MPP [53], [54], and the Connection Machine [55], [56].

B. Multiple-SIMD Machines

A variation on the SIMD model that may permit more efficient use of the system processors and memories is the "multiple-SIMD machine," a parallel processing system that can be dynamically reconfigured to operate as one or more independent SIMD submachines of various sizes. A multiple-SIMD system consists of N processors, N memory modules, an interconnection network, and C control units, where $C < N$. Each of the multiple control units can be connected to some disjoint subset of the processors, which communicate over subnetworks, creating independent SIMD submachines of various sizes. Examples of multiple-SIMD machines are the proposed MAP [57], [58] and existing Connection Machine 2 [56].

C. MIMD Machines

In contrast to the SIMD machine, where all processors follow a single instruction stream, each processor in a parallel machine may follow its own instruction stream, forming a "multiple instruction stream–multiple data stream (MIMD) machine" [51]. One organization for an MIMD machine consists of N processors, N memory modules, and an interconnection network, where each of the processors executes its own program on its own data. Thus there are multiple instruction streams and multiple data streams. 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, because 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^* [59]–[61], the BBN Butterfly [29], the BBN GP 1000 [13], the BBN TC2000 [14], the Intel iPSC cube [62], and the NCube [63].

D. Partitionable SIMD/MIMD Machines

A fourth model of system organization combines the features of the previous three. A "partitionable SIMD/MIMD machine" is a parallel processing system that can be dynamically reconfigured to operate as one or more independent SIMD and/or MIMD submachines of various sizes. The N processors, N memory modules, interconnection network, and C control units of a partitionable SIMD/MIMD system can be partitioned to form independent subma-

chines as with multiple-SIMD machines. Furthermore, 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). Thus each submachine can operate in the SIMD mode or the MIMD mode. The processors can switch between the two modes of parallelism, from one instruction to the next, when performing a task, depending on which is more desirable at the time. Examples of partitionable SIMD/MIMD systems for which prototypes have been built are PASM [20], [21] and TRAC [4].

E. System Configurations

With any of these four models, there are two basic system configurations [6]. One is the "PE-to-PE configuration," in which each processing element, or PE (formed by pairing a processor with a local memory), is attached to both an input port and an output port of an interconnection network (i.e., PE j is connected to input port j and output port j). This is also referred to as a distributed memory system, or private memory system. In contrast, in the "processor-to-memory configuration," processors are attached to one side of an interconnection network and memories are attached to the other side. Processors communicate through shared memories. This is also referred to as a global memory system, or the "dance-hall" model (boys on one side of the room, girls on the other). Hybrids of the two approaches are also possible, such as using a local cache in a processor-to-memory machine (e.g., Ultracomputer [24]). Deciding which of the configurations or hybrid of them is "best" for a particular machine design involves the consideration of many factors, such as the types of computational tasks for which the machine is intended (e.g., are most data and/or programs shared by all processors or local to each processor), the operating system philosophy (e.g., will multitasking be done within each processor to hide any latency time for network transfer delays when fetching data [64]), and the characteristics of the processors and memories to be used (e.g., clock speed, availability of cache).

Beware of the term shared memory as applied to parallel machines. Some researchers use this term to refer to the way in which a machine is physically constructed (i.e., a processor-to-memory configuration) and others use it to refer to the logical addressing method. A logically shared address space paradigm can be supported by either of the two basic physical configurations (PE-to-PE or processor-to-memory), or hybrids of them.

F. Remark

The multistage cube can support all four models of parallel computation and both types of system configurations. To simplify discussions, the PE-to-PE system configuration will be assumed in sections III–VI unless stated otherwise. However, the information contained in those sections is also relevant to the processor-to-memory configuration and hybrids.

III. Multistage Cube Network Topology Properties

Various inherent properties of the multistage cube topology are presented in this section. These include path establishment, distributed routing tag control, and partitionability. These general properties are not unique to the multistage cube. The single-stage (e.g., hypercube) and other multistage (e.g., ADM) networks have analogous but not identical properties [6], [65], [66].

A. Network Structure

Fig. 1 shows the "Generalized Cube" network topology for N inputs and N outputs. The Generalized Cube topology was introduced as a standard for comparing different types

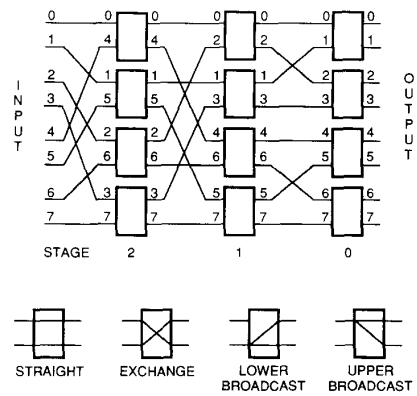


Fig. 1. Multistage cube network for $N = 8$.

of multistage cube networks [26]. It has $n = \log_2 N$ stages, numbered from 0 to $n - 1$, where each stage consists of a set of N lines ("links"), numbered from 0 to $N - 1$, connected to $N/2$ interchange boxes. Each "interchange box" is a two-input two-output device that can be set as shown in Fig. 1. The labels of the links entering the upper and lower inputs of an interchange box are used as the labels for the upper and lower outputs, respectively. At stage i, links whose numbers differ only in the ith bit position are paired at interchange boxes. PE j is attached to network input j and output j.

The name "multistage cube network" will be used to refer to the network consisting of the Generalized Cube topology and interchange boxes with the capabilities shown in Fig. 1, where each interchange box is controlled independently. The term multistage cube has its origin in the multidimensional cube network. In a "multidimensional cube" network, the vertices can be labeled in binary so that vertices whose labels differ only in bit position i are connected across dimension i. This pairing of vertices that differ in bit position i along dimension i corresponds to pairing links whose labels differ only in bit position i at an interchange box in stage i of the multistage cube network. This demonstrates the relationship of the multistage cube to the hypercube (multidimensional cube) networks used in such systems as the Connection Machine [56], Intel iPSC cube [62], and NCube [63].

B. Establishing Paths

"One-to-one connections" use the straight and exchange interchange box settings. To go from a source $S = s_{n-1} \cdots s_1 s_0$ to a destination $D = d_{n-1} \cdots d_1 d_0$, the stage i interchange box in the path from S to D should be set as follows:

if $d_i = s_i$ then the interchange box is set to straight, and if $d_i = \bar{s}_i$ then the interchange box is set to exchange. For example, if $s_2s_1s_0 = 110$ and $d_2d_1d_0 = 000$, then the interchange boxes are set as shown in Fig. 2. The link that a message from S to D uses is $s_{n-1} \cdots s_1s_0$ before stage $n-1$, $d_{n-1}s_{n-2} \cdots s_1s_0$ after stage $n-1$, $d_{n-1}d_{n-2}s_{n-3} \cdots s_1s_0$ after stage $n-2$, $\cdots$, and $d_{n-1} \cdots d_1d_0$ after stage 0. In general, the link that a message from S to D uses after stage i is $d_{n-1} \cdots d_{i+1}d_is_{i-1} \cdots s_1s_0$. For the example in Fig. 2, the links traversed are $s_2s_1s_0 = 110$ at the input, $d_2s_1s_0 = 010$ after stage 2, $d_2d_1s_0 = 000$ after stage 1, and $d_2d_1d_0 = 000$ after stage 0. Because the stage i output link used in the path from S to D must be $d_{n-1} \cdots d_{i+1}d_is_{i-1} \cdots s_1s_0$, there is only one path from a given source to a given destination.

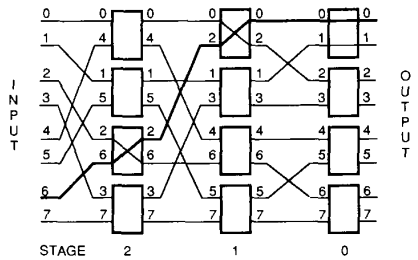


Fig. 2. Path from input 6 to output 0 in multistage cube network for $N = 8$.

This unique path property limits the fault tolerance of the Generalized Cube topology in that a single network fault will prevent some source/destination pairs from being able to communicate. Design techniques for fault-tolerant variations of the Generalized Cube are surveyed in [38].

A "conflict" occurs in a multistage cube network when the messages on the two input links of an interchange box want to go out the same output link. Typically, when a situation like this arises, one message is blocked and must wait until the other has completed its transmission. Both requests cannot be accommodated simultaneously. This is discussed further in Section IV.

A "broadcast (one-to-many) connection" is performed when the lower and/or upper broadcast states of interchange boxes are used in a path. For example, in Fig. 3 input 0 broadcasts to outputs 4, 5, 6, and 7.

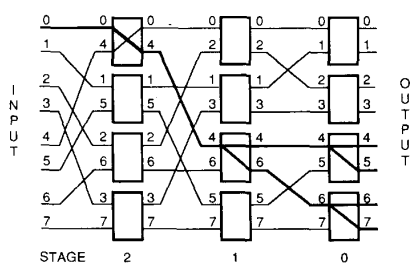


Fig. 3. Broadcast path from input 0 to outputs 4, 5, 6, and 7 in multistage cube network for $N = 8$.

C. Distributed Routing Tag Control

Network control for the multistage cube is distributed among the PEs by using a routing tag as the header (first item) of each message to be transmitted through the network. For one-to-one (nonbroadcast) connections, an n-bit routing tag can be computed by the source PE from its number $S = s_{n-1} \cdots s_1s_0$ and desired destination PE number $D = d_{n-1} \cdots d_1d_0$. The routing tag $T = t_{n-1} \cdots t_1t_0 = S \oplus D$ (where "$\oplus$" means bitwise "exclusive-or") is called the "XOR (exclusive-or) routing tag" [11]. When an interchange box in the network at stage i receives a message, it examines bit t_i of the tag. If $t_i = 0$, the straight connection is used, because $t_i = 0$ implies $d_i = s_i$. If $t_i = 1$, an exchange is performed, because $t_i = 1$ implies $d_i = \bar{s}_i$. If $N = 8$, $S = 6 = 110$, and $D = 0 = 000$, then $T = 110$, and the corresponding interchange box settings from input to output are exchange ($t_2 = 1$), exchange ($t_1 = 1$), and straight ($t_0 = 0$), as shown in Fig. 2.

The same tag used to route data from S to D can be used to route data from D to S because the exclusive-or operation is commutative (i.e., $S \oplus D = D \oplus S$). This can be used to "handshake" (confirm receipt of the data). Because each network destination D knows its own address (output port number), from the tag it can compute the source address S (input port number) that sent it data: $S = T \oplus D$.

Another approach to routing tags is the "destination tag" scheme, where the destination $D = d_{n-1} \cdots d_1d_0$ is the tag sent as the header [34]. The upper output link of a stage i interchange box always has a 0 in the ith bit position of its label, while the lower output link always has a 1. Therefore, taking the upper output link leads to a destination where $d_i = 0$, while taking the lower output link leads to a destination where $d_i = 1$. Therefore, when an interchange box in the network at stage i receives a message, it examines d_i: if $d_i = 0$ the upper output is taken, if $d_i = 1$ the lower output is taken. For the example from source 6 to destination 0 in Fig. 2, the upper output of the stage 2 interchange box is taken ($d_2 = 0$), the upper output of the stage 1 interchange box is taken ($d_1 = 0$), and the upper output of the stage 0 interchange box is taken ($d_0 = 0$). This same destination tag ($D = 000$) can be used to route data from any input to output 0.

The advantages of the destination scheme over the XOR method are that it is easier to compute and a destination PE can compare the destination tag that arrives against its own address to determine if the message arrived at the correct network output (if it did not, the network must be faulty). The destination tag scheme has the disadvantage that it cannot be used to determine the source; the XOR scheme can. Sending the source address along with the destination tag or sending the destination address along with the XOR tag are methods that provide the capability to determine both the source of the data and if the data arrived at the proper destination. The destination tag scheme is more practical, while the XOR scheme is more mathematically pleasing (and is used in section VIII for the single-stage variant). A broadcast routing tag scheme that consists of an n-bit broadcast mask along with either type of n-bit routing tag can be used to specify a variety of broadcast connections (but not all those physically possible) [6], [11], [67].

D. Partitionability

The "partitionability" of the multistage cube network is the ability to divide the network into independent subnetworks of different sizes so that each subnetwork of size

$N' \leq N$ has all of the interconnection capabilities of a multistage cube network built to be of size N'. The methods for partitioning the multistage cube network assume a PE-to-PE configuration, where PE i is connected to both network input i and network output i, so both input i and output i must belong to the same partition [6], [26], [66], [68]. These methods can also be used to partition the processor-to-memory configuration, but there are some partitionings that will support the processor-to-memory configuration and not the PE-to-PE.

The ability to partition an interconnection network into independent subnetworks implies that the network can support the partitioning of the system for which it is providing communications. Partitioning is necessary to support multiple-SIMD and partionable SIMD/MIMD systems. It can be used to partition MIMD systems and, in some cases, improve the efficiency of SIMD machines [6, pg. 111]. The advantages of partitionable systems [20] include fault tolerance (if a processor fails, only those partitions that include the failed processor are affected), fault detection (for situations where high reliability is needed, more than one partition can run the same program on the same data and compare results), multiple simultaneous users (because there can be multiple independent partitions, there can be multiple simultaneous users of the system, each executing a different parallel program), program development (rather than trying to debug a parallel program on, for example, for example, 1024 PEs, a user can debug the program on a smaller size partition of 32 PEs and then expand it to 1024 PEs), efficiency (if a task requires only $N/2$ of N available processors, the other $N/2$ can be used for another task), and subtask parallelism (two or more independent parallel subtasks that are part of the same job can be executed in parallel, sharing results if necessary).

Consider partitioning a multistage cube of size N into two independent subnetworks, each of size $N/2$. There are n choices for doing this, each one based on a different bit position of the input/output port addresses. One choice is to set all interchange boxes in stage $n - 1$ to the straight state. This forms two subnetworks, one consisting of input/output ports 0 to $(N/2) - 1$ (those with a 0 in the high-order bit position of their addresses) and the other consisting of ports $N/2$ to $N - 1$ (those with a 1 in the high-order bit position). These two disjoint sets of input/output ports could communicate with each other only by using the exchange in stage $n - 1$. By setting this stage to straight, the subnetworks are independent and have full use of the rest of the network (stages $n - 2$ to 0). This is shown in Fig. 4.

Because each subnetwork has the properties of a multistage cube, it can be further subdivided. Assume the size $N/2$ subnetworks were created by setting stage j to straight, $0 \leq j < n$. A size $N/2$ subnetwork can be divided into two size $N/4$ subnetworks by setting all the stage i interchange boxes in the size $N/2$ subnetwork to straight, for any i, $0 \leq i < n, i \neq j$. Partitioning one subnetwork into halves based on bit position $n - 2$ is shown in Fig. 5. This process of dividing subnetworks into independent halves can be repeated on any existing subnetworks (independently) to create any size subnetwork from one to $N/2$. The sizes of the subnetworks may differ. The only constraints are that the size of each subnetwork must be a power of two, each input/output port can belong to at most one subnetwork, the physical addresses of the input/output ports of a subnetwork of size 2^s must all agree in any fixed set of $n - s$ bit

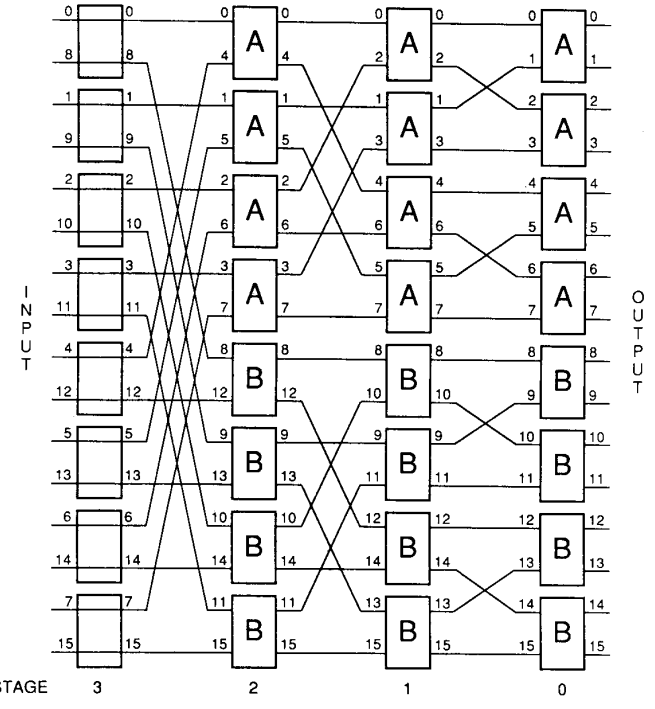


Fig. 4. Partitioning the multistage cube for $N = 16$ into two subnetworks of size eight (A and B) based on high-order bit position.

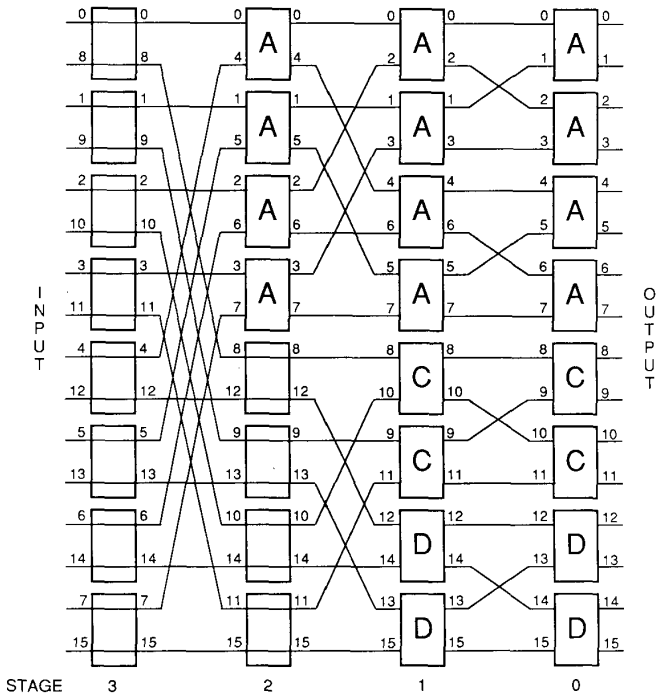


Fig. 5. Partitioning the multistage cube for $N = 16$ into one subnetwork of size eight (A) and two subnetworks of size four (C and D).

positions, and the interchange boxes used by this subnetwork are set to straight in the $n - s$ stages corresponding to these $n - s$ bit positions (the other s stages are used by the subnetwork to form a multistage cube network of size 2^s).

The routing tag schemes described in section III-C can be used within partitions. In fact, these routing tag schemes can be used to enforce the partitioning, assuming there exist some privileged instructions with which the operating system can set a mask each time a partition is initially created.

Consider enforcing a partition with routing tags using the XOR tag routing scheme. For a partition where stages $i_0, i_1, \cdots, i_k$ are set to straight, form a "partition mask M" of all 1's, except for 0's in bits $i_0, i_1, \cdots, i_k$. The PE generated XOR tag, T, is logically ANDed bitwise with the partition mask M to form the "effective tag E" that is sent into the network. For example, with $N = 1024$, to form a partition consisting of PEs 0, 1, 2, $\cdots$, 255, $M = 0011111111$ and $E = (T \text{ AND } M) = 00t_7 \cdots t_1 t_0$, forcing the stages 9 and 8 interchange boxes used to be set to straight.

For networks with destination tag routing, an effective tag $E = e_{n-1} \cdots e_1 e_0$ enforces the partition where, for $0 \le j < n$:

$$e_j = \begin{cases} d_j & j \notin \{i_0, i_1, \cdots, i_k\} \\ s_j & j \in \{i_0, i_1, \cdots, i_k\}. \end{cases}$$

E can be generated by evaluating the following logical expression:

$$E = D \cdot M + S \cdot \overline{M}$$

where $\cdot$ is bitwise logical AND and $+$ is bitwise logical OR. Using the preceding example, but with destination tag routing, the effective tag is $E = s_9 s_8 d_7 \cdots d_1 d_0$, again forcing the stages 9 and 8 interchange boxes used to be set to straight.

E. Interchange Box Size

Throughout this section it has been assumed that the network is constructed from 2×2 interchange boxes. Consider for N a power of b using $b \times b$ crossbars as the interchange boxes. All of the properties described in this section for the 2×2 case can be adapted for the $b \times b$ case by using base b arithmetic as the basis instead of binary. For example, the b links that differ in the ith digit of their base b representation will enter the same interchange box at stage i, $0 \le i < \log_b N$.

IV. Modes of Operation for Interchange Boxes

The paths through the multistage cube and routing tags for determining these paths were discussed in section III. In this section, different operational modes that can be used by the interchange boxes to establish paths and transmit data over these paths are described.

A. Circuit and Packet Switching

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. Data is sent directly from the source to the destination over this circuit. Circuit switching must be used in networks constructed from combinational logic where there are no buffers in the interchange boxes for storing data.

Different strategies can be employed in a circuit-switched network when a conflict occurs during path establishment [69]. Under the "hold" algorithm the blocked path request remains pending (holds) at the blocking interchange box until the path that is blocking its progress is relinquished. Then the previously blocked request (which is holding) can proceed through the network. Using the "drop" algorithm, a path request is immediately dropped when a conflict is encountered. A new attempt can be made at a later time to establish the path. The tradeoff is between a dropped request having to re-request the blocked subpath already established versus a held request possibly blocking other paths with that blocked subpath. A hybrid of the two is the "modified hold" algorithm. A blocked request is held for a predetermined length of time before it is dropped under the modified hold algorithm. One advantage of the modified hold algorithm is that paths blocked for short periods of time are not dropped and do not incur the added delay associated with requesting the same path again.

In the "packet-switched mode," the routing tag and data to be transmitted are collected together into a "packet." A packet consists of one or more words and can be of fixed or variable size. Packet switching uses data buffers in each interchange box to store packets as they move through the network. 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 n interchange boxes, one from each stage, are used simultaneously for the entire duration of the message transmission.

One way to handle conflicts in a packet-switched network is to make one packet wait until the other is transmitted. The "wait" is implemented by storing the packet in the interchange box's packet buffer. When an interchange box's packet buffer is full, it will not accept new packets from the previous stage. Networks that employ packet switching and can store multiple packets at each interchange box are often referred to as "buffered networks."

Wormhole routing [70] and virtual cut-through routing [71], [72] are hybrids of circuit and packet switching. "Wormhole routing" differs from packet switching in that an entire packet is not stored in an interchange box before being forwarded to the next interchange box. Instead, an interchange box forwards a word of a packet to the appropriate interchange box in the next stage of the network immediately after the word is latched at one of its input ports. The contents of each packet are thus pipelined through the network [68]. As packet words are forwarded through the network, the packet becomes spread across many interchange boxes. When the header of a packet is blocked, all the words of that packet stop advancing. Any other packets requiring the use of an interchange box output port for which any blocked packet word is waiting are also blocked, analogous to the hold circuit-switching protocol. "Virtual cut-through" routing is similar to wormhole routing except that when the header gets blocked, the other words from the packet continue to advance and are buffered in the interchange box that contains the blocked packet header.

Thus there are a variety of choices for switching techniques (e.g., circuit switching, packet switching, or their variations of cut-through and wormhole routing) and routing strategies (e.g., drop, hold, and modified hold algorithms). The routing tags described in section III-C are appropriate for operating the network in any of these modes of operation. Details of communication protocols and interchange box physical implementations are discussed in the literature (e.g., [11], [70]–[75]).

There are a large number of factors that can affect the relative performance of the different switching techniques. These factors include implementation technology, communication protocols, size of the network (i.e., number of I/O ports), size of interchange box (e.g., 2×2 versus 4×4), if the message size is fixed or variable (and if variable, the distribution of the sizes), width of each path through the network, network load (the amount of data input into the network at a given time, which is affected by the processor speed, memory access time, actual application program being executed, etc.), the extent to which paths being established through the network conflict, the modes of parallelism (SIMD, MIMD, or both) in which the network is being used, cost constraints, and transfer time requirements. Because of the many variations in switching techniques and the large number of parameters that can affect network performance, determining which switching technique is "best" (even with a subset of the parameters fixed) is extremely difficult and is a problem currently under study by many researchers (e.g., [70], [76]). An interchange box implementation that allows both packet- and circuit-switching capabilities is discussed in [4], [77]. Interchange boxes designed to support fiber optics communication are described in [78].

B. Hot Spots and Combining

This subsection assumes that the processor-to-memory configuration (described in section II-E) uses a bidirectional multistage cube (as opposed to two unidirectional networks). The techniques can be adapted to hybrid configurations. Also, packet switching, as discussed in the preceding, is assumed.

The strategies just reviewed work well assuming that requests are randomly directed at the outputs, which is usually a reasonable assumption. In fact, the hardware or software can hash (map) logical to physical memory locations, thereby guaranteeing that requests look random and are spread across memory modules. There is, however, one exception: multiple memory requests can be directed at the same memory word. Such requests will at the very least have to queue up at the associated memory module, which then becomes a bottleneck. Furthermore, these requests to the same memory word will start to back up into the network, not only causing requests to that module to take a long time to be serviced, but also blocking and therefore slowing down requests to other modules. This phenomenon has been termed "hot spots" [79] and has also been studied by various researchers (e.g., [80]–[82]).

To avoid hot spots, one can use a technique called "combining." When two load requests to the same memory location meet at an interchange box, they can be combined

without introducing extra delay by forwarding just one of the two (identical) loads and then satisfying both requests with the value returned from memory to that interchange box. This does assume that the load value will return to the processor via the same route through the multistage cube network (but in the reverse direction) that the load request used to get to memory. This gives a machine the ability to satisfy a large number of loads to the same location in about the same time it takes to satisfy one such load. The combining idea can be generalized to handle stores and combinations of loads and stores [24]. When a load meets a store, forward the store and return its value to satisfy the load. When a store meets a store, forward either store and ignore the other.

Asynchronous parallel machines typically have synchronization primitives such as test-and-set [49]. A more powerful synchronization primitive is the "fetch-and-add." The format of this operation is F&A(V, e), where V is an integer variable and e is an integer expression. This indivisible operation returns the (old) value of V and replaces V by the sum $V + e$. By including adders at the interchange boxes, fetch-and-adds can be combined [24]. This allows multiple processors of a machine to perform simultaneously operations, such as fetching unique index values for a job queue, in about the same amount of time as it takes for one processor to access memory. The disadvantage is that the interchange boxes become complicated, which may make them too slow and/or costly. It is claimed in [24], however, that using 1990's technology, any on-chip delays will be negligible compared to chip-to-chip delays. The idea of combining adds easily generalizes to any associative operation [24]. In fact, many nonassociative "read-modify-write" operations can be combined (relatively) efficiently [83].

The cost-effectiveness of different types of combining will depend on the task domain and operating system. Under what conditions to build in combining and which types of combining to use are currently open questions.

V. Performance in SIMD Mode

This section considers the ability of the multistage cube to permute data in an SIMD environment. Recalling that the input and output ports of a network are numbered from 0 to $N - 1$, a "permutation" of data among the PEs is mathematically representable as a bijection from the set $\{0, 1, \cdots, N - 1\}$ onto itself. That is, if a permutation f maps i to $f(i)$, $0 \le i, f(i) < N$, then PE i transfers data to PE $f(i)$. This occurs for all enabled PEs simultaneously.

In general, there are $N!$ ways to construct such permutations. However, not every permutation can be realized by a network in a single network transfer. A permutation is said to be "passable" by an interconnection network when all N inputs can send data to the appropriate N outputs simultaneously without any conflicts occurring; i.e., PE i can send data to PE $f(i)$, $\forall i$, simultaneously, without any conflicts. The routing tags described in section III-C can be used to establish the paths of any passable permutation. An example of the multistage cube network set to the permutation input j to output $j - 1$ modulo N, $0 \le j < N$, is shown in Fig. 6.

In the following, the permuting power of the multistage cube network with respect to the permutations that are passable is discussed. Performing arbitrary permutations with the multistage cube is also overviewed.

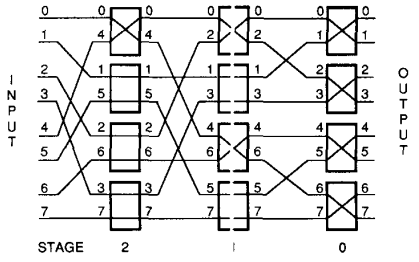


Fig. 6. Permutation input j to output $j - 1$ modulo N, $0 \le j < N$, in the multistage cube for $N = 8$.

Consider the permuting power of the multistage cube network. When routing a permutation, each of the $Nn/2$ interchange boxes is either in the straight state or in the exchange state. Because each of the $Nn/2$ individual interchange boxes can assume two separate states (and no two distinct settings of all the interchange boxes yields the same permutation), the total number of unique switch settings, and thus permutations, possible with the multistage cube is $2^{Nn/2} = N^{N/2}$. In general, $N^{N/2} \ll N!$; e.g., for $N = 8$, $N^{N/2} = 4096$, while $N! = 40\,320$. However, the multistage cube network has been shown to perform most of the permutation types frequently needed in parallel algorithms [32], [34], [84].

To demonstrate the types of permutations passable by the multistage cube, some particular types will be defined. Groups of useful permutations that arise from patterns of communication seen in parallel implementations of algorithms such as FFT, matrix operations, and divide-and-conquer are characterized in [85]. These "frequently used permutations" have been generalized into families. Let $X = x_{n-1}x_{n-2} \cdots x_1x_0$ be an arbitrary I/O port number, $0 \le X < N$. Three of the basic permutation groups are

perfect shuffle

$$\sigma^{(n)}(x_{n-1}x_{n-2} \cdots x_1x_0) \to x_{n-2} \cdots x_1x_0x_{n-1}$$

bit reversal

$$\rho^{(n)}(x_{n-1}x_{n-2} \cdots x_1x_0) \to x_0x_1 \cdots x_{n-2}x_{n-1}$$

cyclic shift of amplitude k

$$\pi_k^{(n)}(X) \to X + k \bmod 2^n$$

These basic permutation groups were used in [85] to represent families of permutations frequently used in SIMD processing.

The first of these families is the $\lambda_{j,k}^{(n)}$ family that connects input X to output port $jX + k \bmod 2^n$ (j odd), $0 \le X < N$. These permutations are used to access rows, columns, diagonals, and blocks of matrices where the matrices are stored across the memories of a parallel processing system. The $\lambda_{1,k}^{(n)}$ permutations are the same as $\pi_k^{(n)}$.

The $\delta_{j,k}^{(n)}$ permutation family dictates the cyclic shift of data within segments of size 2^j. Let $\hat{a}_1$ and $\hat{a}_0$ be bit substrings of $X = x_{n-1}x_{n-2} \cdots x_1x_0$, where $\hat{a}_1 = x_{n-1}x_{n-2} \cdots x_j$ and $\hat{a}_0 = x_{j-1}x_{j-2} \cdots x_0$. Then $\delta_{j,k}^{(n)}$ represents the permutation

input port $(\hat{a}_1, \hat{a}_0) \to$ output port $(\hat{a}_1, \pi_k^{(j)}(\hat{a}_0))$ $(0 \le j < n)$.

These permutations can perform cyclic shifts within halves of the network, quarters of the network, etc. "Divide and conquer" algorithms utilize these permutations frequently.

The remainder of these frequently used bijections from [85] are generalizations of the bit reversal permutations within segments and the perfect shuffle permutations within segments. These permutations are useful in FFT computations, for example.

The multistage cube network can perform the $\lambda_{j,k}^{(n)}$ and $\delta_{j,k}^{(n)}$ permutations in one pass, but cannot route the perfect shuffle or the bit reversal permutations in one pass [85]. Based on the results in [32] it can be shown that the multistage cube can perform the bit reversal permutation in two passes. The routing tags needed to do this can be precomputed and stored. It is possible for the multistage cube network to pass the perfect shuffle (i.e., $\sigma^{(n)}$ $(x_{n-1}x_{n-2} \cdots x_1x_0)$) in two passes if, in the first pass, only those input ports numbered $X = 0x_{n-2} \cdots x_1x_0$ send data into the network. Then, in the second pass, those ports numbered $X = 1x_{n-2} \cdots x_1x_0$ send data into the network. The routing tags to do this can be precomputed or computed dynamically, using the methods given in section III-C. As described in section III-D, the multistage cube network can be partitioned into independent subnetworks, each with the connectivity of a full multistage cube network of that smaller size [66]. This property allows the multistage cube to perform the bit reversal and perfect shuffle permutations within segments in two passes.

The "universality" of a network is the ability of that network to route an arbitrary permutation [86]. From the results in [87], it can be shown that the multistage cube network can perform any permutation in three passes for all $N = 2^n$. Two passes are necessary, three passes are sufficient. Showing that the lower bound of two passes is both a necessary and sufficient number of passes to route any permutation is an open problem for $N > 8$. Control schemes for routing an arbitrary permutation in two or three passes are not known; i.e., a technique for computing, without centralized control, the routing tags for each pass is not known.

VI. Performance in MIMD Mode

The MIMD performance of multistage cube networks is analyzed under the following two assumptions:

1) at each cycle, packets are generated at each source independently with probability p;
2) each packet is sent with equal probability to any destination.

The packet "rate," i.e., the number of packets issued per cycle by a PE, is p, where $p \leq 1$. The "traffic intensity per processor" is loosely defined as the packet rate p times the average packet length. The "bandwidth per processor" is the largest traffic intensity per processor that the network can support. The "bandwidth of a network" is N times the bandwidth per processor. The "delay" is the average time for a packet to reach its destination.

One of the most important performance measurements of an interconnection network is the delay for a given traffic intensity. Ideally, one desires a network that handles heavy traffic with small delay. However, these two quantities must generally be traded off: to obtain small delay, the traffic has to be relatively light, and to handle heavy traffic, the delay has to be relatively large.

In this section it is assumed that a multistage cube network with N input and output ports is composed of $k \times k$ interchange boxes ($k \times k$ crossbar switches, as mentioned in section III-E). Let the number of stages in the network be $\eta = \log_k N$. The stages are numbered 0 to $\eta - 1$, from first to last (i.e., the reverse of the Generalized Cube topology of section III-A, as if the stages are traversed from output to input).

A. Dilated and Replicated Networks

There are many possible enhancements to the basic multistage cube network defined in section III. Two such enhancements are dilation and replication [88]. The "d-dilation" of a network G is the network obtained by replacing each connection by d distinct connections. This is shown in Fig. 7. A request entering an interchange box may exit

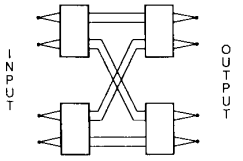


Fig. 7. Two-dilation for multistage cube for $N = 4$.

using any of the d connections going to the desired successor interchange box at the next stage. The "d-replication" of a network G is the network consisting of d identical distinct copies of G, with the d corresponding input nodes and the d corresponding output nodes connected to the same PE. This is shown in Fig. 8. Both enhancements improve reliability and performance, as discussed in the following.

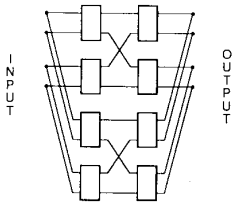


Fig. 8. Two-replication of multistage cube network for $N = 4$.

B. Unbuffered Packet-Switching Networks

First, consider packet-switching networks built of $k \times k$ unbuffered interchange boxes, where only one word can be stored at an input link to the interchange box. Recall that for purposes of analysis, at each cycle, each PE generates a packet with probability p. To simplify the analysis it is assumed that when multiple packets entering an interchange box (on different interchange box input ports) are routed to the same interchange box output, one randomly chosen packet is transferred out and the others are dropped (i.e., they are deleted from the network and would have to be retransmitted).

The relevant figure of merit for such networks is the probability p_i that there is some packet on any particular input at the ith stage of the network. This quantity will immediately yield the bandwidth; it will be used later to approx-

imate the delay. In [30] the following recurrence is derived:

$$p_i = 1 - \left(1 - \frac{p_{i-1}}{k}\right)^k \tag{1}$$

with the boundary condition $p_0 = p$ (where p is the probability of packet creation at a source node). To see this, note that p_{i-1}/k is the probability that a packet exists at a particular input of stage $i - 1$ and is directed to a particular output, so $1 - p_{i-1}/k$ is the probability that from a particular input a packet is not directed to that particular output, so $(1 - p_{i-1}/k)^k$ is the probability that no packet is directed to that particular output from any input, so $1 - (1 - p_{i-1}/k)^k$ is the probability that some packet is directed to that particular output from some input. This recurrence allows one to compute numerically the value of p_i (for any initial p_0).

In [88], asymptotic formulas for p_i are derived and it is shown that p_i can be approximated as

$$p_i \cong \frac{1}{\frac{(k-1)i}{2k} + \frac{1}{p}}. \tag{2}$$

This closed-form approximation provides insight as to how p_i behaves. It is shown in [89] that this is actually an upper bound for p_i, and a (much more complicated) lower bound is presented. The bandwidth of a network is $N_{p\eta}$ (where $\eta = \log_k N$). In [88] it is also shown that the bandwidth of a network is asymptotically

$$\frac{2kN}{(k-1)\log_k N},$$

as $N \to \infty$, for any fixed p_0. Thus the bandwidth per processor decreases logarithmically with network size.

For comparison, consider the bandwidth of a crossbar network (discussed in section I-A). Here the only bottleneck is the ability of the destination to handle the packets. In fact, a crossbar can be modeled as a single-stage (one interchange box) $N \times N$ network. By (1), the probability that a packet survives (is not blocked) a single-stage network (i.e., a crossbar) is $1 - (1 - p/N)^N$ (which asymptotically approaches $1 - 1/e^p$). Thus the bandwidth of a crossbar network is approximately $(1 - 1/e^p)N$. Fig. 9 compares the bandwidths of networks composed of 2×2, 4×4, and 8×8 interchange boxes with each other and with a crossbar network for various network sizes (N) up to 2^{24}. (The bandwidths are computed by setting $p = 1$ (a worst-case situation that is unlikely to occur) and using (1).) The performance graph in Fig. 9 is for comparative purposes; recall that an $N \times N$ crossbar network uses N^2 switches and is therefore impractical to build for $N \geq 64$ given current technology.

A similar recurrence relation can be calculated for dilated networks [88]. The recurrence is much more complicated than in the preceding, and difficult to solve asymptotically. By making a simplifying assumption, an approximate formula for the performance of unbuffered d-dilated networks based on 2×2 interchange boxes is derived in [89]. An interesting conclusion from their formula is that the bandwidth per processor of a d-dilated network[1] is

$$\Theta\left(\frac{1}{(\log N)^{1/d}}\right).$$

So a small increase in dilation can significantly improve bandwidth. Recently this formula was proved formally in [90] (removing the simplifying assumption of [89]). This is also generalized to $k \times k$ interchange boxes in [89].

The performance of a d-replicated network is easy to approximate: it consists of d copies of a multistage cube network, so (1) (and related asymptotic results) apply to each copy independently. Assume that every source issues one packet with probability p at each cycle and randomly sends it to one of the d copies. Making the simplifying assumption that the d copies are independent, the probability that a packet exists after stage η is $1 - (1 - p_\eta)^d$, where $p_0 = p/d$ [88].

Using these equations, one can compare the bandwidth of dilated and replicated networks using comparable hardware [88]. For practical sizes, dilated and replicated networks with comparable cost provide approximately the same performance. For extremely large (and impractical) sizes, dilated networks are superior to replicated networks.

[1] $f(N) = \Theta(g(N))$ essentially means there are two constants a and b such that $ag(N) \leq f(N) \leq bg(N)$, e.g., $(2N + 5) = \Theta(N)$.

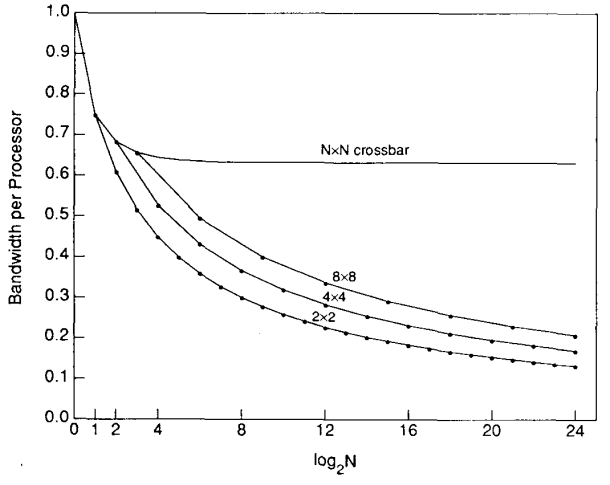


Fig. 9. Comparison of bandwidth per processor of multistage cube networks of size N built from 2×2, 4×4, 8×8 interchange boxes and $N \times N$ crossbar network.

It will now be shown how to approximate the delay of a packet. For simplicity, assume that a dropped packet is reissued from its original source to a random location rather than back to the same location that it just attempted but failed to access (as one would expect in real life). Simulations show that this is a reasonable approximation, but it will produce slightly optimistic results.

As usual, assume new packets are issued at rate p. Let q be the rate that dropped packets return to a source. Then the effective rate of packets is $p + q$. Make the simplifying assumption that at each cycle a single packet is issued by a given processor with probability $p + q$. By (2), successful packets exit at a destination at a rate that is approximately

$$1\Big/\left(\frac{(k-1)\eta}{2k} + \frac{1}{p+q}\right).$$

For the system to be in equilibrium, new packets must enter the system at the same rate that successful packets exit the system, so

$$p \cong 1\Big/\left(\frac{(k-1)\eta}{2k} + \frac{1}{p+q}\right).$$

Solving for q gives

$$q \cong \frac{\eta p^2}{\frac{2k}{k-1} - \eta p}.$$

The probability s that an issued packet survives, whether new or previously dropped, is the probability that a packet exits the system divided by the probability that a packet enters the system (as a new packet or reissued previously dropped packet). Thus, using this fact and the approximation for q in the preceding,

$$s \cong \frac{\left\lfloor 1\Big/\left(\frac{(k-1)\eta}{2k} + \frac{1}{p+q}\right)\right\rfloor}{p+q} \cong 1 - \frac{(k-1)p\eta}{2k}.$$

The expected number of times τ that a packet is issued or reissued is one over the probability that a packet survives. So

$$\tau \cong \frac{1}{s} \cong \frac{1}{1 - \frac{(k-1)p\eta}{2k}} = 1 + \frac{p\eta}{\frac{2k}{k-1} - p\eta}.$$

A packet will always require at least one attempt to traverse the system (its final and successful attempt). Note that p must be less than $2k/((k-1)\eta)$, which is the bandwidth per processor, otherwise the network will be unstable (i.e., will saturate). To fully interpret these equations, one must make further assumptions about the hardware. As long as p is somewhat less than $2k/((k-1)\eta)$, the expected number of resubmissions will be bounded by a constant. The time between resubmissions depends on how soon a dropped packet can be reissued, but anywhere between a few cycles and a few η cycles seems reasonable. Even with the latter conservative assumption, the delay for a message is $\Theta(\eta)$.

C. Circuit-Switching Networks

As discussed in section IV-A, there are several variants of circuit-switching networks. In a synchronous network in which each source attempts simultaneously with probability p (at the beginning of each cycle) to establish a communication path with a randomly chosen destination, the preceding analysis of unbuffered packet-switching networks applies. For asynchronous networks (where messages are not necessarily issued simultaneously), the hold and drop schemes are extremely difficult to analyze in closed form. However, they can be compared qualitatively: the advantage of the hold scheme compared to the drop scheme is that messages do not waste time entering and backing out of the network. The disadvantage is that while a message is being blocked, it blocks other potential messages. In [69], the performance of circuit-switching networks is numerically approximated and their analyses are supported with simulations. It is found that for heavy traffic, the drop scheme is better than the hold scheme when the time to transfer a message is longer than about ten cycles.

D. Buffered Packet-Switching Networks

This subsection discusses the performance of buffered networks, where there are queues at the output ports of each interchange box to store blocked packets (see section IV-A). The "waiting time" of a packet at a stage is how long it spends in the buffer at the stage; the "delay" of a packet at a stage is the waiting time at the stage plus the length of the packet. The waiting time and delay of a packet through the entire network are similarly distinguished. Initially, only the waiting time is discussed; later the delay is analyzed by adding in the packet length.

The performance of a buffered interconnection network is very sensitive to the size of the buffers. Networks with small buffers (i.e., buffers that can store just a few packets) are difficult to analyze in closed form. Networks with moderate and large buffers have approximately the same behavior as networks with infinite buffers [24], which is advantageous because these latter networks are, mathematically, much more tractable. The following analyses assume infinite buffers.

Assume that at each cycle, a processor issues a packet with probability p, and each packet is c cycles long (i.e., it takes c cycles for the whole packet to pass through an interchange box, which is called the "service time"). Queuing theoretic analysis shows that the expected waiting time of a packet in the buffer at the first stage [88], [91] is

$$\frac{\left(c - \frac{1}{k}\right)cp}{2(1-cp)}. \tag{3}$$

The waiting time at later stages is not the same because the packets do not leave the buffers in the first stage or later stages independently (i.e., there is temporal dependence because, for example, if a packet has just finished exiting a buffer, it is more likely that a new packet will start exiting the buffer). The waiting time at later stages can be approximated [91] by

$$\frac{\left(1 + \frac{4cp}{5k}\right)\left(c - \frac{1}{k}\right)cp}{2(1-cp)}.$$

There is an interesting and important point that can be learned from these equations (this point has been made in [24], [88], [91] but does not seem to be widely appreciated).

When packets are large (i.e., take many cycles to service), unless a network is very lightly loaded it is desirable to split packets into several smaller packets, each directed to the same location, despite the fact the same routing information will have to be duplicated on all of the smaller packets. Similarly, it is not desirable to bunch together several packets directed to the same location in order to preclude each packet carrying the same routing information. This follows immediately from the preceding two formulas on waiting time at an interchange box: for the same traffic intensity (cp), the delay at each interchange box grows linearly in the packet service time (c).

In [91] there are also general formulas for the expected value and variance of the waiting time at a buffer in the first stage, and approximations for later stages. In particular, there are formulas for various combinations of nonsquare ($a \times b, a \neq b$) interchange boxes, bulk arrivals, nonuniform traffic, and multiple service times.

As noted at the beginning of this subsection, the delay of a packet at an interchange box is its waiting time plus its service time, and the delay of a packet through the entire network is the sum of its waiting times through all of its interchange boxes plus its total service time. If a network does not pipeline packets, the total service time is ηc, so the total delay is the sum of the delays at each stage plus ηc. If the network pipelines packets using cut-through, the total service time is $\eta + c - 1$, so the total delay is the sum of the delays at each stage plus only $\eta + c - 1$. Related ideas were discussed in [68].

The bandwidth per processor of a buffered multistage cube network with packets with service time c is $1/c$, independent of the interchange box size. For comparison, a buffered crossbar network is simply a buffered network consisting of one stage of one $N \times N$ interchange box. The delay is obtained from (3) by substituting N for k. Equation (3) shows that each stage of a buffered network has, up to about a factor of two, the same delay as the one stage in a buffered crossbar network. So the main cause of delay in a buffered network is the number of stages.

Using the preceding formulas it is easy to analyze the delay in a replicated buffered network. When comparing replicated networks based on different interchange box sizes but containing comparable total hardware, for very light traffic 2×2 interchange boxes perform best and, as the traffic gets heavier, larger interchange boxes are better [88]. Dilated networks are much harder to analyze.

E. Network Comparisons

From a performance point of view, the actual choice of network depends on many factors, including machine size, traffic intensity, traffic pattern, message sizes, and implementation technology. The performance of like circuit-switching and unbuffered packet-switching networks differing only in their use of either dilation or replication has been studied. Also, the relative merits of the use of the hold and drop algorithms in circuit-switching networks have been established to some extent.

Circuit-switching and unbuffered packet-switching networks are easier and cheaper to build than buffered networks, while only buffered networks provide bandwidth proportional to the network size N. Thus, in a general sense, it appears that buffered networks are better for very large machines and the other networks are better for small machines. This qualitative analysis is supported quantitatively in [92], where the performance of interconnection networks is approximated using Markov chains. They compare unbuffered networks, networks with buffers of size one, and networks with infinite buffers, and conclude that buffering significantly improves performance.

As mentioned in section I-A, what network is "best" depends on a great many factors. This is also true when selecting an implementation for a multistage cube network. Therefore, when one reads a study of network performance, one must take great care to understand all of the assumptions made about these factors in that study. For someone designing a new system, these assumptions must be compared to the values for these factors expected in the new system. For someone conducting interconnection network research, these assumptions must be compared to one's own view of what is reasonable.

Because of all of the different factors involved, no easy way currently exists to predict the "best" implementation for any given set of actual operating conditions or range of conditions (although some special cases may be known). The development of a general prediction methodology such as this is a difficult and open problem.

VII. Case Studies

In this section some existing systems using multistage cube networks are briefly overviewed. These systems are ASPRO, Cedar, GP1000 (and TC2000), PASM, RP3, and Ultracomputer. These case studies demonstrate some of the different ways the multistage cube network is being employed. They also provide examples of operational speeds that can be attained. Information about these systems are from the references cited and from personal communications with principal people involved in each project.

A. The Goodyear Aerospace Corporation ASPRO

The Goodyear Aerospace Corporation ASPRO (ASsociative PROcessor) [54] is a next-generation evolution of the STARAN series of parallel processors of the 1970's [23]. The goal of the ASPRO was to put the processing power of a STARAN system on a single chip. Up to 16 array modules can be combined to form an array unit, where each array module consists of four SIMD machines. Each SIMD machine uses a processor-to-memory organization with 32 bit-serial processors and their associated 32×4096 bit array (plane) of multidimensional access storage. The ASPRO is intended for signal processing, decision processing, and scientific processing. Goodyear Aerospace Corporation is now known as Loral Defense Systems Division-Akron, and is still building ASPROs.

ASPRO uses multidimensional access (MDA) memory [93] with a flip network similar to the STARAN parallel processors. MDA memory permits horizontal and vertical (row and column) accesses to the bit plane of memory.

The flip network, located between the processors and the memory, is needed to implement MDA memory. The flip network is a multistage cube network with individual stage control, i.e., interchange boxes in a given stage are all set to exchange or all set to straight. Control of the flip network is centralized; routing tags are not used. Instead, for a flip network with $2^n = 32$ inputs, $n = 5$ control lines from the

central controller (one control line for each stage) set the network to any of 2^n permutations.

In the ASPRO, 32 processors along with a 32 line flip network are packaged on a VLSI chip. Each 32-processor VLSI chip is connected to two custom memory chips containing the memory for the 32 processors. The MDA is permitted among groups of 32 processors. A read access to the MDA memory takes 600 ns, including the time to access the memory and go through the flip network. All VLSI processor chips are connected to a common I/O bus. Unlike the STARAN systems, where the processors could use the flip network to pass data among themselves, processor connectivity in the ASPRO is limited to communicating through the MDA memory.

B. The Cedar System

Cedar [16] is a hybrid processor-to-memory configuration multiprocessor system under construction at the University of Illinois. In Cedar, multiple computing clusters are connected through a multistage cube network to a global shared memory. Each Cedar cluster is a slightly modified Alliant FX/8 minisupercomputer with eight 64-bit floating-point microprocessors. The global shared memory is intended to hold data that must be shared among clusters. A Cedar cluster has memory modules that provide local memory for the processors of the cluster. The cluster memory modules may be shared among the processors of that cluster. The prototype under construction will have 32 processors, while the architecture is designed to support 1024 processors.

Cedar's prototype multistage cube interconnection network is based on a unidirectional 8×8 interchange box (implemented as an 8×8 crossbar). Thus two unidirectional networks are required to have a path from the processors to the global shared memory and a path back from the global shared memory to the processors. Each unidirectional network has two stages of 8×8 interchange boxes. Two stages of 8×8 interchange boxes form a network with 64 inputs and 64 outputs. For the planned 32-processor prototype, either a subset of the links and/or interchange boxes that compose a full 64×64 will be used, or the full network will be used and path redundancy will be exploited for improved performance and fault tolerance [94]. The number of memory modules to be included in the shared global memory is yet to be finalized.

Packet switching is used to route packets through the interconnection networks. Packets are one, two, or three words long for read, write, and synchronization primitives, respectively. Each packet word is 64 bits wide. The remaining lines of each 80-bit network link are used for parity and control.

It takes one 85-ns cycle for a packet word to traverse an 8×8 interchange box. The access speed of the global memory is eight cycles but is pipelined to service a read/write request every four cycles. Thus a global memory read access takes 12 cycles (1020 ns): two cycles to traverse the two-stage network from the processor to global memory, eight cycles to retrieve the data, and two cycles to traverse the other two-stage network from global memory back to the processor. The 8×8 interchange box has input and output buffers that can hold two packet words. A three-word packet is held in two buffers. An interchange box consists of an 8×8 crossbar implemented by ECL gate arrays and interface/control logic constructed from off-the-shelf components.

A four-cluster machine, with two processors in each cluster, is operational. A system with 32 processors is expected by the third quarter of 1990. Further growth to larger numbers of processors is a possibility.

C. The GP1000 and TC2000 Systems

The GP1000 multiprocessor [13] is a commercial PE-to-PE parallel processor manufactured by BBN Advanced Computers, Inc. It is a logically shared memory space MIMD machine with up to 128 PEs that uses a multistage cube type of interconnection network called the "butterfly switch" for communication.

Each PE is a processor/memory pair based on the Motorola MC68020 microprocessor with four M-bytes of main memory. Although each PE processor has only a portion of the machine's total memory physically located in the PE, it "sees" the rest of the system's memory (i.e., other PEs' memories) in its address space. That is, it can access both local and nonlocal memory by placing a valid memory address on the PE's internal bus. If the address maps into the PE's own locally held portion of the system's memory, the appropriate memory transaction is carried out with no use of the network. For accesses that map to a portion of memory held in another PE, use of the network is required. Each PE has a processor node controller (PNC) that acts as the PE's intelligent interface to the butterfly switch. When a nonlocal access is made, the PNC captures the address placed on the local PE bus by the processor and determines which PE contains the target memory location. The PNC then forms a packet and sends it through the network to the remote PE. The PNC of the remote PE receives the packet and, in the case of a write access, updates the addressed location, or in the case of a read access, reads the addressed location. It sends a reply packet back to the originating PE's PNC. For the case of the read access, the data item contained in the reply packet is placed on the PE's internal bus and is read by the processor.

The GP1000 butterfly switch is unidirectional and uses 4×4 interchange boxes. Each interchange box is implemented on a single VLSI chip. A form of wormhole routing (section IV-A) routes packets through the butterfly switch by using the destination tag routing scheme (section III-C). However, blocked packets are immediately dropped from the network rather than kept waiting for an open channel. If a packet is blocked at any interchange box on the way to the destination PE, a reject signal is asserted back along the established portion of the path to the originating PE. This reject signal relinquishes the portions of the path already established and informs the originating PE that the packet has been rejected. All packets are long enough (or are padded so that they are long enough) such that the head of the packet reaches the destination PE before the last portion of the packet leaves the PE originating the packet. This is necessary to ensure that the reject signal reaches the originating PE before the end of the packet leaves its network interface. A blocked (rejected) packet is retransmitted after a random time delay.

Each path through the network is four bits wide and all interchange boxes are clocked at eight MHz (i.e., four bits of a packet get through each 4×4 interchange box in 125

ns). However, the potential bandwidth for a network path of 32 Mbits/s is not realizable due to the overhead of routing bits, control bits, checksums, etc. For example, with $N = 128$, the effective bandwidth of a network path ranges from 5.3 Mbits/s for packets containing 16 data bits of 24.4 Mbits/s for packets containing 256 data bits.

BBN has recently announced the TC2000 system [14], which is similar to the GP1000. Some of the differences germane to this discussion are as follows: the design can support over 500 PEs; each PE is a Motorola 88000 microprocessor with four or 16 M-bytes of memory; the network is bidirectional; 8×8 interchange boxes are used; each interchange box is implemented by gate array chips; each path through the network is eight bits wide; and all interchange boxes are clocked at 38 MHz (i.e., eight bits of a packet get through each 8×8 interchange box in 26 ns); the potential bandwidth for a network path is 304 Mbits/s; and the effective bandwidth of a network path for $N = 128$ ranges from 86.9 Mbits/s for packets containing 32 data bits to 187.1 Mbits/s for packets containing 128 data bits.

D. The PASM Parallel Processing System

PASM is a partitionable SIMD/MIMD parallel processing system design intended to support as many as 1024 PEs [20], [21]. PASM can be partitioned dynamically to work as one or more independent or cooperating submachines of various sizes. Each submachine can independently switch between SIMD and MIMD modes of operation at instruction-level granularity with negligible overhead. These features, in conjunction with the multistage cube type of interconnection network used, make PASM a highly reconfigurable architecture. A 30-processor prototype, with 16 PEs in the computation unit, has been constructed at Purdue University [21].

The PASM prototype inter-PE network is a fault-tolerant variation of the multistage cube called the extra-stage cube [37]. The network is circuit switched and uses the destination tag routing scheme. Each interchange box is a 2×2 crossbar implemented with standard TTL components. The path width of each network connection is 16 data bits plus two parity bits. Other features of the network hardware include correct destination verification on each path established and parity checking on all transmitted data.

The PE Motorola MC68000 CPUs communicate through the network with other PEs via their network interfaces. The network interface is a set of data and control registers used by the PE CPU to monitor and control its connection to the network (e.g., establish paths, transmit, and receive messages). The operating speed of the PE CPU is therefore the limiting factor in determining how fast the network must transfer data. The use of standard TTL components in the prototype network yield an inexpensive network that does not limit throughput on established paths. Assuming no conflicts, the time to establish a path from PE A to PE B in the PASM prototype is approximately 2 μs. (The goal of the prototype construction was to implement a tool for studying the attributes of such a reconfigurable architecture, not to maximize the raw speed of the prototype hardware.) Once a path is established, the PASM prototype network hardware can support a transfer rate of 24 Mbits/s, but is limited to a sustained transfer rate of 3.8 Mbits/s due to the processing rate of the PE CPU.

E. IBM RP3

IBM has built an experimental PE-to-PE logically shared memory parallel machine, called RP3 (Research Parallel Processor Prototype) [19]. Each processor is paired with a memory module, along with a memory map unit, a cache, and a network interface, to form, in RP3 terminology, a processor/memory element (PME). The memory module is partitioned into local and global address space. This partition is programmable and can be all global, all local, or any proportion of local and global; it can be selected independently on a PME by PME basis. PME virtual memory references are first translated by the memory map unit [95]. Memory references to locations already in cache are handled by the cache. Otherwise, the network interface passes local memory references directly to the memory module and routes global memory references through the network. In the RP3 architectural design, which has 512 PMEs, there are two buffered packet-switched multistage cube networks: one combining and one noncombining (see sections IV-A and IV-B). The noncombining network consists of four stages of 4×4 interchange boxes. In principle, this would provide 256 ports, but the number of ports is reduced to 128 in exchange for providing dual paths between the ports. Furthermore, each port is connected to four PMEs through a multiplexer and a demultiplexer. The network is actually composed of two unidirectional networks—one for requests and one for replies. A single unidirectional network handling both requests and replies can deadlock if requests start accumulating in the network and thereby not allowing replies to make progress. Two networks also improve performance.

In the architectural design, the combining network uses 2×2 interchange boxes and can combine the following operations: load, store, fetch-and-store, fetch-and-store-if-=-zero, fetch-and-add, fetch-and-min, fetch-and-max, fetch-and-OR, and fetch-and-AND. Only like operations can be combined, but loads are treated as fetch-and-adds of zero allowing loads and fetch-and-adds to combine with each other.

A prototype has been operational since October 1988. The network has 128 ports, as described in the preceding, but there are only 64 PMEs; each PME has its own port and half the ports are unused. The noncombining network, which is based on ECL components, has the buffers on the input ports of an interchange box rather than on the output ports as described in section VI-D. This has the advantage of being easier to build, but the disadvantage of increased congestion. The input buffers can hold 32 bytes, which is nominally eight words. Each memory request is for two or three words. The network can operate with a cycle time of 20 ns, but has been slowed to 70 ns to conform to the processor cycle time. An interchange box has a latency of two cycles, so a request takes eight cycles to traverse (in one direction) this four stage network. After the first byte of a message arrives (in eight cycles assuming no conflicts), a subsequent byte arrives each cycle. There is no combining network, but the fetch-and-OP operations (listed in the preceding) are still handled indivisibly at the memory modules.

An important feature of RP3 is that it can measure performance statistics without degrading machine performance [96]. For example, the machine can determine the rate and delay of memory requests.

F. NYU Ultracomputer

New York University is designing a processor-to-memory configuration machine parallel computer called the NYU Ultracomputer [24]. The most novel feature of this machine is that is supports the combining of concurrent memory operations, including concurrent fetch-and-adds. This is done using a buffered packet-switched combining multistage cube network (see section IV-B). The network mode of operation is cut-through routing (section IV-A). Each processor is connected to the network via a processor-network-interface, which supports caching, and each memory module is connected to the network via a memory-network interface.

A full prototype has yet to be built. Several bus-based prototypes based on the Motorola 68010 have been constructed. The largest has eight processors and 16 Mbytes of memory. Combining NMOS and noncombining NMOS and CMOS interchange boxes have been built. The CMOS (noncombining) interchange box has been used to construct a four-processor machine.

The project is now proposing the construction of a 256-processor machine based on the AMD 29000 processor family. Each memory module will have four Mbytes of memory for a total of one gigabyte in the entire machine. The network will use 2×2 interchange boxes. The network path width will be 32 bits; memory requests will use two or four network words. The cycle time of an interchange box is expected to be between 30 and 50 ns. The machine will support the following operations that combine in the network: load, load double word, store, store double word, fetch-and-store, fetch-and-store-if-=-zero, fetch-and-store-if-$\geq$-zero, fetch-and-add, fetch-and-OR, and an operation that is essentially equivalent to fetch-and-AND. It will also support reflection [97] (which allows a process to communicate with other processes without having to know on which processors the other processes are executing) and partial word store operations (e.g., store byte and store half-word operations), but these will not be combined in the network.

VIII. Single-Stage Variants of the Multistage Cube Network

Several problems occur as multistage cube networks get larger. First, the number of components in the network (network interchange boxes, network links) grows as $N \log_2 N$, where N is the number of PEs. For a computer with a very large number of PEs, the network cost can dominate the machine cost. Second, the "distance" (in this case meaning conflict-free communication time) between all arbitrary pairs of PEs is the same and is proportional to $\log_2 N$. If some pairs of PEs communicate more frequently than others, then these PEs cannot be placed "closer together" (in a network distance sense) to reduce the time required to transfer information. Converting the multistage cube network to a single-stage network addresses both of these problems. This section will describe how to do this conversion and will discuss properties of such single-stage networks.

A. Single-Stage Network Structure

To convert an N input/output Generalized Cube topology into a single-stage network, simply connect network output port i directly to network input port i, for $0 \leq i < N$. For this class of networks, each interchange box includes a connection to one (or more) PEs and will be referred to as a "routing node." Because each node is directly connected to other nodes, to move data from one PE to an adjacent one, only a single link must be traversed. Hence this is referred to as a "single-stage" network [6]. This is in contrast to a multistage network where, for any PE to communicate with any other, the traversal of multiple stages of links is required.

For this discussion, it will be assumed that each node in the single-stage network is associated with one or two PEs; actually, there may be a cluster of multiple PEs at each node and the processors of these PEs may or may not share their memory modules. The growth in the number of nodes and links is thus proportional to the number of PEs. A one-PE-per-node single-stage version of the multistage cube network having $m2^{m-1}$ routing nodes has $m2^{m-1}$ PEs. Sending data from one PE to another may require transferring the data through intermediate nodes associated with other PEs. Assuming one PE per node, the shortest distance is the distance between an adjacent pair of nodes or a "single stage." Thus, while in a multistage network there is a fixed distance between any pair of communicating PEs, in a single-stage network there is a range of distances between pairs of PEs. The transmitted information is packet switched between the sending PE and the receiving PE. Both unidirectional and bidirectional information flow approaches have been proposed for this type of network.

The hypercube network topology (mentioned in section III-A) is, in a sense, also a single-stage version of the multistage cube. One of the major differences between the hypercube and the networks described in this section is that the networks here use a fixed number of input/output ports (independent of N) for each node, while a hypercube requires $\log_2 N$. A detailed comparison of hypercube networks to the type presented in this section appears in [41].

B. Unidirectional with One PE Per Node

The MAN-YO network [43] is typical of the unidirectional-flow single-stage versions of the multistage cube network with one PE per node (the CBAR [39] and HCSN [40] networks are similar). The MAN-YO network is shown in Fig. 10. The network consists of m stages of 2^{m-1} routing nodes in the familiar multistage cube topology, except here the order of the stages is reversed from the Generalized Cube topology of section III-A (compare Figs. 1 and 10). A single PE is connected to each of the $m2^{m-1}$ routing nodes. Each network node has three input ports and three output ports. One input port and one output port go to the attached PE (these ports and the attached PE are not shown explicitly in Fig. 10). The other ports are for routing through the network. The information flow is unidirectional from the left side of the network to the right side of the network and then it loops around back to the left side of the network; i.e., information flows from stage i to stage $i + 1 \bmod m$.

An m-stage system can be viewed as consisting of 2^m loops that are interconnected at the routing nodes [42]. In Fig. 11, these loops are generated by using only the straight-through routing in the routing nodes. Each loop is given a label $\ell_{m-1} \cdots \ell_1 \ell_0$. Two loops whose labels differ only in bit position i intersect at a node in stage i, $0 \leq i < m$. A PE's location in the network can be specified by the stage number and

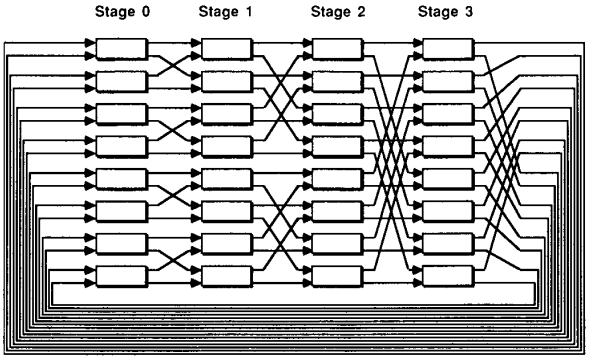


Fig. 10. A four-stage MAN-YO network. Each interchange box is routing node with attached PE.

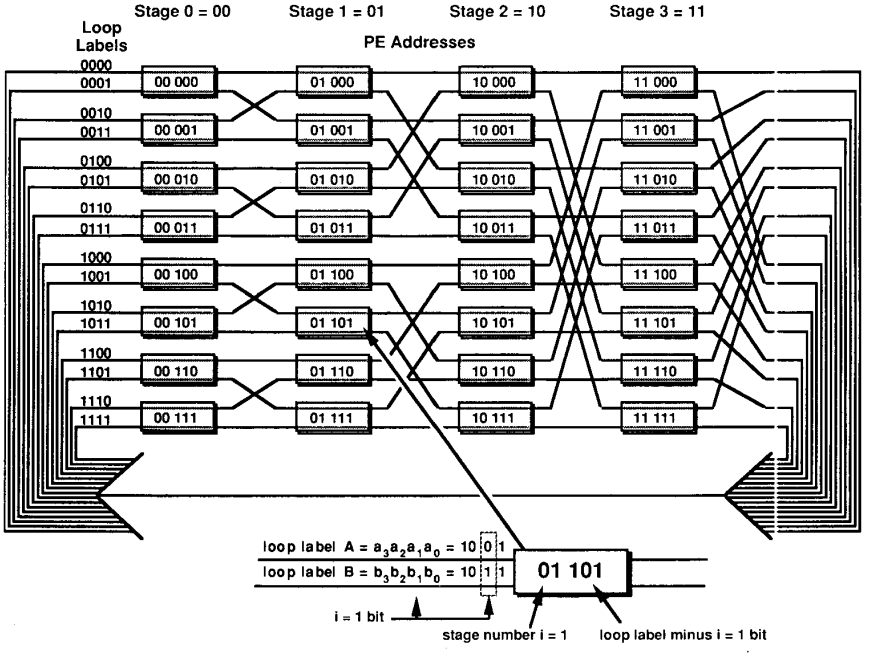


Fig. 11. Loop labels (shown at left) and processor addresses (shown in interchange boxes) in four-stage MAN-YO network.

loop labels of the two loops running through the routing node to which the PE is attached. Because the two loops intersecting at a node in stage i differ only in the ith bit position, the PE address of a PE in stage i is constructed by concatenating its stage number and the label of one of the two loops minus its ith bit position $\{\ell_{m-1}\cdots\ell_{i+1}\ell_{i-1}\cdots\ell_1\ell_0\}$. This is written as (i) $\{\ell_{m-1}\cdots\ell_{i+1}\ell_{i-1}\cdots\ell_1\ell_0\}$. These PE addresses are also shown in Fig. 11. Another way to describe the MAN-YO topology and PE addressing is that PE (i) $\{\ell_{m-1}\cdots\ell_{i+1}\ell_{i-1}\cdots\ell_1\ell_0\}$, for $i < m-1$, is connected to PEs $(i+1)$ $\{\ell_{m-1}\cdots\ell_{i+2}0\ell_{i-1}\cdots\ell_1\ell_0\}$ and $(i+1)$ $\{\ell_{m-1}\cdots\ell_{i+2}1\ell_{i-1}\cdots\ell_1\ell_0\}$, and for $i = m-1$ is connected to PEs (0) $\{0\ell_{m-2}\cdots\ell_1\}$ and (0) $\{1\ell_{m-2}\cdots\ell_1\}$.

Routing in this network is performed in two phases. A transmitted packet first moves between loops until it is on one of the two loops going to the destination PE. Then it follows that loop to the stage of the destination PE.

For the first phase, the packet's destination address is broken into two parts, the loop number and the stage number. If the destination PE is in stage i, then the loop number is expanded to a full-loop label by inserting a "don't care" into the ith bit position of the loop number. When a packet enters a node in stage j, the packet is routed out of the upper node output if bit j in the destination loop address is 0 and out the lower node output if bit j in the destination address is 1. This is analogous to the destination tag routing scheme described in section III-C. Once the packet is on one of the two loops running through the destination PE's routing node (i.e., the first phase is complete), the packet is routed along that loop until it enters a stage whose stage number is equal to that of its destination PE.

Because there are m bit positions in the loop labels, the packet may have to go through $m-1$ routing nodes to get on one of the two loops entering the destination node.

Counting the packet movement between adjacent routing nodes as a step, this can require $m - 1$ steps. Once on the desired loop, then the packet is at most $m - 1$ stages (or steps) from its destination. Therefore the maximum distance in the network with $m2^{m-1}$ PEs is $2m - 2$ steps. However, in contrast to the multistage cube network, there are a range of distances between pairs of PEs. Fig. 12 shows a

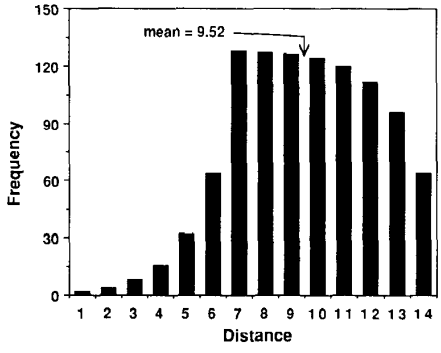


Fig. 12. Interprocessor distance distribution for 1024-node MAN-YO network.

histogram for the distances between one PE and all the other PEs in an eight-stage (1024-PE) MAN-YO system.

C. Unidirectional with Two PEs Per Node

A variant of the unidirectional flow network, the loop structured switching network (LSSN), is proposed in [42]. There two PEs are attached to each routing node (the multiloop packet network in [98] is similar). Each PE is attached to only one of the loops running through a routing node. The ith bit in the loop label is now relevant for a PE attached to a routing node in stage i; the address of a PE is the concatenation of its stage number with the complete loop label of the loop to which it is attached. Routing is performed similarly to the one-PE-per-node networks except the loop label address has a single bit value in bit position i instead of a "don't care."

The performance difference between the single-stage LSSN and the multistage cube network family (specifically, the baseline network) was analyzed in [42]. The simulations used request interval as the varying parameter, where request interval was defined as the mean time between the last successful packet transmission from a PE and the next packet generation in that PE. Network performance was characterized in terms of throughput (the mean number of packets moving through the network per unit time) and mean packet delay (the mean time between generation of a packet at the sending PE and reception of the packet at the destination PE). Both 16- and 64-PE multistage cube networks and a 64-PE single-stage LSSN (that has the same number of interchange boxes as the 16-PE multistage cube) were modeled. It was found that when the request interval was very short the LSSN had about the same throughput as the 16-PE multistage cube (which is about one-fourth the throughput of the 64-PE multistage cube), but about three times the mean delay of a 16-PE multistage cube. However, if the request interval was moderate or long, the throughput and delay of the LSSN approached that of the 64-PE multistage cube network. Because the 64-PE LSSN has one-sixth the number of components (routing nodes and links) of the 64-PE multistage cube network, a substantial network hardware savings is possible without performance degradation for all but very short request intervals.

D. Bidirectional with One PE per Node

The "Lambda network" [41] is a bidirectional single-stage network based on the multistage cube network topology. The Lambda network is very similar to the MAN-YO network except that bidirectional information travel is permitted over each link. PE positions, stage numbers, and loop labels are identical to those of the MAN-YO network. Each network routing node has five input and five output ports: one input-output port pair for the single attached PE and one input-output port pair for each of the four links incident to the node. The effect of bidirectional information travel is twofold: 1) it reduces the mean and maximum distances between PEs through the network and 2) it adds network fault-tolerance while still using a local routing scheme. The reduction in mean and maximum distances is shown in Fig. 13 as a histogram of distances between one PE and

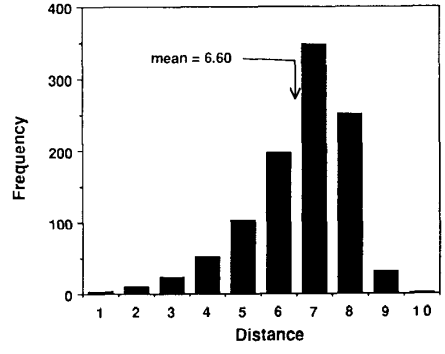


Fig. 13. Interprocessor distance distribution for 1024-node Lambda network.

all the other PEs in a 1024-node Lambda network. The network fault-tolerance is discussed in [41].

There are two drawbacks to the bidirectional information transfer in the Lambda network. First, the routing nodes become more complicated because there are more input and output ports for the information to be switched between inside the node. Second, routing through the network becomes more complicated. Routing is no longer "move until you are on the right loop and then follow that loop to the destination stage"; routing now becomes the choice of moves between loops and stages that achieves the minimum distance path between an arbitrary communicating pair of PEs.

A routing ring model was devised to understand routing in the Lambda network [41]. The routing ring for an m-stage Lambda system is shown in Fig. 14. Define the "loop resultant" to be the exclusive-or of the loop label portion of the packet's destination routing tag and the label of the loop on which the packet is currently traveling. The ring has m circumferential positions, one for each stage, and a loop resultant bit appears at each position. Initially it is the exclusive-or of the destination and source loop labels, analogous

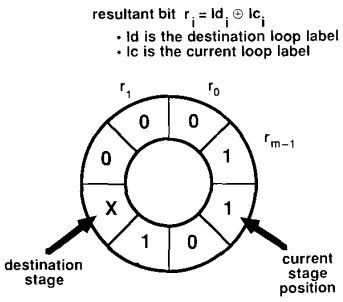


Fig. 14. Routing ring model for Lambda network.

to the XOR routing tag scheme of section III-C. Because the loop label portion of the destination routing tag contains a "don't care" (as in the MAN-YO network), the loop resultant has a "don't care" in its bit position corresponding to the packet's destination stage number (see Fig. 14). There are two pointers to positions on the ring: one pointer to a packet's current stage and another pointer to the packet's destination stage. Each step of a packet's travel moves the current stage pointer one position clockwise or counterclockwise on the ring. The loop resultant bit at a packet's current stage position is changed when the packet switches between the loops in a node at that stage (this is because the label of the loop the packet is traveling on has changed). When the loop resultant is all zeros with a "don't care" at the destination stage bit position, this means that the packet is on one of the two loops going through the destination PE routing node.

Every bit position in the initial loop resultant that contains a one indicates a stage that the packet must pass through on its way to its destination. The shortest length path between a sending PE and a receiving PE is a series of steps around the routing ring beginning at the stage position of the sending PE, ending at the stage position of the destination PE, and touching every nonzero position in the initial loop resultant. There may be several minimum length paths between a communicating PE pair. Each of these paths will require zero, one, or two changes in direction of the packet motion around the routing ring, corresponding to changes of packet direction through the network.

There are two routing decisions for each packet as it moves through a node. The first is whether to change loops, and this is determined by the bit position in the loop resultant corresponding to the current stage number (if it is a "1," the packet should change loops). The second decision is the direction (forward or backward) of packet exit. The optimal choice of direction is a complex function of loop resultant, current stage, and destination stage. The Lambda network uses a routing table in each node to determine exit direction. This routing table must have $2N$ entries for a network with N PEs (there are $2N$ possible values resulting from the concatenation of all stage numbers with all loop labels [41]). However, the exit direction can be written in a sum-of-products form (as a function of the loop resultant and the difference of the current stage number and the destination number) and this sum-of-products form can be minimized. For example, the sum-of-products form for a 1024-PE Lambda network has eleven products, the largest of which has only four binary inputs.

The Lambda network can be shown to be similar to the lens network [99]. They differ in that the lens network uses busses to connect sets of PEs (typically three PEs per bus). There are no point-to-point connections as in the Lambda.

E. Deadlock Considerations

Single-stage versions of the multistage cube network share a common problem: the possibility of deadlock. Deadlock is a condition where packets in transit get stuck indefinitely in intermediate nodes [100]. This occurs because there can be a cyclical demand for network resources, e.g., a packet in node A wants to go to node B but cannot because node B is full of packets, a packet in node B wants to go to node C, which is also full, and a packet in node C want to go to node A, but unfortunately A is also full. Schemes to avoid deadlock are either specific for particular networks (e.g., in [42]) or are designed for a general types of networks [101]–[103]. The schemes fall into two classes: 1) ensuring that a possible deadlocked state cannot be entered [103], and 2) detecting and resolving a found deadlocked state, usually by permanently discarding a packet and forcing it to be resent [102] or by temporarily discarding a packet by placing it in a special buffer and later reentering it in the network [101].

F. Partitionability

Observe that single-stage variants share with the multistage cube network the ability to be partitioned that was discussed in section III-D. By restricting the exchange movement of a packet between loops in all nodes in a given stage of these networks, the networks are effectively partitioned in half just like the multistage cube network. By preventing the exchange movement in r stages, the systems are broken into 2^r identically sized subsystems. If PEs in nodes forced to the straight state are not used, each subsystem is a Lambda network. Variations that make use of the PEs in nodes forced to straight are also possible.

G. Theoretical Results

The unidirectional single-stage networks with one PE per node discussed in this section are often referred to in the theory community as "butterfly" networks. These networks are considered more realistic than the "shared memory parallel computer" model, where each processor can access any memory location in constant time (this theoretical model assumes there are no delays resulting from network or memory conflicts). An N processor shared memory parallel computer is at least as fast as an N processor butterfly network. The question is how much faster is it? One way to determine this is to show how many steps it takes for a butterfly network to simulate one step of a shared memory network. There are two problems that must be overcome in such simulations. The first is to determine how fast a butterfly network can route messages between processors. The second comes from the fact that a butterfly network has N memory modules (one associated with each processor), and many processors may want to access the same memory module, thereby causing a bottleneck. It has been shown that a butterfly network can simulate one step of a shared-memory parallel computer deterministically in time $\Theta(\log_2^2 N)$ and probabilistically (i.e., on average) in time $\Theta(\log_2 N)$ [104]–[107].

Theoreticians have also studied networks like the Lambda, but with two PEs per node. The cube-connected cycles network [108] is essentially of this type [109]. It was studied for its ability to perform "ascend-descend" algorithms (such as merging, sorting, and routing) with optimal implementation layout area.

H. Summary

Briefly summarizing, the single-stage variants of the multistage cube network address some of the problems of the multistage cube network but introduce problems of their own. The variant's one PE per network node structure provides a range in network distances between PEs in contrast to the fixed distance between PEs in the multistage cube network. It has been shown that the single-stage network performs the same as the multistage cube network in terms of network throughput and mean packet transfer time when the network communication load is medium or low, but the multistage network outperforms the single-stage networks for high network loading conditions. The single-stage networks are subject to deadlock, and provisions for deadlock avoidance in these networks are necessary if the networks are to be implemented. However, both unidirectional and bidirectional variants have substantially less network hardware requirements per PE than the multistage cube. Under what conditions one should use the multistage cube itself or one of its single-stage variants (and which variant, i.e., the unidirectional or bidirectional) is an open problem.

IX. Conclusion

Characteristics of the multistage cube topology were presented. A variety of different attributes of this interconnection network family were explored. The topics examined spanned theoretical issues, practical design and implementation concepts and tradeoffs, and examples of networks that have been constructed. Many open problems in this important area of research were pointed out to indicate some of the topics where further study is needed.

It is natural to ask why this paper concentrates on the multistage cube network. There could be many other network topologies that are just as good, if not better, for constructing parallel machines. This paper overviews the multistage cube network and then lists many of its desirable features. One can reverse the situation by describing desirable features that an interconnection network should have, and then listing which networks have these properties. The next few paragraphs examine several important properties: routing control, performance versus cost, and layout. In each case, the multistage cube network is among the networks that are optimal with respect to the property.

Typically, in a processor-to-memory configuration (section II-E) the address of a memory location can be divided so that $n = \log_2 N$ bits denote the desired memory and the remaining bits denote a memory location within the memory. It is then very convenient to use the destination tag scheme (described in section III-C): a packet needs only to know its destination, and the same routing scheme can be used for all packets (independent of the issuing processor). With a processor-to-memory system, it is also desirable that the destination tag scheme can be used when routing in the reverse direction; i.e., the same destination tag can be used to route from any memory to the same processor. It can be shown that any size N interconnection network (composed of two-input two-output interchange boxes) that has this routing property in both directions is isomorphic to the multistage cube network topology [28]. Thus routing considerations argue for the multistage cube network.

One way of measuring the cost of an interconnection network is to count the number of wires (which is essentially the same as counting the number of interchange boxes). The performance of a network can be measured by its bandwidth or by its delay (see section VI). It can be shown that the multistage cube network is among the networks that have optimal bandwidth/cost ratio. When network traffic is $M/M/1$ (Poisson arrivals, exponential service times), as is often assumed in queueing theory, the multistage cube network is also among the networks that have optimal delay for their cost [110], [111].

Another measure of interconnection network cost is (implementation) layout area. A multistage cube network uses $\Theta(N^2)$ area. For example, consider the $N = 16$ network shown in Fig. 4. The number of wire crossings in the first (input) stage is proportional to N^2, and each wire crossing uses constant area. Each half of the second stage has approximately one-fourth as many wire crossings as the first stage. Because there are two halves, the total number of wire crossings is approximately half as many as in the first stage. The third stage has approximately one-fourth as many wire crossings as the second stage. In general (for large N), the ith stage has approximately $1/2^{i-1}$ times as many wire crossings as the input stage. Thus, summing over all the stages, the total number of wire crossings, and hence the area, is $\Theta(N^2)$. Any network capable of supporting bandwidth linear in the number of processors must have quadratic area [112], so the multistage cube network is among the optimal networks for network area.

Wires have constant thickness, so the area of a wire is proportional to its length. The average length of a wire in the layout of Fig. 4 is $\Theta(N/\log N)$, because the area is $\Theta(N^2)$ and there are $\Theta(N \log N)$ wires. However, the longest wires (which are from the first stage) have length $\Theta(N)$. This could slow the network down if the network cycle time is governed by the longest wires. Also, long wires have less desirable electrical properties. It has recently been shown that the multistage cube network can be laid out in area $\Theta(N^2)$ without any long wires (i.e., all wires having length at most $\Theta(N/\log N)$) [113]. Thus multistage cube networks are among the optimal area networks that also have optimal maximum wire length.

The goal of this paper was to explore the different ways in which the multistage cube topology can be used in supercomputer systems employing large-scale parallel processing. The attempt was made to present this material at a level appropriate for a reader who has a technical background, but not necessarily in the field of computing. Due to the importance of this network, there is a vast amount of research related to it being conducted, and hence a great deal of information in the open literature. Thus this paper was not intended to be an exhaustive description of all that is known about this network family, and the list of over one hundred references provided is representative and not exhaustive. The paper described what the authors consider to be some of the many important features and capabilities of this network topology. The reader interested in more depth and/or breadth can use the knowledge gained from

this paper as the background needed to do further reading, employing the list of references as a guide to the relevant literature.

Acknowledgment

The authors gratefully acknowledge the following people for providing information for section VII: Ken Batcher, William Brantley, John Goodhue, Allan Gottlieb, Richard Kenner, Wally Kleinfelder, Duncan Lawrie, Kevin McAuliffe, Pierre Pero, Alex Viedenbaum, and Pam Waterman.

References

[1] K. J. Thurber, "Parallel processor architectures—part 1: General purpose systems," *Comput. Design*, vol. 18, pp. 89–97, Jan. 1979.

[2] G. Broomell and J. R. Heath, "Classification categories and historical development of circuit switching topologies," *ACM Comput. Surveys*, vol. 15, no. 2, pp. 95–133, June 1983.

[3] T. Y. Feng, "A survey of interconnection networks," *Comput.*, vol. 14, no. 12, pp. 12–27, Dec. 1981.

[4] G. J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons*. New York: John Wiley, 1987.

[5] G. M. Masson, G. C. Gingher, and S. Nakamura, "A sampler of circuit switching networks," *Comput.*, vol. 12, no. 2, pp. 32–48, June 1979.

[6] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, Second Ed.*, New York: McGraw-Hill, to be published 1990.

[7] H. J. Siegel, R. J. McMillen, and P. T. Mueller, Jr., "A survey of interconnection methods for reconfigurable parallel processing systems," *AFIPS 1979 Nat. Comput. Conf.*, June 1979, pp. 529–542.

[8] K. J. Thurber and G. M. Masson, *Distributed-Processor Communication Architecture*. Lexington, MA: Lexington Books, D. C. Heath and Co., 1979.

[9] C.-L. Wu and T. Y. Feng, *Tutorial: Interconnection Networks for Parallel and Distributed Processing*, IEEE Computer Society Press, Silver Spring, MD, 1984.

[10] W. C. McDonald and J. M. Williams, "The advanced data processing test bed," *IEEE Comput. Soc. Second Int. Comput. Software Appl. Conf.*, Mar. 1978, pp. 346–351.

[11] H. J. Siegel and R. J. McMillen, "The multistage cube: a versatile interconnection network," *Comput.*, vol. 14, no. 12, pp. 65–76, Dec. 1981.

[12] BBN Advanced Computers, Inc., *Inside the Butterfly Plus*, Cambridge, MA: BBN Advanced Computers, Inc., 1987.

[13] BBN Advanced Computers, Inc., *Inside the GP1000*, Cambridge, MA: BBN Advanced Computers, Inc., 1988.

[14] BBN Advanced Computers, Inc., *Inside the TC2000*, Cambridge, MA: BBN Advanced Computers, Inc., 1989.

[15] G. H. Barnes and S. F. Lundstrom, "Design and validation of a connection network for many-processor multiprocessor systems," *Comput.*, vol. 14, no. 12, pp. 31–41, Dec. 1981.

[16] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and A. H. Sameh, "Parallel supercomputing today and the Cedar approach," *Science*, vol. 231, pp. 967–974, Feb. 1986.

[17] J. B. Dennis, G. A. Boughton, and C. K. C. Leung, "Building blocks for data flow prototypes," *Seventh Ann. Symp. Comput. Architec.*, May 1980, pp. 1–8.

[18] A. E. Filip, "A distributed signal processing architecture," *Third Int. Conf. Distributed Comput. Syst.*, Oct. 1982, pp. 49–55.

[19] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and architecture," *1985 Int. Conf. Parallel Processing*, Aug. 1985, pp. 764–771.

[20] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Computers*, vol. C-30, no. 12, pp. 934–947, Dec. 1981.

[21] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An overview of the PASM parallel processing system," in *Comput. Architec.*, D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, Eds. Washington, DC: IEEE Computer Society Press, 1987, pp. 387–407.

[22] F. A. Briggs, K.-S. Fu, K. Hwang, and B. W. Wah, "PUMPS architecture for pattern analysis and image database management," *IEEE Trans. Comput.*, vol. C-31, no. 10, pp. 969–982, Oct. 1982.

[23] K. E. Batcher, "STARAN series E," *1977 Int. Conf. Parallel Processing*, Aug. 1977, pp. 140–143.

[24] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer—Designing an MIMD shared-memory parallel computer," *IEEE Trans. Computers*, vol. C-32, no. 2, pp. 175–189, Feb. 1983.

[25] H. J. Siegel, "Interconnection networks for SIMD machines," *Computer*, vol. 12, no. 6, pp. 57–65, June 1979.

[26] H. J. Siegel and S. D. Smith, "Study of multistage SIMD interconnection networks," *Fifth Ann. Symp. Computer Architec.*, April 1978, pp. 223–229.

[27] 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.

[28] C. P. Kruskal and M. Snir, "A unified theory of interconnection network structure," *Theoret. Comput. Sci.*, vol. 48, no. 1, pp. 75–94, 1986.

[29] W. Crowther, J. Goodhue, R. Thomas, W. Milliken, and T. Blackadar, "Performance measurements on a 128-node butterfly parallel processor," *1985 Int. Conf. Parallel Processing*, Aug. 1985, pp. 531–540.

[30] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Computers*, vol. C-30, no. 10, pp. 771–780, Oct. 1981.

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

[32] M. C. Pease III, "The indirect binary *n*-cube microprocessor array," *IEEE Trans. Computers* vol. C-26, no. 5, pp. 458–473, May 1977.

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

[34] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Computers*, vol. C-24, no. 12, pp. 1145–1155, Dec. 1975.

[35] G. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *First Ann. Symp. Computer Architec.*, pp. 21–28, Dec. 1973.

[36] M. Jeng and H. J. Siegel, "Design and analysis of dynamic redundancy networks," *IEEE Trans. Computers*, vol. C-37, no. 9, pp. 1019–1029, Sept. 1988.

[37] G. B. Adams III and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems," *IEEE Trans. Computers*, vol. C-31, no. 5, pp. 443–454, May 1982.

[38] G. B. Adams III, D. P. Agrawal, and H. J. Siegel, "A survey and comparison of fault-tolerant multistage interconnection networks," *Computer*, vol. 20, no. 6, pp. 14–27, June 1987.

[39] V. Balasubramanian and P. Banerjee, "A fault tolerant massively parallel processing architecture," *J. Parallel Distributed Comput.*, vol. 4, no. 4, pp. 363–383, Aug. 1987.

[40] S. K. Tripathi and S.-T. Huang "Distributed resource scheduling for a large scale network of processors: HCSN," *Sixth Int. Conf. Distributed Comput. Syst.*, pp. 321–328, May 1986.

[41] L. M. Napolitano, Jr., "The design of a high performance packet-switched network," *J. Parallel Distributed Comput.*, to appear, 1990 (preliminary version available as Sandia National Laboratories Technical Report SAND87-8687).

[42] F. S. Wong and M. R. Ito, "A loop-structured switching network," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 450–455, May 1984.

[43] K. Koike and K. Ohmori, "MAN-YO: A special purpose parallel machine for logic design automation," *1985 Int. Conf. Parallel Processing*, pp. 583–590, Aug. 1985.

[44] J.-L. Baer, *Computer Systems Architecture*, Potomac, MD: Computer Science Press, 1980.

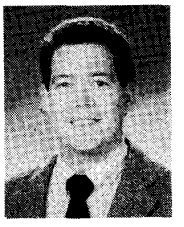
[45] R. W. Hockney and C. R. Jeshope, *Parallel Computers*, Bristol, England: Adam Hilger Ltd., 1981.

[46] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, New York: McGraw-Hill, 1984.
[47] V. M. Milutinovic (editor), *Computer Architecture: Concepts and Systems*, New York: North-Holland, 1988.
[48] H. S. Stone, "Parallel computers," in *Introduction to Computer Architecture* (Second Ed.), H. S. Stone, Ed. Chicago, IL: Science Research Associates, Inc., 1980, pp. 363-425.
[49] H. S. Stone, *High Performance Computer Architecture*, Reading, MA: Addison-Wesley, 1987.
[50] K. J. Thurber, *Large Scale Computer Architecture: Parallel and Associative Processors*, Rochelle Park, NJ: Hayden Book Co., 1976.
[51] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, vol. 54, pp. 1901-1909, Dec. 1966.
[52] K. E. Batcher, "STARAN parallel processor system hardware," *AFIPS 1974 Natl. Computer Conf.*, May 1974, pp. 405-410.
[53] K. E. Batcher, "Design of massively parallel processor," *IEEE Trans. Comput.*, vol. C-29, no. 9, pp. 836-844, Sept. 1980.
[54] K. E. Batcher, "Bit serial parallel processing systems," *IEEE Trans. Comput.*, vol. C-31, no. 5, pp. 377-384, May 1982.
[55] W. D. Hillis, *The Connection Machine*, Cambridge, MA: MIT Press, 1985.
[56] L. W. Tucker and G. G. Robertson, "Architectures and applications of the Connection Machine," *Computer*, vol. 21, no. 8, pp. 26-38, Aug. 1988.
[57] G. J. Nutt, "Microprocessor implementation of a parallel processor," *Fourth Ann. Symp. Computer Architecture*, Mar. 1977, pp. 147-152.
[58] G. J. Nutt, "A parallel processor operating system comparison," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 467-475, Nov. 1977.
[59] A. K. Jones, R. J. Chansler, Jr., I. Durham, P. Feiler, and K. Schwans, "Software management of CM*—A distributed multiprocessor," *AFIPS 1977 Nat. Computer Conf.*, June 1977, pp. 657-663.
[60] R. J. Swan, A. Bechtolsheim, K. W. Lai, and J. K. Ousterhout, "The implementation of the Cm* multimicroprocessor," *AFIPS 1977 Nat. Computer Conf.*, June 1977, pp. 645-655.
[61] R. J. Swan, S. Fuller, and D. P. Siewiorek, "Cm*: A modular multimicroprocessor," *AFIPS 1977 Nat. Computer Conf.*, June 1977, pp. 637-644.
[62] Intel Corporation, *A New Direction in Scientific Computing*, Order # 28009-001, Intel Corporation, Beaverton, OR, 1985.
[63] J. P. Hayes, T. N. Mudge, Q. F. Stout, and S. Colley, "Architecture of a hypercube supercomputer," *1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 653-660.
[64] B. J. Smith, "Architecture and applications of the HEP multiprocessor computer system," *Comput. Architect.*, D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, Eds. Washington, DC: IEEE Computer Society Press, 1987, pp. 486-493.
[65] R. J. McMillen and H. J. Siegel, "Routing schemes for an augmented data manipulator in an MIMD system," *IEEE Trans. Computers*, vol. C-31, no. 12, pp. 1202-1214, Dec. 1982.
[66] H. J. Siegel, "The theory underlying the partitioning of permutation networks," *IEEE Trans. Comput.* vol. C-29, no. 9, pp. 791-801, Sept. 1980.
[67] K. Y. Wen, "Interprocessor Connections—Capabilities, Exploitation, and Effectiveness," Ph.D. thesis, Report UIUCDCS-R-76-830, Computer Science Dept., University of Illinois, 1976.
[68] S. D. Smith and H. J. Siegel, "Recirculating, pipelined, and multistage SIMD interconnection networks," *1978 Int. Conf. Parallel Processing*, August 1978, pp. 206-214.
[69] M. Lee and C.-L. Wu, "Performance analysis of circuit switching baseline interconnection networks," *Eleventh Ann. Int. Symp. Computer Architect.*, June 1984, pp. 82-90.
[70] W. J. Dally and C. L. Sietz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Computers*, vol. C-36, no. 5, pp. 547-553, May 1987.
[71] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer commmunication switching technique," *Comput. Networks*, vol. 3, 1979, pp. 267.
[72] P. Kermani and L. Kleinrock, "A tradeoff study of switching systems in computer communications networks," *IEEE Trans. Comput.*, vol. C-29, no. 12, pp. 1052-1060, Dec. 1980.
[73] S. Cheemalavagu and M. Malek, "Analysis and simulation of banyan interconnection networks with 2×2, 4×4, and 8×8 switching elements," *1982 Real-Time Sys. Symp.*, Dec. 1982, pp. 83-89.
[74] M. A. Franklin and S. Dhar, "Interconnection networks: Physical design and performance analysis," *J. Parallel Distributed Comput.*, vol. 3, no. 3, pp. 352-372, Sept. 1986.
[75] R. J. McMillen, G. B. Adams III, and H. J. Siegel, "Performance and implementation of 4×4 switching nodes in an interconnection network for PASM," *1981 Int. Conf. Parallel Processing*, Aug. 1981, pp. 229-233.
[76] N. J. Davis and H. J. Siegel, "Performance analysis of multiple-packet multistage cube networks and comparison to circuit switching," *1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 108-114.
[77] A. R. Tripathi and G. J. Lipovski, "Packet switching in banyan networks," *Sixth Ann. Symp. Computer Architec.*, April 1979, pp. 160-167.
[78] A. Huang and S. Knauer, "Starlite: A wideband digital switch," *IEEE Global Telecommun. Conf.*, Nov. 1984, pp. 121-125.
[79] G. F. Pfister and V. A. Norton, "'Hot spot' contention and combining in multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 933-938, Oct. 1985.
[80] M. Kumar and G. Pfister, "The onset of hot spot contention," *1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 28-34.
[81] G. Lee, C. P. Kruskal, and D. J. Kuck, "The effectiveness of combining in multistage interconnection networks in the presence of 'hot spots'," *1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 35-41.
[82] R. H. Thomas, "Behavior of the Butterfly parallel processor in the presence of memory hot spots," *1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 46-50.
[83] C. P. Kruskal, L. Rudolph, and M. Snir, "Efficient synchronization on multiprocessors with shared memory," *ACM Trans. Programming Languages Syst.*, vol. 10, no. 4, pp. 579-601, Oct. 1988.
[84] M. A. Abidi and D. P. Agrawal, "On conflict-free permutations in multi-stage interconnection networks," *J. Digital Syst.*, vol. 4, no. 2, Summer 1980, pp. 115-134.
[85] J. Lenfant, "Parallel permutations of data: A Benes network control algorithm for frequently used permutations," *IEEE Trans. Computers*, vol. C-27, no. 7, pp. 637-647, July 1978.
[86] H. J. Siegel, "Partitionable SIMD computer system interconnection network universality," *Sixteenth Allerton Conf. Commun. Contr. Comput.*, Oct. 1978, pp 586-595.
[87] D. S. Parker, "Notes on shuffle-exchange type switching networks," *IEEE Trans. Comput.*, vol. C-29, no. 3, pp. 213-222, Mar. 1980.
[88] C. P. 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.
[89] M. Kumar and J. R. Jump, "Performance of unbuffered shuffle-exchange networks," *IEEE Trans. Comput.*, vol. C-35, no. 6, pp. 573-578, June 1986.
[90] R. R. Koch, "Increasing the size of a network by a constant factor can increase performance by more than a constant factor," *29th Ann. Symp. Foundations Comput. Sci.*, Oct. 1988, pp. 221-230.
[91] C. P. Kruskal, M. Snir, and A. Weiss, "The distribution of waiting times in clocked multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-37, no. 11, pp. 1337-1352, Nov. 1988.
[92] D. M. Dias and J. R. Jump, "Packet switching interconnection networks for modular systems," *Computer*, vol. 14, no. 12, pp. 43-54, Dec. 1981.
[93] K. E. Batcher, "The multidimensional access memory in STARAN," *IEEE Trans. Comput.*, vol. C-26, no. 2, pp. 174-177, Feb. 1977.
[94] K. Padmanabhan and D. H. Lawrie, "A class of redundant path multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-32, no. 12, pp. 1099-1108, Dec. 1983.
[95] W. C. Brantley, K. P. McAuliffe, and J. Weiss, "The RP3 processor-memory element," *1985 Int. Conf. Parallel Processing*, Aug. 1985, pp. 782-789.
[96] W. C. Brantley, K. P. McAuliffe, and T. A. Ngo, "RP3 Performance Monitoring Hardware," in *Instrumentation for*

Parallel Computer Systems, M. Simmons, R. Koskela, and I. Bucher, Eds. Reading, MA: Addison-Wesley, 1989, pp. 35–47.
[97] J. Edler, A. Gottlieb, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, M. Snir, P. Teller, and J. Wilson, "Issues related to MIMD, shared-memory computer architectures—the NYU Ultracomputer approach," *12th Ann. Int. Symp. Computer Architect.*, June 1985, pp. 126–135.
[98] L. M. Napolitano Jr., "A computer architecture for dynamic finite element analysis," *13th Int. Symp. Computer Architecture*, June 1986, pp. 316–323.
[99] R. A. Finkel and M. H. Solomon, "The Lens interconnection strategy," *IEEE Trans. Comput.*, vol. C-30, no. 12, pp. 960–965, Dec. 1981.
[100] K. D. Günther, "Prevention of deadlocks in packet-switched data tansport systems," *IEEE Trans. Commun.*, vol. COM-29, no. 4, pp. 512–524, Apr. 1981.
[101] C.-W. Chan and T.-S. P. Yum, "An algorithm for detecting and resolving store-and-forward deadlocks in packet-switched networks," *IEEE Trans. Commun.*, vol. COM-35, no. 8, pp. 801–807, Aug. 1987.
[102] D. Gelernter, "A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks," *IEEE Trans. Comput.*, vol. C-30, no. 10, pp. 709–715, Oct. 1981.
[103] I. S. Gopal, "Prevention of store-and-forward deadlock in computer networks," *IEEE Trans. Commun.*, vol. COM-33, no. 12, pp. 1258–1264, Dec. 1985.
[104] H. Alt, T. Hagerup, K. Mehlhorn, and F. P. Preparata, "Deterministic simulation of idealized parallel computers on more realistic ones," *SIAM J. Comput.*, vol. 16, pp. 808–835, 1987.
[105] A. K. Karlin and E. Upfal, "Parallel hashing—an efficient implementation of shared memory," *18th ACM Symp. Theory Computing*, 1986, pp. 160–168.
[106] A. G. Ranade, "How to emulate shared memory," *28th Ann. Symp. Foundations of Computer Sci.*, pp. 185–194, Oct. 1987.
[107] E. Upfal and A. Wigderson, "How to share memory in a distributed system," *25th Ann. Symp. Foundations Computer Sci.*, Oct. 1984, pp. 171–180.
[108] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Commun. ACM*, vol. 24, no. 5, pp. 300–309, May 1981.
[109] B. N. Jain and S. K. Tripathi, "An analysis of cube-connected cycles and circular shuffle networks for parallel computation," *J. Parallel Distributed Computing*, vol. 5, no. 6, pp. 741–754, Dec. 1988.
[110] C. P. Kruskal and M. Snir, "Optimal interconnection networks for parallel processors: The importance of being square," in *Current Advances in Distributed Computing and Communications*, Y. Yemini, Ed. Computer Science Press, Inc., 1987, pp. 91–113.
[111] C. P. Kruskal and M. Snir, "Cost-bandwidth tradeoffs for communication networks," *ACM Symp. Parallel Algorithms Architectures*, June 1989, pp. 32–41.
[112] C. P. Kruskal and M. Snir, "The importance of being square," *Eleventh Ann. Int. Symp. Computer Architecture*, June 1984, pp. 91–98.
[113] R. Beigel and C. P. Kruskal, "Processor networks and interconnection networks without long wires," *ACM Symp. Parallel Algorithms Architectures*, June 1989, pp. 42–51.

Howard Jay Siegel (Senior Member, IEEE) received two B.S. degrees from the Massachusetts Institute of Technology (MIT), and the M.S.E. and Ph.D. degrees from Princeton University.

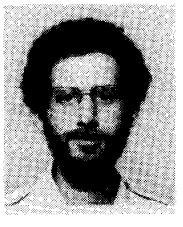
In 1976 he joined the School of Electrical Engineering at Purdue University, where he is a Professor and Coordinator of the Parallel Processing Laboratory. His current research focuses on interconnection networks and the use and design of the unique PASM reconfigurable parallel computer system (a prototype of which is supporting active experimentation). He does consulting, has prepared two videotape courses, and has given tutorials for various organizations, all on parallel computing. Dr. Siegel has coauthored over 120 technical papers, has coedited four volumes, and authored the book *Interconnection Networks for Large-Scale Parallel Processing*. He was General Chairman of the Third International Conference on Distributed Computing Systems (1982), Program Co-Chairman of the 1983 International Conference on Parallel Processing, and General Chairman of the Fifteenth Annual International Symposium on Computer Architecture (1988). He was a guest editor of two special issues of the IEEE TRANSACTIONS ON COMPUTERS and is currently Co-Editor-in-Chief of the *Journal of Parallel and Distributed Computing*. He is a member of the Eta Kappa Nu and the Sigma Xi honorary societies.



Wayne G. Nation received the B.S.E.E. degree from the University of Kentucky, Lexington, KY, in 1985, and the M.S.E.E. degree from Purdue University, West Lafayette, IN, in 1986. He is currently pursuing the Ph.D. degree from the School of Electrical Engineering at Purdue University.

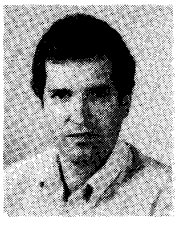
While at Purdue he has been employed as a Teaching and Research Assistant, and since 1988 as Manager of the School of Electrical Engineering Parallel Processing Laboratory. He worked at the Supercomputing Research Center, Lanham, MD, during the summer of 1987 and was one of 25 graduate researchers selected to attend the 1987 Summer Institute in Parallel Computing at Argonne National Laboratories, Argonne, IL.

Mr. Nation has published several papers in his areas of interest, which include interconnection networks, computer architecture, parallel processing systems, and parallel processor prototyping. He is a student member of IEEE and ACM, and a member of the Eta Kappa Nu and Tau Beta Pi honorary societies.



Clyde P. Kruskal (Member, IEEE) received the A.B. degree in mathematics and computer science from Brandeis University, Waltham, MA, in 1976, and the M.S. and Ph.D. degrees in computer science from the Courant Institute of New York University in 1978 and 1981, respectively.

He was an Assistant Professor in computer science at the University of Illinois–Urbana from 1981 to 1985. Since 1985 he has been on the faculty at the University of Maryland, where he is now an Associate Professor. His research interests include the analysis of sequential and parallel algorithms and the design of parallel computers. Dr. Kruskal is a member of the ACM and IEEE.



Leonard M. Napolitano Jr. received the B.S. degree in humanities and science, the B.S. degree in art and design, and the B.S. degree in civil engineering from the Massachusetts Institute of Technology, Cambridge, MA, in 1978; the M.S. degree in civil engineering from the Massachusetts Institute of Technology, Cambridge, MA, in 1979; and the Ph.D. degree in computer architecture for computational mechanics from Stanford University, Palo Alto, CA, in 1986.

Since 1979 he has been a member of the technical staff at Sandia National Laboratories, Livermore, CA. He currently leads research and development efforts for high-speed special-purpose computers, with particular emphasis on automatic target recognition and machine vision applications. His areas of interest include computer architecture, parallel processing, fault tolerance, and digital implementations of real-time systems.

Dr. Napolitano is a member of the IEEE Computer Society, ACM, and the Sigma Xi honorary society.