allel computer architecture in which the network is used. Two main parallel computer architectures exist (1). In the *physically shared-memory parallel computer, N* processors access *M* memory modules over an interconnection network as depicted in Fig. 1(a). In the *physically distributed-memory parallel computer,* a processor and a memory module form a processor–memory pair that is called *processing element* (PE). All *N* PEs are interconnected via an interconnection network as depicted in Fig. 1(b). In a *message-passing system,* PEs communicate by sending and receiving single messages (2), while in a *distributed-shared-memory system,* the distributed PE memory modules act as a single shared address space in which a processor can access any memory cell (3). This cell will either be in the memory module local to the processor, or be in a different PE that has to be accessed over the interconnection network.

Parallel computers can be further divided into SIMD and MIMD machines. In *single-instruction-stream multiple-data-stream* (SIMD) parallel computers (4), each processor executes the same instruction stream, which is distributed to all processors from a single control unit. All processors operate synchronously and will also generate messages to be transferred over the network synchronously. Thus, the network in SIMD machines has to support synchronous data transfers. In a *multiple-instruction-stream multiple-data-stream* (MIMD) parallel computer (5), all processors operate asynchronously on their own instruction streams. The network in MIMD machines therefore has to support asynchronous data transfers.

The interconnection network is an essential part of any parallel computer. Only if fast and reliable communication over the network is guaranteed will the parallel system ex-

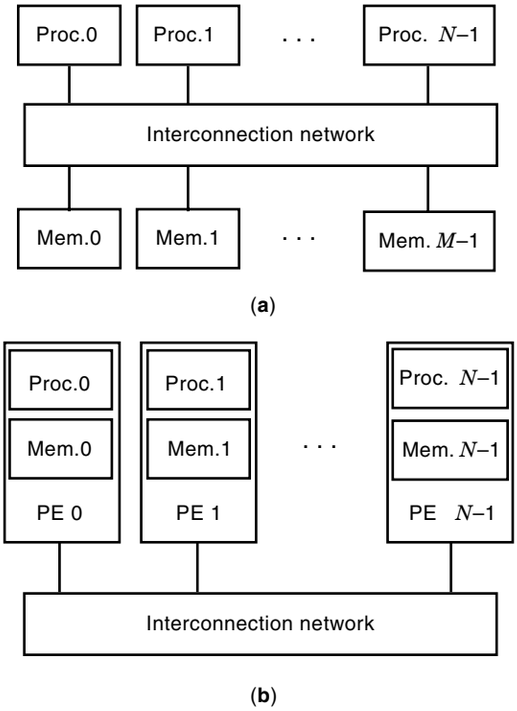

Figure 1. (a) Physically shared-memory and (b) distributed-memory parallel computer architecture.

INTERCONNECTION NETWORKS FOR PARALLEL COMPUTERS

The interconnection network is responsible for fast and reliable communication among the processing nodes in any parallel computer. The demands on the network depend on the par-

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering.*

hibit high performance. Many different interconnection networks for parallel computers have been proposed (6).

One characteristic of a network is its topology. In this article we consider only point-to-point (non-bus-based) networks in which each network link is connected to only two devices. These networks can be divided into two classes: direct and indirect networks. In direct networks, each switch has a direct link to a processing node or is simply incorporated directly into the processing node. In indirect networks, this one-to-one correspondence between switches and nodes need not exist, and many switches in the network may be attached only to other switches. Direct and indirect network topologies are discussed in the following section.

The mechanism to transfer a message through a network is called *switching.* A section below is devoted to switching techniques. Switching does not take into consideration the actual route that a message will take through a network. This mechanism is termed *routing,* and will be discussed in turn. In indirect networks, active switch boxes are used to transfer messages. Switch box architectures are discussed in a final section.

NETWORK TOPOLOGIES

Direct Networks

Direct networks consist of physical interconnection links that connect the nodes (typically PEs) in a parallel computer. Each node is connected to one or more of those interconnection links. Because the network consists of links only, routing decisions have to be made in the nodes. In many systems, dedicated router (switch) hardware is used in each node to select one of the interconnection links to send a message to its destination. Because a node is normally not directly connected to all other nodes in the parallel computer, a message transfer from a source to a destination node may require several steps through intermediate nodes to reach its destination node. These steps are called *hops.*

Two topology parameters that characterize direct networks are the degree and the network diameter. The *degree* Γ of a node is defined as the number of interconnection links to which a node is connected. Herein, we generally assume that direct network links are bidirectional, although this need not always be the case. Networks in which all nodes have the same degree n are called *n-regular* networks. The network *diameter* Φ is the maximum distance between two nodes in a network. This is equal to the maximum number of hops that a message needs to be transferred from any source to any destination node. The degree relates the network topology to its hardware requirements (number of links per node), while the diameter is related to the transfer delay of a message (number of hops through the network). The two parameters depend on each other. In most direct network, a higher degree implies a smaller diameter because with increasing degree, a node is connected to more other nodes, so that the maximum distance between two nodes will decrease.

Many different direct network topologies have been proposed. In the following, only the basic topologies are studied. Further discussion of other topologies can be found in Refs. 7–9.

In a *ring network* connecting N nodes, each node is connected to only two neighbors ($\Gamma = 2$), with PE i connected to PEs $i - 1 \bmod N$ and $i + 1 \bmod N$. However, the network has a large diameter of $\Phi = \lfloor N/2 \rfloor$ (assuming bidirectional links). Thus, global communication performance in a ring network will decrease with increasing number of nodes.

A direct network quite commonly used in parallel computers is the mesh network. In a *two-dimensional mesh,* the nodes are configured in an $M_X \times M_Y$ grid (with M_X nodes in the X direction and M_Y nodes in the Y direction), and an internal node is connected to its nearest neighbors in the north, south, east, and west directions. Each border node is connected to its nearest neighbors only. A 4×4 two-dimensional mesh connecting 16 nodes is depicted in Fig. 2(a). Because of the mesh edges, nodes have different degrees. In Fig. 2(a), the internal nodes have degree $\Gamma = 4$, while edge nodes have degree $\Gamma = 3$ and $\Gamma = 2$ (for the corner nodes). Because the edge nodes have a lower degree than internal nodes, the (relatively large) diameter of a two-dimensional mesh is $\Phi = (M_X - 1) + (M_Y - 1)$.

To decrease the network diameter, the degree of the edge nodes can be increased to $\Gamma = 4$ by adding edge links. The topology of a two-dimensional *torus network* is created by connecting the edge nodes in columns and rows, as depicted in Fig. 2(b). All nodes of this two-dimensional torus network have degree $\Gamma = 4$, and the network diameter is reduced to $\Phi = \lfloor M_X/2 \rfloor + \lfloor M_Y/2 \rfloor$.

The disadvantage of two-dimensional mesh networks is their large diameter, which results in message transfers over many hops during global communication, especially in larger networks. To further reduce the diameter, higher-dimensional meshes can be used. Figure 3(a) depicts a three-dimensional mesh with open edge connections connecting 27 nodes. Internal nodes have degree $\Gamma = 6$, while edge nodes have degree of $\Gamma = 5$, $\Gamma = 4$, or $\Gamma = 3$, depending on their position. The network diameter is equal to $\Phi = (M_X - 1) + (M_Y - 1) + (M_Z - 1)$, with M_i equal to the number of nodes in the i direction. This diameter can be further reduced if edge connections are added.

In a *hypercube network* that connects N nodes, each node has degree $\Gamma = n = \log_2 N$, where n is called the *hypercube dimension* (8). Each link corresponds to a cube function (10). The $cube_k$ *function* on an address (node number) complements the kth bit of that address. To describe the hypercube topology, the Hamming distance H can be used. The *Hamming*

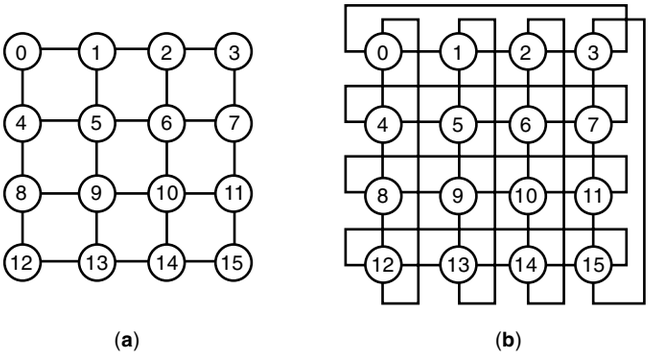

Figure 2. (a) Two-dimensional mesh network connecting 16 nodes, (b) torus network connecting 16 nodes. Because of the edge connections, the torus network has a uniform degree of four, while nodes in the mesh network have different degrees, depending on their location.

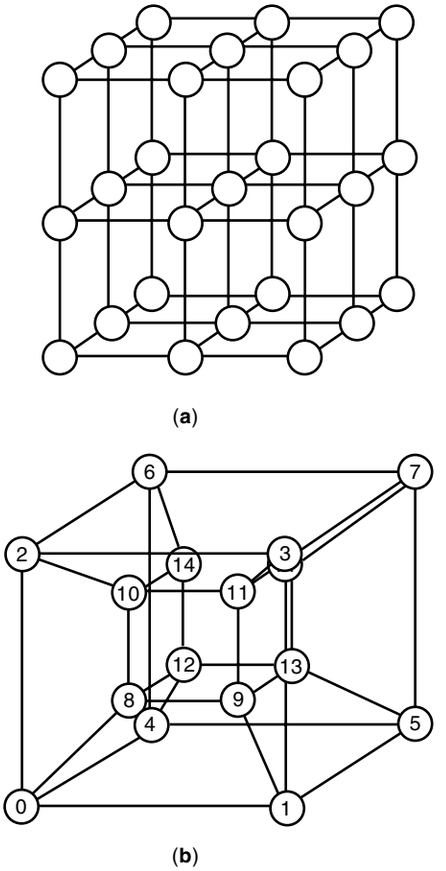

Figure 3. (a) Three-dimensional mesh connecting 27 nodes, (b) four-dimensional hypercube network connecting 16 nodes. In hypercube networks, the nodes that are directly connected have a Hamming distance of $H = 1$.

distance H between two binary numbers is defined in Ref. 11 as the number of bits in which the two numbers differ. Thus, two nodes are directly connected in a hypercube if their Hamming distance is $H = 1$ (the node numbers differ in exactly one bit). The number of hops that a message will take through the network is therefore equal to the Hamming distance between its source and destination addresses. In Fig. 3(b), a four-dimensional hypercube that connects 16 nodes is depicted. The diameter of a hypercube network is $\Phi = n$, because in the worst case, a source and a destination address of a message can differ in all n bits, so that all n cube functions have to be executed in order to transfer that message.

One disadvantage of a hypercube network concerns scalability. To increase the number of nodes a hypercube can interconnect, the degree of each node has to be incremented by at least one. Thus, to obtain the next larger hypercube, the number of nodes has to be doubled. To alleviate this scalability problem, *incomplete hypercubes* were introduced, in which any number of nodes can be interconnected (12).

To relate the different direct network topologies, the k-ary n-cube classification was introduced (13). A *k-ary n-cube network* connects $N = k^n$ nodes, where n is equal to the number of different dimensions the network consists of, while k is the network radix, which is equal to the number of nodes in each dimension. For example, a k-ary 1-cube is equivalent to a k-node ring network, a k-ary 2-cube is equivalent to a k^2-node torus network, and a 2-ary n-cube is equivalent to a 2^n-node n-dimensional hypercube. Figure 3(a) depicts a 3-ary 3-cube (assuming appropriate edge connections not shown in the figure), and Fig. 3(b) a 2-ary 4-cube. The diameter (Φ) is $n \times \lfloor k/2 \rfloor$.

Indirect Networks

In indirect networks, each processing node is connected to a network of switches over one or more (often bidirectional) links. Typically, this network consists of one or more stages of switch boxes; a network stage is connected to its successor and predecessor stage via a set of interconnection links. Depending on the number of stages, the number of switch boxes per stage, and the interstage interconnection topology, indirect networks provide exactly one path (*single-path networks*) or multiple paths (*multipath networks*) from each source to each destination.

Many different indirect network topologies have been proposed. This section is a brief introduction to multistage cube and fat-tree networks. Further discussion of these and other topologies can be found in Refs. 14–17.

One important indirect single-path network topology is the *generalized-cube network* topology (10), based on the cube interconnection function. A generalized-cube network that connects $N = 2^n$ sources with N destinations consists of $s = \log_B N$ stages of $B \times B$ switch boxes. The stages are numbered from $s - 1$ (stage next to the sources) to 0 (stage next to the destination). Each stage consists of N/B switch boxes; two consecutive stages are connected via N interconnection links. In Fig. 4(a), an 8×8 generalized-cube network comprising 2×2 switch boxes is shown, while Fig. 4(b) depicts a 16×16 generalized-cube network with 4×4 switches.

Consider the link labeling depicted in Fig. 4(a). The labels at the input (and output) side of each switch box differ in exactly one bit, which is bit k in stage k. Thus, if a message is routed straight through a switch box, its link number is not changed. If a message goes from the upper input to the lower output (or from the lower input to the upper output) at stage k, it moves to an output link that differs in bit k (the cube_k operation transforms the link number). Each stage corresponds to a specific cube function, and all n cube-functions can be applied to a message on its way through the network.

A simple distributed routing algorithm can be used to transfer messages through the network. As routing information, each message header contains its destination address (*destination-tag routing*). If a message enters a switch box in stage k, this switch box will examine the kth bit of the message destination address. This bit determines the switch box output port to which the message is destined. If the bit is 0, the message is destined to the upper output port; if it is 1 to the lower output port. This scheme can be easily extended to $B \times B$ switch boxes, using the kth digit of the radix B representation of the destination address to select one of the B switch output links.

In shared memory parallel computers, many messages are requests for memory data, which results in reply messages that send data back to the original source. Thus, a read request sent through the network to the memory has to include the destination address (memory address) and also the source address (the node number to with the data is to be sent back).

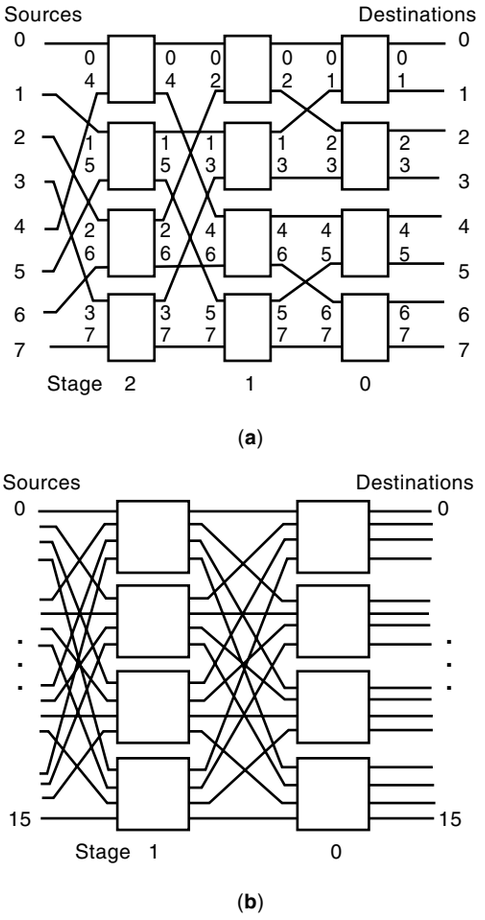

Figure 4. (a) 8×8 generalized-cube network comprising 2×2 switch boxes, (b) 16×16 generalized-cube network comprising 4×4 switch boxes. The link labels at the input (and output) side of each switch box in (a) differ in exactly one bit (bit k in stage k).

Thus, when destination-tag routing is used, the source address has to be added to the message header. This overhead can be avoided by using the XOR-routing algorithm. During *XOR routing,* an n-bit routing tag T that is formed by XORing the source and the destination address ($T = S \oplus D$) is added to each message as a message header. If a message enters a switch box in stage k, this switch box will examine the kth bit of the message routing tag T. If this bit is 0 (the corresponding source address bit is equal to the destination address bit), the message will be routed straight through that switch box (e.g., if it arrived at the upper input, it will be routed to the upper output). If the routing bit is 1, the switch will be set to exchange (e.g., if the message arrived at the upper input, it will be routed to the lower output). Once a message has arrived at its destination, the destination can determine the message's source address by XORing its own address with the message's routing tag T. XOR routing works in networks comprising 2×2 switch boxes only. A similar scheme can be used in hypercube networks.

Many different single-path multistage networks have been proposed in the literature, among them the SW-banyan, omega, indirect binary n-cube, delta, baseline, butterfly, and multistage shuffle-exchange networks. In Ref. 10 it was shown (by reordering switches and/or renumbering links) that instances of these networks are typically equivalent to the generalized-cube network topology.

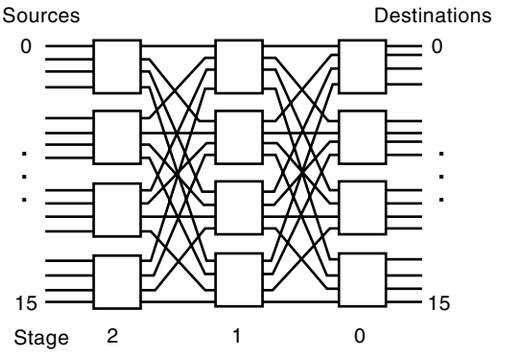

Figure 5. 16×16 three-stage multipath indirect network comprising 4×4 switch boxes. This network provides four link-disjoint paths from any source to any destination.

A generalized topology of a multipath indirect network is the three-stage network. This network consists of three stages of switches. Each switch box in the first and third network stages is connected to all switches in the network middle stage. A 16×16 multipath network comprising 4×4 switches is depicted in Fig. 5. The number of switches in the middle stage determines the number of distinct paths from each source to each destination (in Fig. 5, there are four distinct paths between any source and destination).

Another multipath indirect network topology that is used in parallel computers is the fat-tree network (18). The *binary fat-tree* network has the topology of a binary tree in which the leaves are connected to the processing elements and the root and intermediate tree nodes are switch boxes. All interconnection links are bidirectional. Unlike in an ordinary tree, the number of links between internal tree nodes is increasing when ascending the tree from the leaves to its root. Figure 6(a) depicts a binary fat-tree network that interconnects eight nodes. A cluster of processors is connected to the same switch box in the lowest switch level of the network (the switch level closest to the processors). This network provides only a single path that connects processors within a cluster. For all other connections, there exist multiple paths. To route a message between two nodes, the message first ascends in the network, rising to the lowest common ancestor of the source and desti-

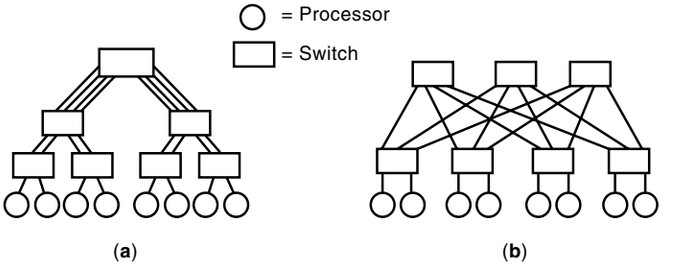

Figure 6. (a) Binary fat-tree network and (b) generalized fat-tree network connecting eight processors. This topology results in fast local communication, while the performance of global communication depends on the network size.

nation, and then descends to the destination. This indirect topology thus rewards local communication by providing shorter paths between nearer nodes.

A different network topology that is similar to a fat tree is shown in Fig. 6(b). As in the binary fat-tree network, only a single path connects two processors within a cluster. However, each switch box on the lower switch level is connected to all switches on the next higher level. Thus, the number of switches in the higher switch level determines the number of different paths between two processors in different processor clusters. More switch levels can be added to the network, which will increase the number of distinct paths among processors in different clusters. However, with each switch level, the message transfer delay will increase, because more switches have to be traversed by a message if the message is routed through higher switch levels.

SWITCHING TECHNIQUES

The mechanism to transfer a message through a network is called *switching*. Switching does not take into consideration the actual route that a message will take through a network (this mechanism is termed routing and will be discussed in the next section). The four fundamental and most-used switching techniques in interconnection networks are circuit switching, packet switching, wormhole routing, and virtual cut-through.

In a *circuit-switched* network, a complete connection through the network (from the source to the destination) is established before any data are sent. Network resources such as network links and switch ports are exclusively reserved for this connection. Once the connection is established, data are sent over the reserved network links and ports. After all data are sent, the established connection is disconnected to free the reserved resources for new connections. The connection establishment and disconnection can either be controlled centrally through a central network controller, or decentralized through messages that are sent through the network during connection establishment and disconnection. If a connection cannot be established because needed network resources are unavailable, the connection is refused (data cannot be transmitted) and the source has to try to establish the connection again.

In a *packet-switched* network, a message is divided into one or more data packets and routing information is added to each packet. These packets are sent through the network without the establishment of an overall connection between the source and destination. Network resources are reserved only when needed by a packet. Thus, network resources forming the path of a given packet that are not occupied by the given packet can be used to transfer other packets while the given packet is still in the network. This is impossible under circuit switching. The packet-switching technique is also called *store-and-forward packet-switching,* because a packet will be forwarded to the next node only if it was completely received by the current node. Therefore, nodes need enough space to buffer at least one complete packet. If a network resource such as a node's output port that a packet needs to use is unavailable (used by another message), the packet waits in its buffer within the node until the resource becomes available.

Wormhole routing is a switching technique similar to packet switching and is currently most often used in direct networks. In a wormhole-routed network, a message is divided into several *flow-control digits* (*flits*) (19). The first flit of a message (*header flit*) contains the message's routing information, and the last flit (*tail flit*) indicates its end. A message will be sent, flit by flit, in a pipelined fashion through the network. The header flit will reserve network resources exclusively for its message, and the tail flit will release each resource after it has passed it. Thus, the message will traverse a network like a worm through a hole. Depending on the message length (number of flits) and the length of the path the message takes through the network (number of intermediate nodes), the tail flit will be submitted to the network either while the head is still in the network, or when part of the message is already received by the destination.

If a header flit cannot acquire a network resource (e.g., an output port of an intermediate node), it has to be temporarily buffered in that node (normally at the input port of that node). This will stop the worm from advancing through the network. To minimize the network hardware, normally each input port of a node has the capability of buffering one or two flits only. Therefore, once a worm has stopped advancing through the network, each flit of the worm will wait in the node it currently resides in, without releasing any network resources. Thus, while a worm is blocked in a network, it will block the corresponding network resources from being used by other messages. This can result in deadlocks within the network, and the routing algorithm used in the network has to handle those situations (see the next section).

The *virtual-cut-through* (VCT) switching technique combines characteristics of store-and-forward packet switching and wormhole routing. Each data packet is divided into flits again and sent through the network, as is done during wormhole routing. However, each node has the capability to buffer a whole packet. If a flit reaches an empty node buffer and is not blocked, it will either be directly routed through the node or be buffered in that buffer for one flit cycle and then routed through the node (depending on the implementation). If a message is blocked and cannot be forwarded to the next node, all the flits of that message will be received one by one and buffered in that blocked node. Thus, under a light network load, VCT behaves similarly to wormhole routing. Under heavier loads, when blocking occurs more frequently, the message worm will be completely buffered in the blocked node, similarly to store-and-forward packet switching. This way, the message does not block resources of several nodes and will therefore block fewer messages in the network.

In Fig. 7, the data transport from a source to a destination through an intermediate node over time is shown for a circuit-switching, a store-and-forward packet-switching, and a wormhole-routing network (in the circuit-switching example, line propagation delays are neglected). It can be seen that circuit-switching and wormhole-routing networks behave similarly over time, while the packet transmission in a store-and-forward packet-switching network takes longer. As long as the header and tail parts of a message are much shorter than the message itself, the transmission time for a message in a wormhole-routing and circuit-switching network is virtually independent of the length of the path the message has to take through the network. Pipelining of the message bits or flits on the network interconnection links can further reduce the

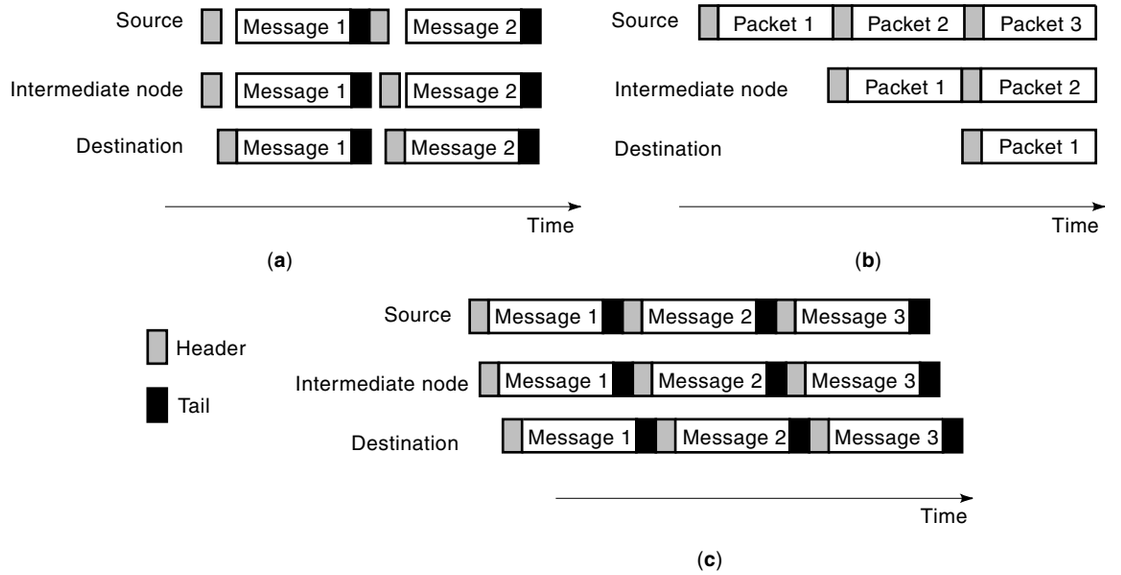

Figure 7. Data transport through an intermediate node in (a) a circuit-switching network, (b) a store-and-forward packet-switching network, and (c) a wormhole-routing network. Circuit switching and wormhole routing result in a shorter message transmission time, while packet-switching networks tend to have fewer message blockings.

transmission time. On the contrary, in a store-and-forward packet-switching network, the transmission time of a message is proportional to the length of the path through the network. This has to be weighted against the fact that blocked messages will normally block fewer other messages in a store-and-forward packet-switching network than in a wormhole-routing network, while in a circuit-switching network, connections might be refused due to internal blocking. As noted earlier, the behavior of virtual cut-through depends on the network load.

The main disadvantage of wormhole-routing networks is that a blocked message may spread over several nodes in the network and will then block several network links, which become unavailable for other messages. As an example, consider Fig. 8(a). Two interconnected wormhole switches are shown that have a flit buffer at each input port. Assume that a message is currently routed through switch 2 from port D to port E. This message blocks another message that enters switch 1 at port A, which is destined to port E as well. The head flit will wait in the flit buffer at input port C. However, this message blocks a third message entering switch 1 at port B that is destined to port F. In this example, two messages are blocked because port E is currently unavailable.

To alleviate this problem, *virtual channels* were introduced (20). As depicted in Fig. 8(b) each switch now has two parallel flit buffers per input port, resulting in two virtual channels that are multiplexed over one physical interconnection link. In this case, the message entering switch 1 at input port A is still blocked at input port C because it is destined to the busy output port E. However, the third message is able to use the second virtual channel at input port C, so that it can proceed to the idle output port F.

The concept of virtual channels enhances the performance of wormhole-routing networks substantially, especially when the data traffic consists of a mixture of short and long messages. Without virtual channels, long messages can block short messages for quite some time. However, short messages often result from time-critical operations such as synchronization, so that a short latency is crucial for those messages. Because message latency also includes blocking time, virtual channels result in a decreased latency because there is less message blocking in the network.

ROUTING TECHNIQUES FOR DIRECT NETWORKS

The network mechanism that selects certain network resources (e.g., a specific output port of a switch) in order to transfer a message from a source to a destination is termed *routing*. Routing can either be done through a centralized net-

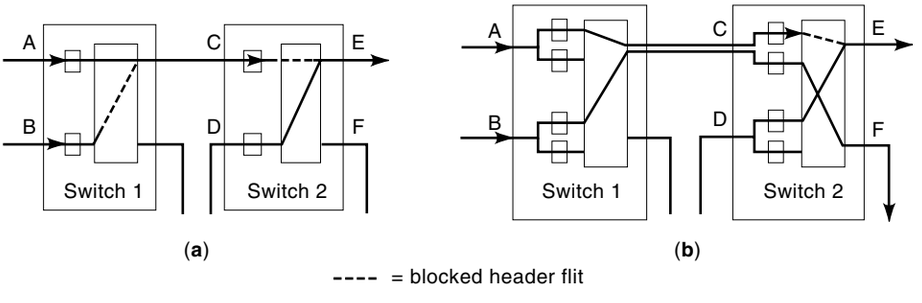

Figure 8. (a) Conventional wormhole-routing network, (b) wormhole-routing network with virtual channels. The virtual channels enhance the network performance substantially because fewer messages are blocked.

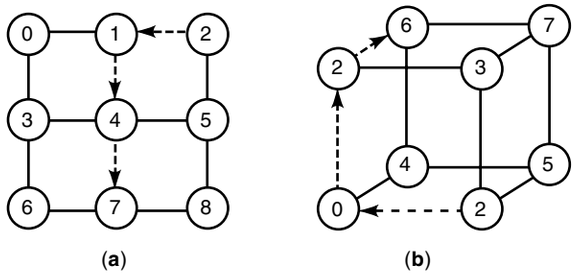

Figure 9. (a) *XY* routing in a mesh with $N = 9$, (b) *e*-cube routing in a hypercube with $N = 8$. Messages are routed in a dimension-ordered fashion.

work controller, or, as it is most often the case, decentralized in the individual network switches.

Routing algorithms can be either deterministic or adaptive. During *deterministic routing,* the path to be taken through the network is determined by the source and destination addresses only. The network load and the availability of network resources do not influence the routing of a message. *Adaptive routing protocols* take the availability of network links into account as well. To support adaptive routing, multiple paths between a source and a destination have to be present in the network.

Routing deadlock occurs when a set of messages has a cyclic dependency on resources (buffers or links). Because of the problem of deadlocks in direct networks, most routing algorithms have been proposed for direct networks to avoid deadlock situations. This section therefore focuses on routing algorithms for direct networks, and only a few basic algorithms are outlined here. Basic routing algorithms for indirect networks are covered in the subsection "Indirect Networks" of the section on "Network Topologies" above.

Deterministic Routing

The most common deterministic routing strategy used in direct networks is dimension-order routing in which a message traverses the network by successively traveling over an ordered set of dimensions of path. Two examples of dimension-ordered routine are *XY* routing and *e*-cube routing.

The *XY routing algorithm* used for mesh networks routes a message always in the X direction first. Once it has reached its destination column, the message will be routed in the Y direction (of course, this method also works if messages are routed in the Y direction first and then in the X direction). This routing strategy results in deadlock-free message delivery because cyclic dependences cannot occur (21). Consider the mesh network in Fig. 9(a), and assume *XY* routing (X dimension first, then Y dimension). A message from source 2 destined to node 7 will be routed through the intermediate nodes 1 and 4 as shown in the figure. If one of the network links on that path is blocked (e.g., the link between nodes 4 and 7), the message is blocked as well. An alternative path of the same length exists through nodes 5 and 8, but this path cannot be taken because of the *XY* routing algorithm. Thus, on the one hand, *XY* routing restricts the number of paths a message can take (and therefore increases the possibility of message blocking), but, on the other hand, guarantees deadlock freedom in the network (for a detailed explanation, see Ref. 19).

Similarly to the *XY* routing strategy in mesh networks, a message in a hypercube network under the *e-cube algorithm* will always traverse the dimensions of the network in the same order (e.g., cube_0, then cube_1, then cube_2, . . .). In Fig. 9(b), the transfer of a message from source node 1 to destination node 6 (over intermediate nodes 0 and 2) is shown in a hypercube with $N = 8$ using the e-cube algorithm. If a network resource on this path is blocked, the message has to wait, even though alternative paths exist (e.g., over intermediate nodes 5 and 7). However, cyclic dependences cannot occur when the e-cube algorithm is used, so that deadlocks are avoided (for a detailed explanation, see Ref. 21).

The *e*-cube algorithm, initially proposed for hypercube networks, can be generalized for k-ary n-cubes (21). The original *e*-cube algorithm cannot guarantee deadlock freedom in these networks because of inherent cycles due to the wrap-around edge connections (see the subsection "Direct Networks" under "Network Topologies" above). Thus, in order to avoid deadlocks, the routing algorithm is not allowed to used certain edge connections. This results in some message paths that are longer than in the network with unrestricted routing, but deadlock freedom is guaranteed.

Adaptive Routing

Adaptive routing protocols can be characterized by three independent criteria: progressive versus backtracking, profitable versus misrouting, and complete versus partial adaptation (22).

Once a routing decision is made in a *progressive protocol,* it cannot be reversed. The path has to be taken even if the message might end up being blocked. In a *backtracking protocol,* routing decisions can be reversed if they lead to the blocking of a message. Thus, if a message reaches a blocked network resource (e.g., a temporarily unavailable network link), the message will track back its path taken so far to try to find an alternative route that is not blocked. This method is mainly used in circuit-switching or packet-switching direct networks with bidirectional links between nodes that enable the backtracking. Backtracking protocols are not well suited for wormhole-routing networks, because a message can be spread over several nodes, which makes it difficult to backtrack the worm.

A *profitable protocol* (also called *minimal routing protocol*) will always choose a network resource (e.g., a node output) that guides the message closer to its destination. If a message encounters a blocked link, it can only use other links that result in the same path length through the network. If those links are blocked as well, the message has to wait. This results in a minimal length of the path a message will take through a network. This routing restriction is omitted in *misrouting protocols* (also called *nonminimal routing protocols*) so that a misroute is preferred over message blocking. Thus, the length of the path a message will take can be longer than the minimum path from the source to its destination.

The two above-mentioned criteria define classes of paths that the routing algorithm can choose from. *Completely adaptive routing protocols* can use any path out of a class, while *partially adaptive* ones can only use a subset of those paths (to avoid deadlock situations). Examples of a progressive and a backtracking completely adaptive routing protocol are now given.

A very simple adaptive progressive routing protocol with a profitable path choice is the *idle algorithm.* It is based on a deterministic routing scheme (e.g., *XY* or *e*-cube routing). If the deterministic routing scheme encounters a blocked node output port, the adaptive protocol will choose a different output port that will bring the message closer to its destination. This way, a message either reaches its destination or is blocked when no other output port is available that would bring the message closer to its destination. The resulting path will always be of minimal length, and the network performance will be increased over the deterministic routing scheme because a message is allowed to take alternative paths. However, this routing protocol is not deadlock-free. Thus, if a deadlock occurs, it has to be detected by the routing algorithm (e.g., through timeouts) and dissolved. Each occurring deadlock will decrease the network performance, though, so that it is more efficient to use an adaptive routing protocol that is inherently deadlock-free.

A backtracking routing algorithm allows a message to reverse routing steps to avoid the blocking of the message. Deadlocks cannot occur, because messages will rather backtrack than wait. To avoid a livelock situation (i.e., when a message is routed indefinitely through the network without ever reaching its destination), information about path segments already taken has to be added to the message or stored in the network nodes in a distributed fashion.

A simple backtracking algorithms is the *exhaustive profitable backtracking protocol.* This protocol performs a depth-first search of the network, considering profitable network links only. If a shortest path that is not blocked exists between a source and a destination, this routing algorithm will find it. The *k-family routing protocol* speeds up the path search through a two-phase algorithm. As long as the distance of a message from its destination is larger then the parameter k, a profitable search heuristic is used that considers a subset of all available shortest paths only. If the distance is lower than k, then the exhaustive profitable search is used, which considers all available shortest paths (22).

Both routing protocols forbid misrouting, so that a nonblocked path through the network cannot always be found. *Exhaustive misrouting backtracking protocols* will always find an existing nonblocked path in a network, because messages can be misrouted. However, the search itself can degrade the network performance, especially when a nonblocked path does not exist. In this case, the routing algorithm will search the whole network before it recognizes that a path does not exist. Thus, a message may stay inside the network for quite a while and will use network resources during the search that are then unavailable for other messages.

To alleviate this search problem, the *two-phase misrouting backtracking protocol* can be used. This protocol divides the search into two phases, similarly to the *k*-family routing protocol. Each phase is determined by the current distance between the message and its destination. If the distance is larger than a parameter d, then the protocol will use an exhaustive profitable search. If the message is closer to its destination than d, then the protocol switches to an exhaustive misrouting search. Because the second phase can route the message further away from its destination again, the search may switch between the two phases multiple times.

SWITCH BOX ARCHITECTURES

The architecture of the switch boxes depends on the underlying switching mechanism (see the section "Switching Techniques" above) and has a large effect on network performance. This section discusses architectural issues with respect to switch boxes and their effect on network performance.

When a connection is established in a circuit-switching network, each switch box is set in a specific switching state. For example, in the 2×2 switch boxes that are sometimes used to construct multistage indirect networks, there are four distinct settings for each switch: straight, exchange, upper broadcast, and lower broadcast. The *straight* setting connects the upper input port with the upper output port, and the lower input port with the lower output port. In the *exchange* setting, the upper input port is connected to the lower output port, while the lower input port is connected to the upper output port. Finally, in the *broadcast* setting, one of the input ports is connected to both switch output ports (in the *lower broadcast* the lower input port is chosen; in the *upper broadcast,* the upper input port). If during the connection establishment for a message transmission a switch box within the network already uses a setting that is different from the requested one, the connection cannot be established and will be refused. One way to implement 2×2 and larger switches is the *crossbar* [see Fig. 10(a)]. A $B \times B$ crossbar consists of

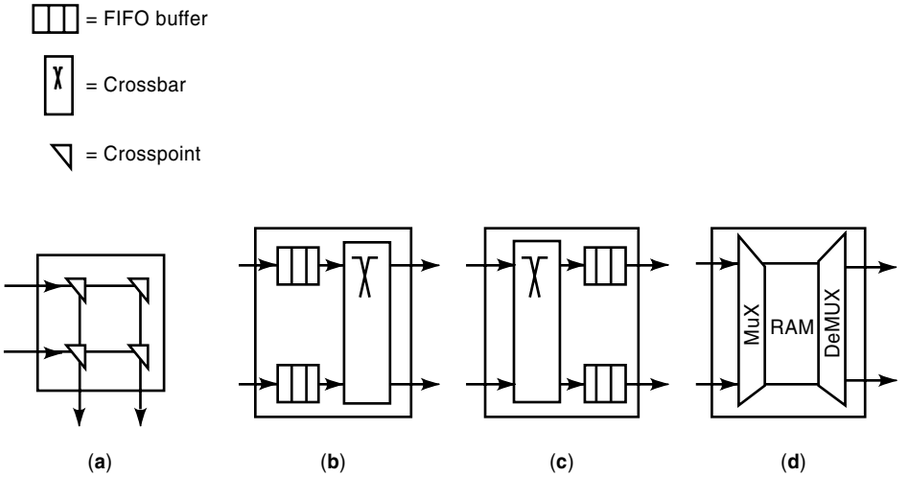

Figure 10. 2×2 (a) crossbar, (b) input-buffered, (c) output-buffered, and (d) central-memory-buffered switch box architectures. The placement of the buffers within a switch box has a major effect on the network performance and on the buffer requirements.

B inputs, B outputs, and B^2 crosspoints that can connect the horizontal line with the corresponding vertical one.

In packet-switching (and wormhole-routing) networks, packets (or flits) can be blocked within the network and have to be temporarily buffered inside a switch box. The placement of these buffers within a switch box has a major effect on the network performance and on the buffer requirements. The method that results in the lowest hardware requirement is *input buffering,* where a first-in-first-out (FIFO) buffer for storing multiple packets is placed at each input port of a switch box [see Fig. 10(b)]. During each network cycle, each buffer must be able to store up to one packet and dequeue up to one packet. A packet reaching a switch box input port that cannot be transferred to an output port because that port is currently busy will be stored in that input buffer. Although these buffers are easy to implement, they have the major disadvantage of *head-of-line* (HOL) blocking because of their FIFO discipline. If the packet at the head of an input buffer is blocked, it will block all other packets in that buffer, although some of those packets might be destined to an idle switch box output port. This blocking reduces the switch box throughput significantly, especially in larger switches.

To eliminate the HOL-blocking effect, *output buffering* can be employed, where FIFO buffers reside at each switch box output port [see Fig. 10(c)]. Because, during each network cycle, up to B packets can be destined to one specific output port in a $B \times B$ switch box (one from each switch box input), an output buffer must be able to store up to B packets and dequeue up to one packet during each network cycle. Because in an output buffer only packets are stored that are destined to the same output port of that switch box, HOL blocking cannot occur. If buffers with an infinite length are assumed, a maximum switch throughput of 100% can be achieved.

To achieve high performance with output buffered switch boxes, considerable buffer space is needed. To reduce this buffer requirement, a central memory can be used. In *central-memory-buffered switch boxes,* there are no dedicated buffers at either the switch input or the output ports. Packets arriving at a switch box input port are buffered in a central memory that is shared among all switch inputs [see Fig. 10(d)]. The central memory is divided into virtual FIFO queues of variable length (one for each output port) in which the packets are stored corresponding to their destination. The bandwidth requirement for the central memory is even higher than that for a buffer in an output buffered switch box, because during each network cycle, up to B packets have to be stored in the memory and up to B packets have to be read out of a $B \times B$ switch box. Because the length of each virtual queue is variable, virtual queues that are only lightly utilized require less memory and heavily utilized virtual queues can have more space (23). Thus, the buffer space can be very efficiently utilized, so that a smaller overall buffer space is needed as than for switch boxes with dedicated output buffers at each output port.

CONCLUSIONS

This article is a brief introduction to some of the concepts involved in the design of interconnection networks for parallel machines. See the references cited for more details. A reading list provides further sources of information.

BIBLIOGRAPHY

1. R. Duncan, A survey of parallel computer architectures, *IEEE Comput.,* **23** (2): 5–16, 1990.
2. W. C. Athas and C. L. Seitz, Multicomputers: Message-passing concurrent computers, *IEEE Comput.,* **21** (8): 9–24, 1988.
3. B. Nitzberg and V. Lo, Distributed shared memory: A survey of issues and algorithms, *IEEE Comput.,* **24** (8): 52–60, 1991.
4. M. Jurczyk and T. Schwederski, SIMD processing: Concepts and systems, in Y. Zomaya (ed.), *Handbook of Parallel and Distributed Computing,* New York: McGraw-Hill, 1996, pp. 649–679.
5. R. Duncan, MIMD architectures: Shared and distributed memory designs, in Y. Zomaya (ed.), *Handbook of Parallel and Distributed Computing,* New York: McGraw-Hill, 1996, pp. 680–698.
6. H. J. Siegel and C. B. Stunkel, Inside parallel computers: Trends in interconnection networks, *IEEE Comput. Sci. Eng.,* **3** (3): 69–71, 1996.
7. V. Cantoni, M. Ferretti, and L. Lombardi, A comparison of homogeneous hierarchical interconnection structures, *Proc. IEEE,* **79**: 416–428, 1991.
8. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes,* San Mateo, CA: Morgan Kaufmann, 1992.
9. I. Stojmenovic, Direct interconnection networks, in Y. Zomaya (ed.), *Handbook of Parallel and Distributed Computing,* New York: McGraw-Hill, 1996, pp. 537–567.
10. H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies,* 2nd ed., New York: McGraw-Hill, 1990.
11. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes,* Cambridge, MA: MIT Press, 1972.
12. H. P. Katseff, Incomplete hypercubes, *IEEE Trans. Comput.,* **C-37**: 604–608,1988.
13. W. J. Dally, Performance analysis of k-ary n-cube interconnection networks, *IEEE Trans. Comput.,* **C-39**: 775–785, 1990.
14. D. P. Agrawal, Graph theoretical analysis and design of multistage interconnection networks, *IEEE Trans. Comput.,* **C-32**: 637–648, 1983.
15. H. Ahmadi and W. E. Denzel, A survey of modern high-performance switching techniques, *IEEE J. Sel. Areas Commun.,* **7**: 1091–1103, 1989.
16. K. Y. Lee and D. Lee, On the augmented data manipulator network in SIMD environments, *IEEE Trans. Comput.,* **37**: 574–584, 1988.
17. H. J. Siegel et al., Using the multistage cube network topology in parallel supercomputers, *Proc. IEEE,* **77**: 1932–1953, 1989.
18. C. E. Leiserson, Fat-trees: Universal networks for hardware-efficient supercomputing, *IEEE Trans. Comput.,* **C-34**: 892–901, 1985.
19. L. M. Ni and P. K. McKinley, A survey of wormhole routing techniques in direct networks, *IEEE Comput.,* **26** (2): 62–76, 1993.
20. W. J. Dally, Virtual-channel flow control, *IEEE Trans. Parallel Distrib. Syst.,* **3**: 194–205, 1992.
21. W. J. Dally and C.L. Seitz, Deadlock-free message routing in multiprocessor interconnection networks, *IEEE Trans. Comput.,* **C-36**: 547–553, 1987.
22. P. T. Gaughan and S. Yalamanchili, Adaptive routing protocols for hypercube interconnection networks, *IEEE Comput.,* **26** (5): 12–23, 1993.
23. M. Jurczyk et al., Strategies for the implementation of interconnection network simulators on parallel computers, *Int. J. Comput. Syst. Sci. Eng.,* **13** (1): 5–16, 1998.

Reading List

Books That Cover Interconnection Networks

J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach,* Los Alamitos, CA: IEEE Computer Society Press, 1997.

F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes,* San Mateo, CA: Morgan Kaufmann, 1992.

I. D. Scherson and A. S. Youssef (eds.), *Interconnection Networks for High-Performance Parallel Computers,* Los Alamitos, CA: IEEE Computer Society Press, 1994.

T. Schwederski and M. Jurczyk, *Interconnection Networks: Structures and Properties* (in German), Stuttgart: Teubner, 1996.

H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies,* 2nd ed., New York: McGraw-Hill, 1990.

K. J. Thurber (ed.), *Tutorial: Distributed Processor Communication Architecture,* New York: IEEE Press, 1979.

A. Varma and C. S. Raghavendra (eds.), *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice,* Los Alamitos, CA: IEEE Computer Society Press, 1994.

C.-L. Wu and T. Y. Feng (eds.), *Tutorial: Interconnection Networks for Parallel and Distributed Computing,* Los Alamitos, CA: IEEE Computer Society Press, 1984.

Books and Articles That Cover Interconnection Networks in Commercial Parallel Processing Systems

J. Beecroft, M. Homewood, and M. McLaren, Meiko CS-2 interconnect Elan-Elite design, *Parallel Comput.,* **20**: 1626–1638, 1994.

T. Blank, The MasPar MP-1 architecture, *IEEE Int. Comput. Conf. CompCon,* 1990, pp. 20–24.

R. Esser and R. Knecht, Intel Paragon XP/S—architecture and software environment, in H. W. Meurer (ed.), *Supercomputer '93,* Berlin: Springer-Verlag, 1993.

K. Hwang, *Advanced Computer Architecture,* New York: McGraw-Hill, 1993.

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

R. E. Kessler and J. L. Schwarzmeier, Cray T3D: A new dimension for Cray Research, *IEEE Int. Comput. Conf. CompCon,* 1993, pp. 176–182.

N. Koike, NEC Cenju-3: A microprocessor-based parallel computer multistage network, *8th Int. Parallel Process. Symp.,* 1994, pp. 393–401.

C. B. Stunkel et al., The SP2 high-performance switch, *IBM Syst. J.,* **34** (2): 185–202, 1995.

L. W. Tucker and G. G. Robertson, Architecture and applications of the Connection Machine, *IEEE Comput.,* **21** (8): 26–38, 1988.

Papers That Cover Network Fault Tolerance

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

G. B. Adams III, D. P. Agrawal, and H. J. Siegel, A survey and comparison of fault-tolerant multistage interconnection networks, *IEEE Comput.,* **20** (6): 14–27, 1987.

G.-M. Chiu and S.-P. Wu, A fault-tolerant routing strategy in hypercube multicomputers, *IEEE Trans. Comput.,* **C-45**: 143–154, 1996.

M. Jeng and H. J. Siegel, Design and analysis of dynamic redundancy networks, *IEEE Trans. Comput.,* **C-37**: 1019–1029, 1988.

V. P. Kumar and S. M. Reddy, Augmented shuffle-exchange multistage interconnection, *IEEE Comput.,* **20** (6): 30–40, 1987.

R. J. McMillen and H. J. Siegel, Routing schemes for the augmented data manipulator network in an MIMD system, *IEEE Trans. Comput.,* **C-31**: 1202–1214, 1982.

K. Padmanabhan and D. H. Lawrie, A class of redundant path multistage interconnection networks, *IEEE Trans. Comput.,* **C-32**: 1099–1108, 1983.

Papers About Comparing Interconnection Networks

K. J. Liszka, J. K. Antonio, and H. J. Siegel, Problems with comparing interconnection networks: Is an alligator better than an armadillo? *IEEE Concurrency,* **5** (4): 18–28, 1997.

Papers About Trends in Interconnection Networks

H. J. Siegel and C. B. Stunkel, Inside parallel computers: Trends in interconnection networks, *IEEE Comput. Sci. Eng.,* **3** (3): 69–71, 1996.

MICHAEL JURCZYK
University of Missouri–Columbia

HOWARD JAY SIEGEL
Purdue University

CRAIG STUNKEL
IBM T. J. Watson Research Center

INTERCONNECTION OF NETWORKS. See INTERNETWORKING.

INTEREST OPERATORS. See FEATURE EXTRACTION.