A Survey of Wormhole Routing Techniques in Direct Networks

Lionel M. Ni and Philip K. McKinley
Michigan State University

Efficient routing of messages is critical to the performance of direct network systems. The popular wormhole routing technique faces several challenges — particularly flow control and deadlock avoidance.

Massively parallel computers with thousands of processors are considered the most promising technology to achieve teraflops computational power. Such large-scale multiprocessors are usually organized as ensembles of nodes, where each node has its own processor, local memory, and other supporting devices. These nodes may have different functional capabilities. For example, the set of nodes may include vector processors, graphics processors, I/O processors, and symbolic processors.

The way the nodes are connected to one another varies among machines. In a direct network architecture, each node has a point-to-point, or direct, connection to some number of other nodes, called neighboring nodes. Direct networks have become a popular architecture for constructing massively parallel computers because they scale well; that is, as the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system also increase. Figure 1 shows a generic multiprocessor with a set of nodes interconnected through a direct network.

Because they do not physically share memory, nodes must communicate by passing messages through the network. Message size may vary, depending on the application. For efficient and fair use of network resources, a message is often divided into packets prior to transmission. A packet is the smallest unit of communication that contains routing and sequencing information; this information is carried in the packet header. Neighboring nodes may send packets to one another directly, while nodes that are not directly connected must rely on other nodes in the network to relay packets from source to destination. In many systems, each node contains a separate router to handle such communication-related tasks. Although a router's function could be performed by the corresponding local processor, dedicated routers are used to allow overlapped computation and communication within each node.

Figure 2 shows the architecture of a generic node. Each router supports some number of input and output channels. Normally, every input channel is paired with a corresponding output channel. Internal channels connect the local processor/memory to the router. Although it is common to provide only one pair of internal channels, some systems use more internal channels to avoid a communication bottleneck between the local processor/memory and the router. External channels are used for communication between routers and, therefore, between nodes. In

0018-9162/93/0200-0062$03.00 © 1993 IEEE

this article, unless otherwise specified, the term *channel* will refer to an external channel. By connecting the input channels of one node to the output channels of other nodes, the topology of the direct network is defined. A packet sent between two nodes that are not neighboring must be forwarded by routers along multiple external channels. Usually, a crossbar is used to allow all possible connections between the input and output channels within the router. The sequential list of channels traversed by such a packet is called a path, and the number of channels in the path is called the path length.

The programmer of a multiprocessor based on a direct network can invoke various system primitives to send messages between processes executing on different nodes. Writing such a message-passing program has been traditionally difficult and error prone. Systems used in this manner have been referred to as message-passing multicomputers.[1] Recently, an alternative approach has been pursued, whereby a sophisticated compiler generates data-movement operations from shared-memory parallel programs. For a user, the shared-memory programming paradigm is usually simpler and more intuitive than dealing with the low-level details of message passing. Systems used in this manner have been referred to as scalable shared-memory multiprocessors.[2]

Whether a direct network system is used to support a message-passing or a shared-memory programming paradigm, the time required to move data between nodes is critical to system performance, as it effectively determines what granularity levels of parallelism are possible in executing an application program. A metric commonly used to evaluate a direct network system is *communication latency*, which is the sum of three values: start-up latency, network latency, and blocking time.

Start-up latency is the time required for the system to handle the packet at both the source and destination nodes. Its value depends mainly on the design of system software and the interface between local processors and routers. Start-up latency can be further decomposed into sending latency and receiving latency — the start-up latencies incurred at the sending node and the receiving node, respectively. The network latency equals the elapsed time after the head of a packet has entered the network at the source until the tail of the packet emerges from the network at the destination. Start-up latency and network latency are static for a given system; that is, the sum of their values reflects the latency of packets sent in the absence of other network traffic and transient system activities.

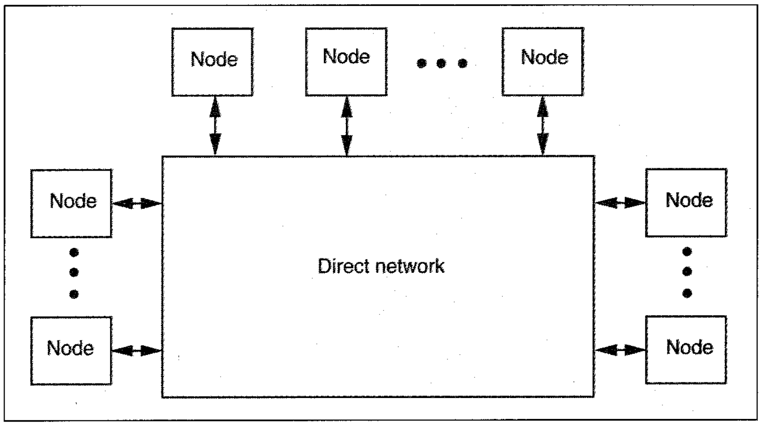


Figure 1. A generic multiprocessor based on a direct network.

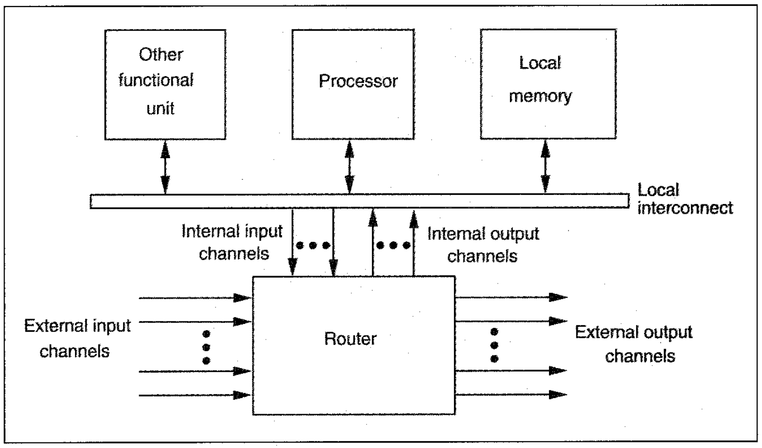


Figure 2. A generic node architecture.

The blocking time includes all possible delays encountered during the lifetime of a packet. These delays are due mainly to conflicts over the use of shared resources, for example, delays due to channel contention, in which two packets simultaneously require the same channel. Blocking time reflects the dynamic behavior of the network resulting from the passing of multiple packets; it may be high if the network traffic is heavy or unevenly distributed.

The communication latency of a direct network depends on several architectural characteristics; one of the most important is the type of switching technology used by routers to transfer data from input channels to output channels. A variety of switching techniques have been used in direct networks. One method, called *wormhole routing*,[3] has become quite popular in recent years. This article surveys the research contributions and commercial ventures related to wormhole routing. We review the properties of direct networks, then describe in detail the operation and characteristics of wormhole routing. By its nature, wormhole routing is particularly susceptible to deadlock situations, in which two or more packets may block one another indefinitely. Deadlock avoidance is usually guaranteed by the routing algorithm, which selects the path a packet takes. We describe several approaches to deadlock-free routing, along with a technique that allows multiple *virtual* channels to share the same physical channel. In addition, we discuss sev-

Direct network topologies

There are many ways to interconnect nodes in a direct network. Most of the popular direct network topologies fall in the general category of either *n*-dimensional meshes or *k*-ary *n*-cubes because their regular topologies simplify routing.

Formally, an *n*-dimensional mesh has $k_0 \times k_1 \times \ldots \times k_{n-2} \times k_{n-1}$ nodes, k_i nodes along each dimension i, where $k_i \geq 2$. Each node x is identified by n coordinates, $\sigma_{n-1}(x), \sigma_{n-2}(x), \ldots, \sigma_1(x), \sigma_0(x)$, where $0 \leq \sigma_i(x) \leq k_i - 1$ for $0 \leq i \leq n - 1$. Two nodes x and y are neighbors if and only if $\sigma_i(x) = \sigma_i(y)$ for all i, $0 \leq i \leq n - 1$, except one, j, where $\sigma_j(y) = \sigma_j(x) \pm 1$. Thus, nodes have from n to $2n$ neighbors, depending on their location in the mesh.

In a *k*-ary *n*-cube, all nodes have the same number of neighbors. The definition of a *k*-ary *n*-cube differs from that of an *n*-dimensional mesh in that all of the k_i's are equal to k and two nodes x and y are neighbors if and only if $\sigma_i(x) = \sigma_i(y)$ for all i, $0 \leq i \leq n - 1$, except one, j, where $\sigma_j(y) = (\sigma_j(x) \pm 1) \bmod k$. The use of modular arithmetic in the definition results in *wraparound* channels in the *k*-ary *n*-cube, which are not present in the *n*-dimensional mesh. A *k*-ary *n*-cube contains k^n nodes. If $k = 2$, then every node has n neighbors, one in each dimension. If $k > 2$, then every node has $2n$ neighbors, two in each dimension.

Several special cases of the *n*-dimensional meshes and *k*-ary *n*-cubes have been proposed or implemented as direct network topologies. When $n = 1$, the *k*-ary *n*-cube collapses to a ring with k nodes. The hypercube is a symmetric network and a special case of both the *n*-dimensional mesh and the *k*-ary *n*-cube. A hypercube is an *n*-dimensional mesh in which $k_i = 2$ for all i, $0 \leq i \leq n - 1$, that is, a 2-ary *n*-cube. Figure A1 depicts a binary 4-cube, or 16-node hypercube. When $n = 2$, the topology is a 2D torus with k^2 nodes. Figure A2 illustrates a 3-ary 2D torus. Figure A3 shows a 3 × 3 × 3 3D mesh, which results from removing the wraparound channels from a 3-ary 3-cube.

Hypercubes, low-dimensional meshes, and tori can be compared in terms of their bisection width η, that is, the minimum number of channels that must be removed to partition the network into two equal subnetworks. To make a fair comparison, we consider networks in which the number of nodes is $N = 2^{2n}$. Three possible network topologies containing this number of nodes are a $2n$-cube, a $2^n \times 2^n$ 2D mesh, and a $2^n \times 2^n$ 2D torus. The bisection widths of these three topologies are $\eta_{\text{hypercube}} = 2^{2n-1}$, $\eta_{\text{2D mesh}} = 2^n$, and $\eta_{\text{2D torus}} = 2^{n+1}$, respectively. The bisection density is the product of η and the channel width W and can be used as a measure of the network cost. If all of the networks have the same bisection density, then

$$W_{\text{2D mesh}}/W_{\text{hypercube}} = 2^{n-1} = \sqrt{N}/2$$

and

$$W_{\text{2D torus}}/W_{\text{hypercube}} = 2^{n-2} = \sqrt{N}/4.$$

In other words, for the same cost, the 2D mesh and the 2D torus can support wider channels and thereby offer higher channel bandwidth. However, the low-dimensional networks have larger diameters, which is the maximum distance between two nodes. The diameters of the topologies are $2n$, $2^{n+1} - 2$, and $2^n - 1$, respectively.

Both the hypercube and the torus are symmetric networks in that there exists a homomorphism that maps any node of the graph representing the network graph onto any other node.[1] Mesh networks, on the other hand, are asymmetric because the wraparound channels are absent. Assuming uniform traffic between nodes, channels near the center of the mesh are likely to experience higher traffic density than channels on the periphery. In addition, the network diameter is doubled. On the other hand, it is easier to provide deadlock-free routing in mesh networks than in torus networks. Furthermore, in a 2D layout, long wraparound wires connecting boundary nodes are eliminated to further reduce the wire complexity and increase the potential clock rate. In fact, the network bisection width is reduced by half.

Most direct network topologies used in wormhole-routed systems are low-dimensional meshes and hypercubes. The Intel Touchstone Delta, the Intel Paragon, and the Symult 2010 use a 2D mesh; the MIT J-machine and Caltech's Mosaic use a 3D mesh; and the Ncube-2/3 uses a hypercube.

Further information on direct network topologies is available from several sources. Reed and Fujimoto offer an analysis of many direct network topologies and discuss the treatment of many other issues related to multicomputer design.[1] Dally provides a performance analysis of *k*-ary *n*-cube topologies[2] and also deals with wire limit issues and constraints with regard to direct network topologies.[3]

References

1. D.A. Reed and R.M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, Cambridge, Mass., 1987.
2. W.J. Dally, "Performance Analysis of *k*-ary *n*-cube Interconnection Networks," *IEEE Trans. Computers*, June 1990, Vol. 39, No. 6, pp. 775-785.
3. W.J. Dally, "Wire-Efficient VLSI Multiprocessor Communication Networks," *Proc. Stanford Conf. Advanced Research in VLSI*, P. Losleben, ed., MIT Press, Cambridge, Mass., Mar. 1987, pp. 391-415.

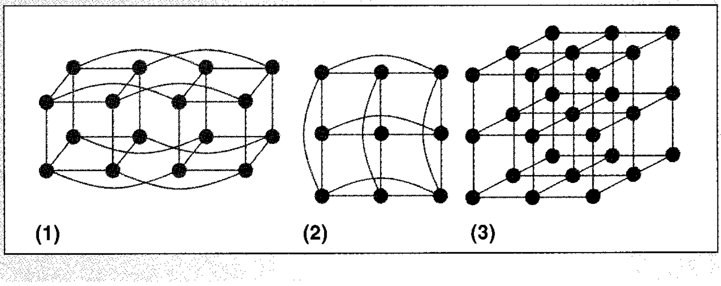


Figure A. Direct network topologies: (1) 2-ary 4-cube (hypercube); (2) 3-ary 2-cube (torus); (3) 3 × 3 × 3 3D mesh.

eral open issues related to wormhole routing.

Characteristics of direct networks

The average communication latency in a direct network depends on several network properties. A direct network is characterized by four factors: topology, routing, flow control, and switching.

Topology. The topology of a network, usually modeled as a graph, defines how the nodes are interconnected by channels. If every node is connected directly to every other node, the network topology is fully connected, or complete. Although complete topologies obviate forwarding of packets by intermediate nodes, they are practical only for very small networks because the number of physical connections per node is limited by rigid constraints, such as the number of available pins and pads on the router and the amount of VLSI area available for communication-related hardware. These engineering and scaling difficulties preclude networks with large complete topologies.

Therefore, many direct networks use a fixed, *multiple-hop* topology, such as a hypercube or two-dimensional mesh, each of which is a special case of k-ary n-cubes or n-dimensional meshes. (See the "Direct network topologies" sidebar for detailed definitions.) In multiple-hop topologies, packets may traverse one or more intermediate nodes before reaching the destination. Some computers provide basic cells that can be configured as different topologies, depending on the application. For example, if each router has one input channel and one output channel, the only feasible interconnection topology is a unidirectional ring. If each router has two input channels and two output channels, possible interconnection topologies include a bidirectional linear array and a bidirectional ring. With enough input and output channels, direct networks of arbitrary size and topology can be constructed.

Two conflicting requirements of a direct network are that it must accommodate a large number of nodes and exhibit a low network latency. As the number of nodes increases, the number of wires needed to interconnect them also increases. The complexity of the connection is said to be *wire limited:* the more edges in a topology, the more difficult that topology is to fabricate in a limited area.

Several parameters are used to study this problem. The *bisection width* of a topology is the minimum number of channels that must be removed, or cut, to partition the network into two subnetworks, each containing half the nodes in the network. The channel width is the number of bits that can be transmitted simultaneously on a physical channel between two adjacent nodes, and the channel rate is the peak rate at which bits can be transferred over each individual line of a physical channel. The channel bandwidth, which is the product of the channel width and the channel rate, determines the communication performance of a direct network. Bisection density, the product of bisection width and the channel width, can be used as a measure of network cost. For a given bisection density, a large bisection width dictates a small channel width.

For a given number of network nodes, low-dimensional mesh networks have much lower bisection widths than, say, hypercubes; consequently, they can offer wider channels and a higher channel bandwidth for a given bisection density (see "Direct network topologies" sidebar for details). A disadvantage of low-dimensional networks is that the average distance between nodes is relatively large. For systems in which the network latency depends on the path length, the hypercube is a popular choice of topology because of its relatively small internode distance. However, in other systems, such as those that support wormhole routing, the network latency is almost independent of the path length when there is no contention and the packet length is relatively large. Low-dimensional meshes are popular topologies for such systems because the negative effects of their large internode distance are minimized.

Routing. A direct network topology must allow every node to send packets to every other node. In the absence of a complete topology, routing determines the path selected by a packet to reach its destination. Efficient routing is critical to the performance of direct networks.

Routing can be classified in several ways. In *source routing*, the source node selects the entire path before sending the packet. Each packet must carry this routing information, increasing the packet size. Furthermore, the path cannot be changed after the packet has left the source. Most direct network systems use *distributed routing*. In this approach, each router, upon receiving the packet, decides whether it should be delivered to the local processor or forwarded to a neighboring router. In the latter case, the routing algorithm is invoked to determine which neighbor should be sent the packet. In a practical router design, the routing decision process must be as fast as possible to reduce the network latency. A good routing algorithm should also be easily implemented in hardware. Furthermore, the decision process usually does not require global state information of the network. Providing such information to each router creates additional traffic and requires additional storage space in each router.

Routing can also be classified as deterministic or adaptive. In deterministic routing, the path is completely determined by the source and destination addresses. This method is also referred to as oblivious routing. A routing technique is adaptive if, for a given source and destination, the path taken by a particular packet depends on dynamic network conditions, such as the presence of faulty or congested channels.

A routing algorithm is said to be minimal if the path selected is one of the shortest paths between the source and destination pair. Using a minimal routing algorithm, every channel visited will bring the packet closer to the destination. A nonminimal routing algorithm allows packets to follow a longer path, usually in response to current network conditions. If nonminimal routing is used, care must be taken to avoid a situation in which the packet will continue to be routed through the network but never reach the destination.

Flow control. A network consists of many channels and buffers. Flow control deals with the allocation of channels and buffers to a packet as it travels along a path through the network. A resource collision occurs when a packet cannot proceed because some resource that it requires is held by another packet. Whether the packet is dropped, blocked in place, buffered, or rerouted through another channel depends on the flow control policy. A good flow control policy should avoid channel congestion while reducing the network latency.

The allocation of channels and their associated buffers to packets can be viewed from two perspectives. The routing algorithm determines which output channel is selected for a packet arriving on a given input channel. Therefore, routing can be referred to as the *output selection policy*. Since an outgoing channel can be requested by packets arriving on many different input channels, an *input selection policy* is needed to determine which packet may use the output channel. Possible input selection policies include round robin, fixed channel priority, and first come, first served. The input selection policy affects the fairness of routing algorithms.

Switching techniques

Early direct networks used store-and-forward switching borrowed from the computer network community. In this approach, when a packet reaches an intermediate node, the entire packet is stored in a packet buffer. The packet is then forwarded to a selected neighboring node when the next output channel is available and the neighboring node has an available buffer. This switching strategy was adopted in the research prototype Cosmic Cube and several first-generation commercial multicomputers, including the iPSC-1, Ncube 1, Ametek 14, and FPS T-series. Store-and-forward switching is simple, but it has two major drawbacks. First, each node must buffer every incoming packet, consuming memory space. Second, the network latency is proportional to the distance between the source and destination nodes. The network latency is $(L/B)D$, where L is the packet length, B is the channel bandwidth, and D is the length of the path between the source and destination nodes.

To decrease the amount of time spent transmitting data, Kermani and Kleinrock introduced the virtual cut-through method for computer communication networks. In virtual cut-through, a packet is stored at an intermediate node only if the next required channel is busy. The network latency is $(L_h/B)D + L/B$, where L_h is the length of the header field. When $L >> L_h$, the second term, L/B, will dominate, and the distance D will produce a negligible effect on the network latency. The research prototype Harts, developed at the University of Michigan, is a hexagonal mesh multicomputer that adopts virtual cut-through switching. Felperin et al. provide a good survey of routing algorithms based on store-and-forward or virtual cut-through switching mechanisms.[1]

In circuit switching, a physical circuit is constructed between the source and destination nodes during the circuit establishment phase. In the packet transmission phase, the packet is transmitted along the circuit to the destination. During this phase, the channels constituting the circuit are reserved exclusively for the circuit; hence, there is no need for buffers at the intermediate nodes. In the circuit termination phase, the circuit is torn down as the tail of the packet is transmitted.

The network latency for circuit switching is $(L_c/B)D + L/B$, where L_c is the length of the control packet transmitted to establish the circuit. If $L_c << L$, the distance D has a negligible effect on the network latency. If a circuit cannot be established because a desired channel is being used by other packets, the circuit is said to be blocked. Depending on the way blocked circuits are handled, the partial circuit may be torn down, with establishment to be attempted later. This policy is called *loss mode*. Some second-generation multicomputers, such as Intel's iPSC-2 and iPSC/860, use circuit switching. Grunwald and Reed present a detailed performance study of different circuit switching techniques.[2]

Wormhole routing also uses a cut-through approach to switching. A packet is divided into a number of *flits* (flow control digits) for transmission. The header flit (or flits) governs the route. As the header advances along the specified route, the remaining flits follow in a pipeline fashion. If the header flit encounters a channel already in use, it is blocked until the channel becomes available. Rather than buffering the remaining flits by removing them from the network channels, as in virtual cut-through, the flow control within the network blocks the trailing flits and they remain in flit buffers along the established route. The network latency for wormhole routing is $(L_f/B)D + L/B$, where L_f is the length of each flit, B is the channel bandwidth, D is the path length, and L is the length of the message. If $L_f << L$, the path length D will not significantly affect the network latency unless it is very large.

Both computer networks and direct networks can implement and share the above switching techniques. Since cut-through switching does not have to buffer the entire packet before forwarding it to the next node, the data-link-level protocol is eliminated. This is good for direct networks, as the network complexity and overhead are further reduced. However, because of the high transmission error rate in computer networks, eliminating the data-link-level protocol will delay the detection of a transmission error. The error will be detected by an end-to-end acknowledgment provided at the transport layer. This is the main reason that cut-through switching is not normally used in computer networks.

Figure B compares the communication latency of wormhole routing with that of store-and-forward switching and circuit switching in a contention-free network. In this case, the behavior of virtual cut-through is the same as that of wormhole routing, so virtual cut-through is not shown explicitly. The channel propagation delay is typically small relative to L/B and is ignored here. For example, in the Intel Touchstone Delta, the time to transmit one flit on a channel is 75 nanoseconds for packets traveling in the same direction and 150 nanoseconds for packets that change direction. Even for a channel a foot long, the propagation delay is only 1.5 nanoseconds. The delay in transmitting a "ready" signal from the destination to the source in circuit switching is also ignored.

The figure shows the activities of each node over time when a packet is transmitted from a source node S to the destination node D through three intermediate nodes, $I1$, $I2$, and $I3$. The time required to transfer the packet between the source processor and its router, and between the last router and the destination processor, is ignored. Unlike store-and-forward switching, both circuit switching and

Switching. While the input and output selection policies determine how a packet uses channels as it traverses an intermediate router, switching is the actual mechanism that removes data from an input channel and places it on an output channel. Network latency is

wormhole routing have communication latencies that are nearly independent of the distance between the source and destination nodes.

This characteristic is confirmed by measurements on actual machines. Figure C plots the communication latency versus path length for a 1-Kbyte packet transmitted using three switching techniques: store-and-forward switching (on a 64-node Ncube-1 at Michigan State University), circuit switching (on a 32-node iPSC/2 at the University of Missouri–Rolla), and wormhole routing (on a 64-node Ncube-2 at Purdue University). The latencies of both circuit switching and wormhole routing demonstrate virtually no sensitivity to distance. In these measurements, the traffic is generated such that there is no channel contention. Thus, the communication latency does not include the blocking time.

The first commercial multiprocessor to adopt wormhole routing was the Ametek 2010, which used a 2D mesh topology. This machine was renamed Symult 2010 and ceased production in 1990. The Ncube-2, announced in 1989, also uses wormhole routing in a hypercube. Intel/DARPA's Touchstone Delta, delivered in 1991, uses wormhole routing based on a 2D mesh, as does the Intel Paragon, announced in 1991. The research prototype J-machine, built at the Massachusetts Institute of Technology in 1991, uses wormhole routing in a 3D mesh. Both Intel/CMU's iWarp and the Transputer IMS T9000 family use wormhole routing in their basic building-block nodes. Additional material and complete references for these machines are available in the literature.[3]

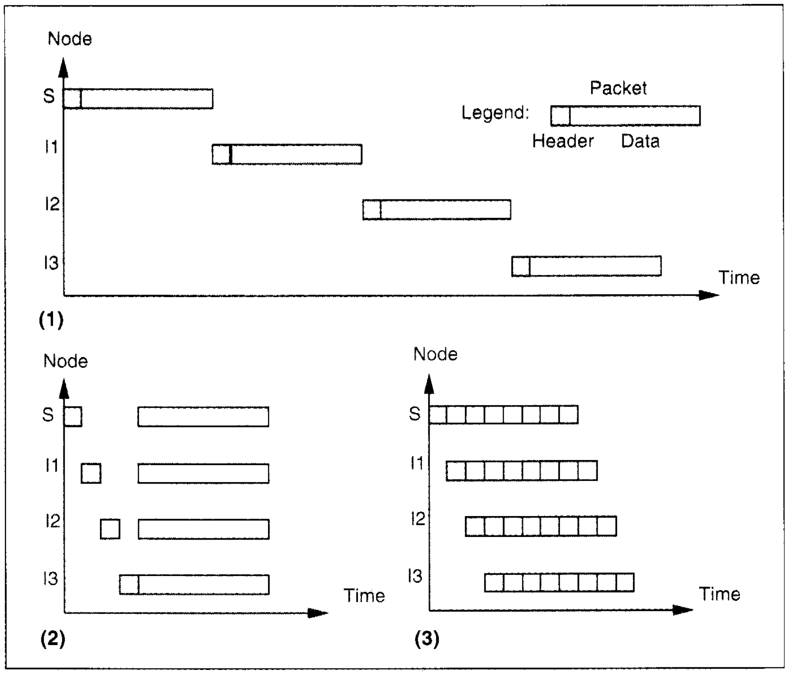


Figure B. Comparison of different switching techniques: (1) store-and-forward switching; (2) circuit switching; (3) wormhole routing.

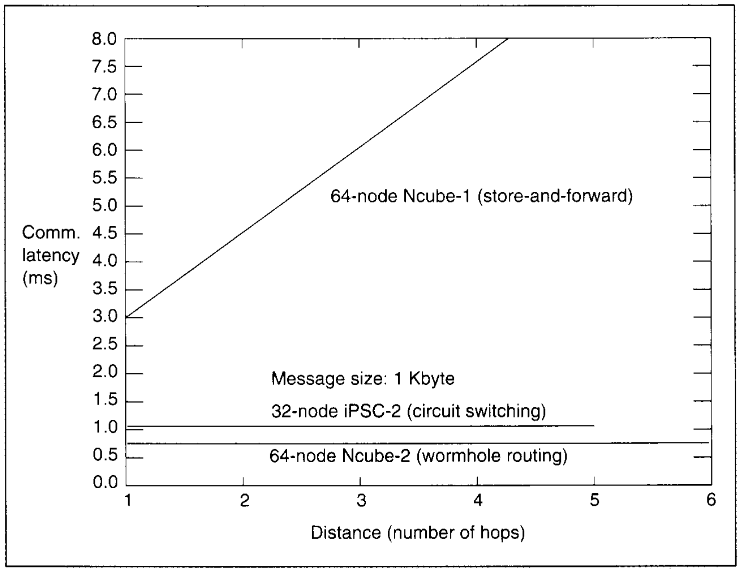


Figure C. Communication latency in milliseconds versus distance for transmitting a 1-Kbyte message.

References

1. S.A. Felperin et al., "Routing Techniques for Massively Parallel Communication," *Proc. IEEE*, Vol. 79, No. 4, Apr. 1991, pp. 488-503.

2. D.C. Grunwald and D.A. Reed, "Networks for Parallel Processors: Measurements and Prognostications," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, Vol. 1, ACM, New York, Jan. 1988, pp. 610-619.

3. L.M. Ni and P.K. McKinley, "A Survey of Routing Techniques in Wormhole Networks," Tech. Report MSU-CPS-ACS-46, Dept. of Computer Science, Michigan State University, East Lansing, Mich., Oct. 1991.

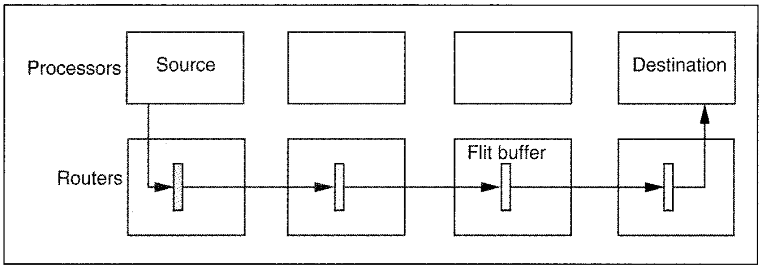


Figure 3. Wormhole routing.

highly dependent on the switching technique used. Four switching techniques have been adopted in direct networks: store-and-forward, circuit switching, virtual cut-through, and wormhole routing (see "Switching techniques" sidebar on pages 66-67). In store-and-forward switching, also called packet switching, when a packet reaches an intermediate node, the entire packet is stored in a packet buffer. The packet is then forwarded to a selected neighboring node when the next channel is available and the neighboring node has an available packet buffer. In circuit switching, a physical circuit is constructed between the source and destination nodes. After the packet has been transmitted along the circuit to the destination, the circuit is torn down. In virtual cut-through, the packet header is examined upon arrival at an intermediate node. The packet is stored at the intermediate node only if the next required channel is busy; otherwise, it is forwarded immediately without buffering.

Circuit switching and virtual cut-through are both based on the concept of cut-through, which can significantly reduce the network latency. Specifically, the delay introduced by each intermediate router is small. If the start-up latency (about 385 microseconds in Ncube-1 and 150 microseconds in Ncube-2) is very large relative to the delay at each router, the network latency contributes little to the communication latency unless the path is very long. However, as network traffic increases, the blocking time, which is a function of the path length, may become significant.

Wormhole routing

Although both virtual cut-through and circuit switching offer low network latencies that are relatively independent of path length, virtual cut-through requires that blocked packets be buffered, and circuit switching makes it difficult to support sharing of channels among packets. Wormhole routing, proposed by Dally and Seitz,[3] was designed to overcome these difficulties while offering similar network latency.

Wormhole routing also uses a cut-through approach to switching. A packet is divided into a number of *flits* (flow control digits) for transmission. The size of a flit depends on system parameters, in particular the channel width. Normally, the bits constituting a flit are transmitted in parallel between two routers. The header flit (or flits) of a packet governs the route. As the header advances along the specified route, the remaining flits follow in a pipeline fashion, as shown in Figure 3. If the header flit encounters a channel already in use, it is blocked until the channel becomes available. Rather than buffering the remaining flits by removing them from the network channels, as in virtual cut-through, the flow control within the network blocks the trailing flits and they remain in flit buffers along the established route. Once a channel has been acquired by a packet, it is reserved for the packet. The channel is released when the last, or tail, flit has been transmitted on the channel.

The pipelined nature of wormhole routing produces two positive effects. First, the absence of network conten-

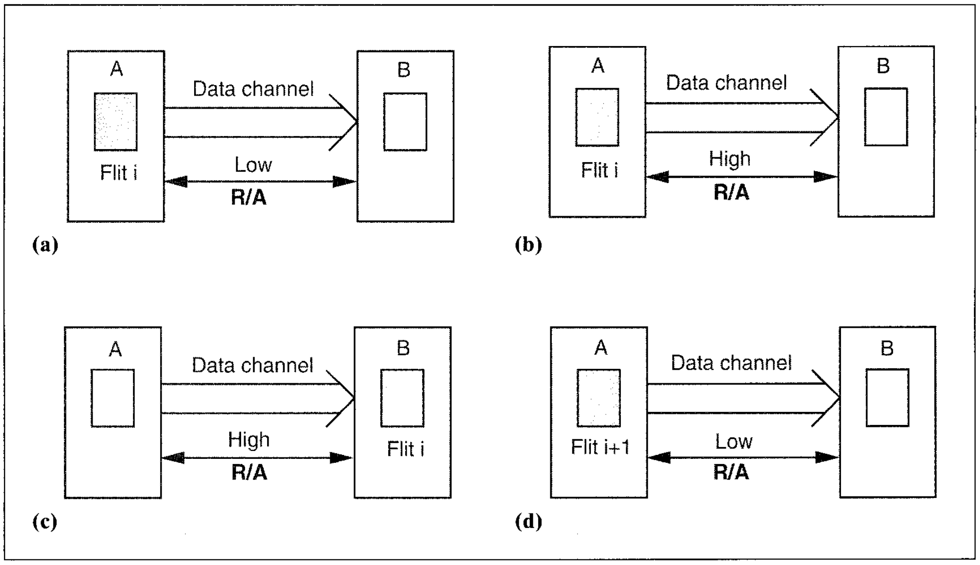


Figure 4. Handshaking between two routers through a request/acknowledge line: (a) *B* is ready to accept a flit by setting R/A to low; (b) *A* is ready to send flit *i* by raising R/A to high; (c) flit *i* is latched in *B*'s flit buffer; (d) *B* sets R/A to low when flit *i* is removed (also, *A* has received flit *i* + 1).

tion makes the network latency relatively insensitive to path length. Second, large packet buffers at each intermediate node are obviated; only a small FIFO (first in, first out) flit buffer is required. In some wormhole-routed systems, such as the Ncube-2 and Symult 2010, the flit buffer can hold only one flit. Other systems, such as the J-machine, a fine-grained system built at the Massachusetts Institute of Technology, have demonstrated improved network performance by using larger flit buffers. In the extreme, when the flit buffers are as large as the packets themselves, the behavior of wormhole routing resembles that of virtual cut-through.

If a large-scale wormhole-routed network is to be constructed, the effects of propagation delay make it difficult to distribute a high-speed synchronous clock to all nodes over a physically large area. Therefore, a popular approach has been self-timed circuit design,[4] in which flits passing between two adjacent nodes must use a handshaking protocol. In the example in Figure 4, a unidirectional channel from router *A* connects to router *B*. A single-wire request/acknowledge (R/A) line is associated with the channel. The R/A line can be raised only by router *A*, the requesting side, and lowered only by router *B*, the acknowledging side. When *A* is ready to send a flit to *B*, *A* must wait until the R/A line is low. *A* then places the data on the data channel and raises the R/A line to high. Router *B* will lower the R/A line when it has removed the flit from the flit buffer (or, in the case of large flit buffers, if there is an empty flit slot in the buffer).

The way wormhole-routed packets acquire and use channels leads to other advantages over circuit switching. In circuit switching, once a channel is assigned to a packet, it cannot be used by other packets until the channel is released. In contrast, wormhole routing allows a channel to be shared by many packets. We discuss this *virtual channel* concept later. Furthermore, wormhole routing allows packet replication, in which copies of a flit can be sent on multiple output channels. Packet replication is useful in supporting broadcast and multicast communication.[5] By its nature, circuit switching does not permit packet replication.

Wormhole routing has been a popular switching technique in new-generation direct networks. The first commercial multicomputer to adopt wormhole routing was the Ametek 2010, which used a 2D mesh topology. (This machine was later renamed the Symult 2010.) The Ncube-2, which uses a hypercube topology, has also adopted wormhole routing. The Intel Touchstone Delta and Intel Paragon use wormhole routing in a 2D mesh. Finally, MIT's research prototype J-machine uses wormhole routing in a 3D mesh.

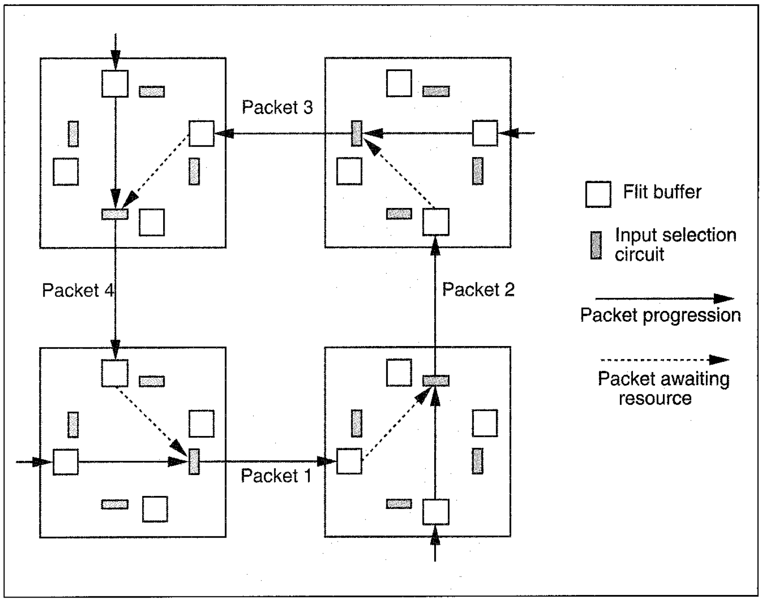


Figure 5. An example of channel deadlock involving four packets.

Deadlock

Switching strategy and the routing algorithm used are among several factors that affect communication latency. One situation that can postpone packet delivery indefinitely is deadlock, in which a set of packets may become blocked forever in the network. Deadlock can occur if packets are allowed to hold some resources while requesting others. In store-and-forward and virtual cut-through switching, the resources are buffers. In circuit switching and wormhole routing, the resources are channels. Because blocked packets holding channels (and their corresponding flit buffers) remain in the network, wormhole routing is particularly susceptible to deadlock. Figure 5 shows an example of channel deadlock involving four routers and four packets. Each packet is holding a flit buffer while requesting the flit buffer being held by another packet.

One way to solve the deadlock problem is to allow the preemption of packets involved in a potential deadlock situation. Preempted packets can be either rerouted or discarded. The former policy gives rise to adaptive nonminimal routing techniques. The latter policy requires that the packets be recovered at the source and retransmitted. Because of requirements for low latency and reliability, packet preemption is not used in most direct network architectures.

More commonly, deadlock is avoided by the routing algorithm. By ordering network resources and requiring that packets request and use these resources in strictly monotonic order, circular wait — a necessary condition for deadlock — is avoided. Hence, deadlock involving these resources cannot arise.

In wormhole-routed networks, channels are the critical resources. A *channel dependence graph*[6] can be used to develop a deadlock-free routing algorithm. The channel dependence graph for a direct network and a routing algorithm is a directed graph $D = G(C, E)$, where the vertex set $C(D)$ consists of all the unidirectional channels in the

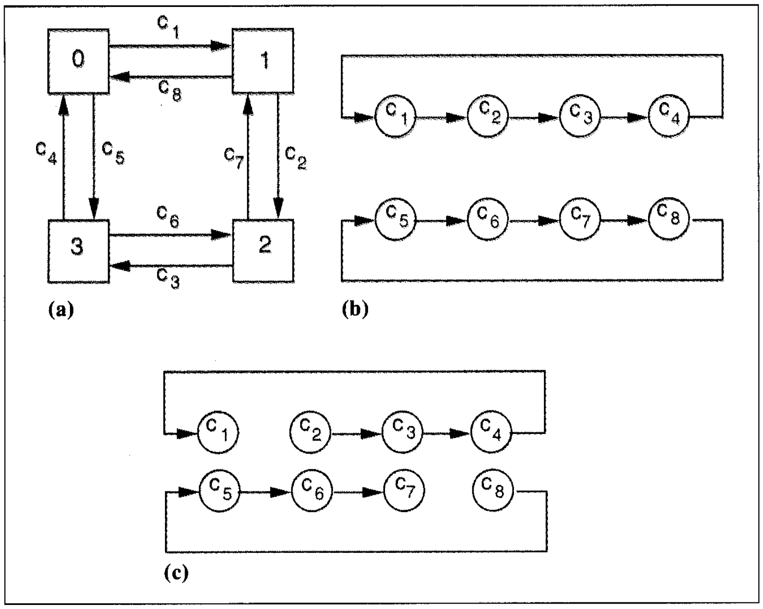


Figure 6. A four-node network and the corresponding channel dependence graphs: (a) a direct network with four nodes; (b) channel dependence graph; (c) channel dependence graph based on restricted minimal routing.

network, and the edge set $E(D)$ includes all the pairs of connected channels, as defined by the routing algorithm. In other words, if $(c_i, c_j) \in E(D)$, then c_i and c_j are, respectively, an input channel and an output channel of a node, and the routing algorithm may route packets from c_i to c_j. A routing algorithm for a direct network is deadlock-free if and only if there is no cycle in the channel dependence graph.[6]

Figure 6 demonstrates the channel dependence graph method. The four nodes shown in Figure 6a can be considered as a ring, a 2 × 2 mesh, a 2-cube, a 4-ary 1-cube, or a 2 × 2 torus. Assuming a packet can be delivered through any minimal routing path, the corresponding channel dependence graph is shown in Figure 6b. Since there are two cycles in the channel dependence graph, deadlock is possible. One way to avoid deadlock is to disallow packets to be forwarded from channel c_1 to c_2 and from c_7 to c_8. The resulting channel dependence graph is shown in Figure 6c. It can be easily verified that the routing is still minimal. However, to send a packet from node 0 to node 2, the packet must be forwarded through node 3, as the path through node 1 is no longer permitted.

Deterministic routing

One approach to designing a deadlock-free routing algorithm for a wormhole-routed network is to ensure that cycles are avoided in the channel dependence graph. This can be achieved by assigning each channel a unique number and allocating channels to packets in strictly ascending (or descending) order. If the behavior of the algorithm is independent of current network conditions, it is deterministic.

Dimension-ordered routing. A channel numbering scheme often used in n-dimensional meshes is based on the dimension of channels. In dimension-ordered routing, each packet is routed in one dimension at a time, arriving at the proper coordinate in each dimension before proceeding to the next dimension. By enforcing a strictly monotonic order on the dimensions traversed, deadlock-free routing is guaranteed. Hypercube and 2D mesh topologies each use a deadlock-free minimal deterministic routing algorithm. Both algorithms are based on the concept of dimension ordering.

In an n-cube, each node is represented using an n-bit binary number. Each node has n outgoing channels, and the ith channel corresponds to the ith dimension. In the E-cube routing algorithm, the packet header carries the destination node address d. When a node v in the n-cube receives a packet, the E-cube routing algorithm computes $c = d$

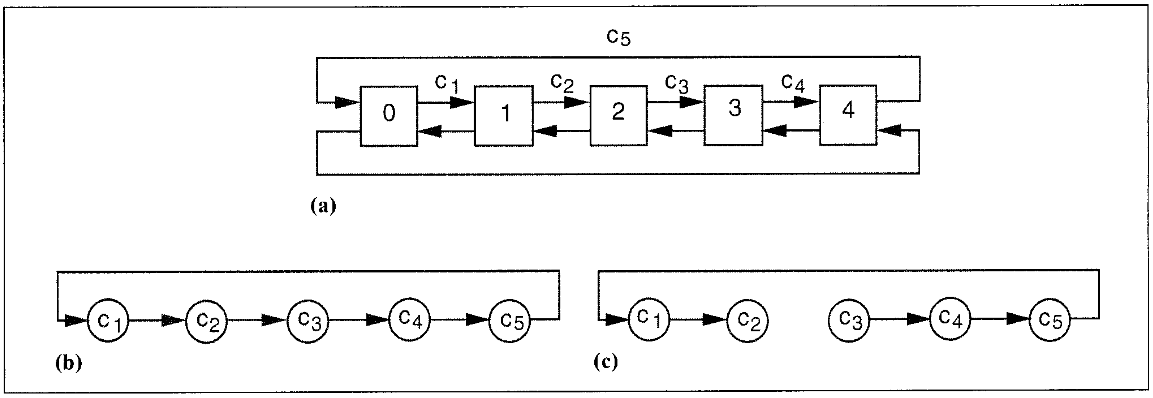


Figure 7. A five-node ring topology and channel dependence graphs: (a) 5-ary 1-cube (ring); (b) channel dependence graph; (c) channel dependence graph for deadlock-free nonminimal deterministic routing.

$\oplus v$, where $\oplus$ is the exclusive-OR operation. If $c = 0$, the packet is forwarded to the local processor. Otherwise, the packet is forwarded on the outgoing channel in the kth dimension, where k is the position of the rightmost (alternatively, leftmost) 1 in c.

In a 2D mesh, each node is represented by its position (x, y) in the mesh. In the XY routing algorithm, packets are sent first along the X dimension and then along the Y dimension. In other words, at most one turn is allowed, and that turn must be from the X dimension to the Y dimension. Let (s_x, s_y) and (d_x, d_y) denote the addresses of a source and destination node, respectively. Furthermore, let $(g_x, g_y) = (d_x - s_x, d_y - s_y)$. XY routing can be implemented by placing g_x and g_y in the first two flits, respectively, of the packet. When the first flit of a packet arrives at a router, it is decremented or incremented, depending on whether it is greater than 0 or less than 0. If the result is not equal to 0, the packet is forwarded in the same dimension and direction it arrived in. If the result equals 0 and the packet arrived on the Y dimension, the packet is delivered to the local processor. If the result equals 0 and the packet arrived on the X dimension, the flit is discarded and the next flit is examined upon arrival. If that flit is 0, the packet is delivered to the local processor; otherwise, the packet is forwarded in the Y dimension. Using this method, the largest possible 2D mesh with an 8-bit flit is 128 × 128. To construct a larger mesh, either the flit size must be increased or the flit buffer must be able to store multiple flits.

Routing in general k-ary n-cubes. For k-ary n-cube topologies with $k > 4$, it is impossible to construct a deadlock-free minimal deterministic routing algorithm. This result is true even when $n = 1$, as illustrated by the one-dimensional ring topology shown in Figure 7a, where $k = 5$. (The case of $k = 4$ was demonstrated in Figure 6, where deterministic minimal routing *is* possible.) Since only minimal routing is allowed, there are two disjoint channel dependence graphs. Figure 7b shows one of these; recall that the vertices represent channels, as labeled. To break the cycle, one of the edges must be deleted. However, in that case, minimal routing cannot be guaranteed. For example, if the edge between c_2 and c_3 is deleted, as shown in Figure 7c, then packets arriving at node 2 on channel c_2 cannot depart on channel c_3. Hence, packets sent from node 1 to node 3 must take a nonminimal path. Thus, a deadlock-free nonminimal deterministic routing algorithm is obtained. By using this technique, deadlock-free nonminimal deterministic routing algorithms can be developed for general k-ary n-cube topologies.[6]

Adaptive routing

The main disadvantage of deterministic routing is that it cannot respond to dynamic network conditions, such as congestion. An adaptive routing algorithm for a wormhole-routed network, however, must address the deadlock issue. To do so often requires the use of additional channels; in particular, some adjacent nodes must be connected by multiple pairs of opposite unidirectional channels. These pairs of channels may share one or more physical channels. The concept of virtual channels will be discussed later. To simplify the discussion, we will not distinguish between physical and virtual channels in this section.

Minimal adaptive routing. One general adaptive routing technique works by partitioning the channels into disjoint subsets. Each subset constitutes a corresponding subnetwork. Packets are routed through different subnetworks, depending on the location of destination nodes.

Figure 8 illustrates the application of this method to a 2D mesh. As Figure 8a shows, the mesh contains an additional pair of channels added to the Y dimension. The network can be partitioned into two subnetworks called the $+X$ subnetwork and the $-X$ subnetwork, each having a pair of channels in the Y dimension and a unidirectional channel in the X dimension. The $+X$ subnetwork is shown in Figure 8b. If the destination node is to the right of the source, that is, if $d_x > s_x$, the packet will be routed through the $+X$ subnetwork. If $d_x < s_x$, the $-X$ subnetwork is used. If $d_x = s_x$, the packet can be routed using either subnetwork.

This double Y-channel routing algorithm is minimal and fully adaptive; that is, a packet can be delivered through any of the shortest paths. The algorithm can be proved to be deadlock-free by ordering the channels appropriately.[7] Such an ordering of the channels in the $+X$ subnetwork is shown in Figure 8b. For any pair of source and destination nodes, the channels will be traversed in descending order, no matter which shortest paths are taken. Hence, deadlock cannot occur. In Figure 8b, for example, any of the minimal paths from node (1,0) to node (2,2) — specifically, (25, 24, 18), (25, 17, 14), and (16, 15, 14) — are valid.

Providing deadlock-free minimal fully adaptive routing algorithms for the hypercube, 2D torus, or more general k-ary n-cube topologies may require additional channels. Linder and Harden[7] have shown that a k-ary n-cube can be partitioned into 2^{n-1} subnetworks, $n + 1$ levels per subnetwork, and k^n channels per level. The number of additional channels increases rapidly with n. While this approach does provide minimal fully adaptive routing, the cost associated with

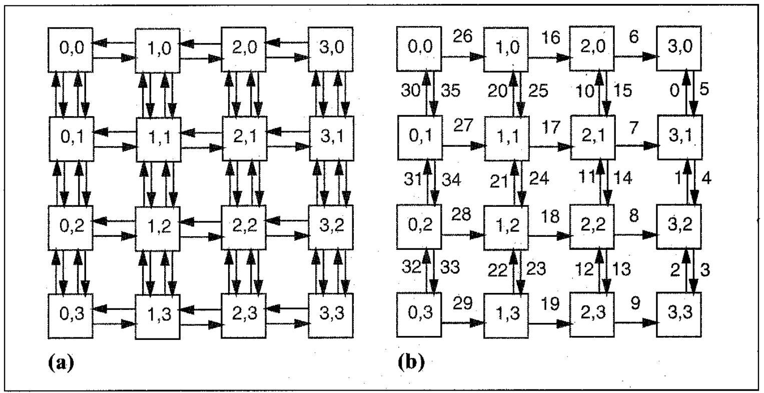


Figure 8. Adaptive double Y-channel routing for a 2D mesh: (a) double Y-channel 2D mesh; (b) $+X$ subnetwork and labeling.

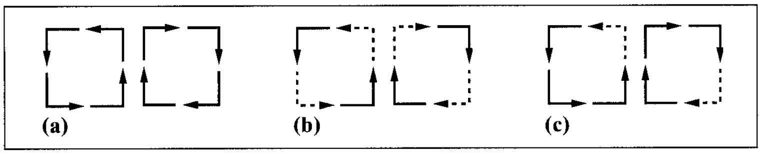


Figure 9. An illustration of the turn model in a 2D mesh: (a) abstract cycles in a 2D mesh; (b) four turns (solid arrows) allowed in XY routing; (c) six turns (solid arrows) allowed in west-first routing.

the additional channels makes it impractical when n is large.

Nonminimal adaptive routing. If minimal routing is not required, deadlock-free adaptive routing can be provided using fewer additional channels. If r pairs of channels connect every pair of adjacent nodes, then the following nonminimal adaptive routing algorithms, proposed by Dally and Aoki,[8] can be applied to k-ary n-cube and mesh topologies. Both of these algorithms allow the packets to take a longer path if there is no shortest path with all its channels available.

In the *static dimension reversal routing algorithm*, there are r pairs of channels between any two adjacent nodes. The network is partitioned into r subnetworks. The class-i ($0 \le i \le r - 1$) subnetwork consists of all the ith pair channels. The packet header carries an additional class field c initially set to 0. Packets with $c < r - 1$ can be routed in any direction in the class-c subnetwork; thus, the route may be nonminimal. However, each time a packet is routed from a high-dimensional channel to a low-dimensional channel, that is, reverse to the dimension ordering, the c field is increased by 1. Once the value of c has reached $r - 1$, the packet must use the deterministic dimension-ordered routing described earlier for the remainder of the path. The additional channels allow a packet to be routed in reverse dimension order. The parameter r limits the number of times this can happen, and hence dictates the degree of adaptivity of the routing algorithm.

In the *dynamic dimension reversal routing algorithm*, the channels are divided into two nonempty classes: adaptive and deterministic. Packets originate in the adaptive channels, where they can be routed in any direction with no limit on the number of times the packet can be routed in reverse dimension order. However, a packet with $c = p$ is not allowed to wait on a channel currently occupied by a packet with $c = q$ if $p \ge q$. A packet that reaches a node where all permissible output channels are occupied by packets whose values of c are less than or equal to its own must switch to the deterministic class of channels. When a packet enters the deterministic channels, it must follow the dimension-ordered routing described earlier and cannot reenter the adaptive channels. Since it is impossible for a circular wait to occur among packets in the adaptive channels, because of the way c is used, it can be easily shown that the algorithm is deadlock-free. An important design issue concerns how many channels are classified as adaptive and how many are deterministic between each pair of adjacent nodes.

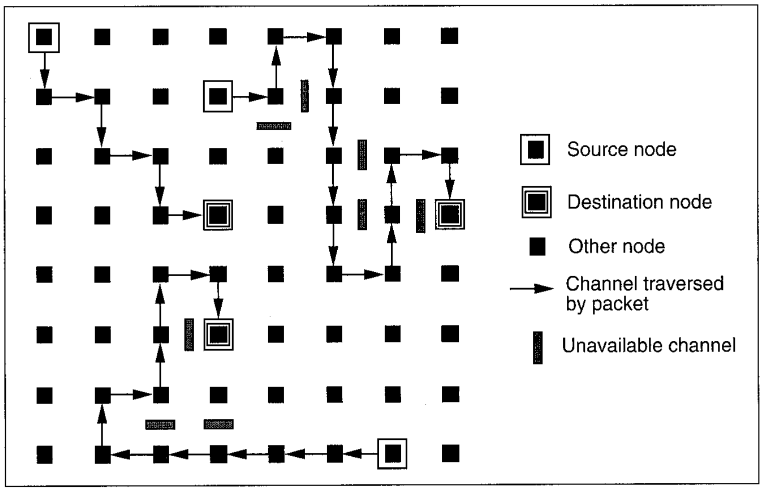


Figure 10. Examples of west-first routing in an 8 × 8 2D mesh.

The turn model. Given a network topology and the associated set of channels, adaptive routing algorithms are usually developed in an ad hoc way. The turn model proposed by Glass and Ni[9] provides a systematic approach to the development of maximally adaptive routing algorithms, both minimal and nonminimal, for a given network without adding channels. As Figure 5 shows, deadlock occurs because the packet routes contain turns that form a cycle. The following six steps can be used to develop maximally adaptive routing algorithms for n-dimensional meshes and k-ary n-cubes:

(1) Classify channels according to the direction in which they route packets.

(2) Identify the turns that occur between one direction and another, omitting 0-degree and 180-degree turns.

(3) Identify the simple cycles these turns can form.

(4) Prohibit one turn in each cycle.

(5) In the case of k-ary n-cubes, incorporate as many turns as possible that involve wraparound channels.

(6) Add 180-degree and 0-degree turns, which are needed for nonminimal routing algorithms or if there are multiple channels in the same direction.

The case of a 2D mesh illustrates the use of the turn model. There are eight possible turns and two possible abstract cycles, as shown in Figure 9a. Cycles among packets may result if the turns are not restricted, as illustrated in Figure 5. The deterministic XY routing algorithm prevents deadlock by prohibiting four of the turns, as shown in Figure 9b. The remaining four turns cannot form a cycle, but neither do they allow any adaptiveness.

The fundamental concept behind the turn model is to prohibit the smallest number of turns such that cycles are prevented. In fact, for a 2D mesh, only two turns need to be prohibited. Figure 9c shows six turns allowed, suggesting the corresponding *west-first routing algorithm:* First route a packet west, if necessary, and then adaptively south, east, and north. The two turns prohibited in Figure 9c are the two turns to the west. Therefore, to travel west, a packet must begin in that direction.

Figure 10 shows three example paths for the west-first algorithm. The channels marked as unavailable are either faulty or being used by other packets. One of the paths shown is minimal, while the other two paths are nonminimal, resulting from routing around unavailable channels. Because cycles are avoided, west-first routing is deadlock-free. For minimal routing, the algorithm is fully adaptive if the destination is on the right-hand side (east) of the source; otherwise, it is deterministic. If nonminimal routing is allowed, the algorithm is adaptive in either case. There are other ways to select six turns so as to prohibit cycles, although the selection of the two prohibited turns is not arbitrary.[9]

By applying the turn model to the hypercube, an adaptive routing algorithm, namely P-cube routing, can be developed. Let $s = \sigma_{n-1}(s), \sigma_{n-2}(s), \ldots, \sigma_0(s)$ and $d = \sigma_{n-1}(d), \sigma_{n-2}(d), \ldots, \sigma_0(d)$ be the source and destination nodes, respectively, in an n-cube. The set E consists of all the dimension numbers in which s and d differ. The size of E is the Hamming distance between s and d. Thus, $i \in E$ if $\sigma_i(s) \neq \sigma_i(d)$. E is divided into two disjoint subsets, E_0 and E_1, where $i \in E_0$ if $\sigma_i(s) = 0$ and $\sigma_i(d) = 1$, and $j \in E_1$ if $\sigma_i(s) = 1$ and $\sigma_i(d) = 0$.

The fundamental concept of P-cube routing is to divide the routing selection into two phases. In the first phase, a packet is routed through the dimensions in E_0, in any order. In the second phase, the packet is routed through the dimensions in E_1. A similar algorithm was proposed by Konstantinidou[10]; however, the P-cube routing algorithm can be systematically generalized to handle nonminimal routing as well.[9]

Routing in reconfigurable networks

In the examples cited thus far, one or more routing algorithms have been developed for each type of direct network topology. It is possible for the network topology itself to be reconfigurable. For example, by using basic building-block nodes such as Intel/CMU's iWarp cells or elements of the Transputer IMS T9000 family, different network topologies can be constructed from a given set of components. In this case, the router must be flexible or programmable to allow for the implementation of different deadlock-free routing algorithms. Two techniques are general enough to accommodate any topology, given a specific routing algorithm for each topology, while permitting the router design to be relatively simple.

Source routing. The first approach is source routing, mentioned earlier. Depending on the underlying network topology, the source node specifies the routing path on the basis of a deadlock-free deterministic routing algorithm. The packet must carry complete routing information in the packet header. Since the header itself must be transmitted through the network, thereby consuming network bandwidth, it is important to minimize header length.

One source-routing method that achieves this goal is called street-sign routing. The header is analogous to a set of directions given to a driver in a city. Only the names of the streets that the driver must turn on, along with the direction of the turn, are needed. Street-sign routing is used in iWarp, where each router has four pairs of channels corresponding to four cardinal directions ($+X, -X, +Y, -Y$). By default, packets arriving from the input channel in $+X$ (or $+Y$) will be forwarded to the output channel in $-X$ (or $-Y$), and vice versa. The source overrides this default by including in the header the addresses of all nodes at which a different action is to be taken.

There are two possible actions. The packet has either reached the destination or it must make a turn. For each turn, the header must contain the node address and the direction of the turn. Furthermore, this information must occur in the header according to the order in which nodes are reached. Upon receiving each header flit, the router compares the node address in the flit to the local node address. If they match, the packet either turns or is sent to the destination, as specified in the header flit; otherwise, the packet will be forwarded through the default output channel. By incorporating the concept of a default direction, the packet header can be kept short, requiring less time to generate and transmit. An appropriate header-generation algorithm can be designed for each of the possible topologies that may be configured.

Table-lookup routing. Another approach that is amenable to reconfigurable topologies is to perform routing by using table lookup. An obvious implementation is to place a lookup table at each node, with the number of entries in the table equal to the number of nodes in the network. Given a destination node address carried in the header, the corresponding entry in the table indicates which outgoing channel should be used to forward the packet. Such an implementation is not practical, however, because the size of the lookup table places an artificial upper bound on the network size, and the large table is inefficient in the use of chip area.

One way to reduce the table size is to define a range of addresses to be associated with each outgoing channel.[5] For example, each node of the 4 × 3 2D mesh shown in Figure 11a is assigned a label $\ell(x, y)$. Consider the node (1,1), which is labeled 4. Let d be the label of

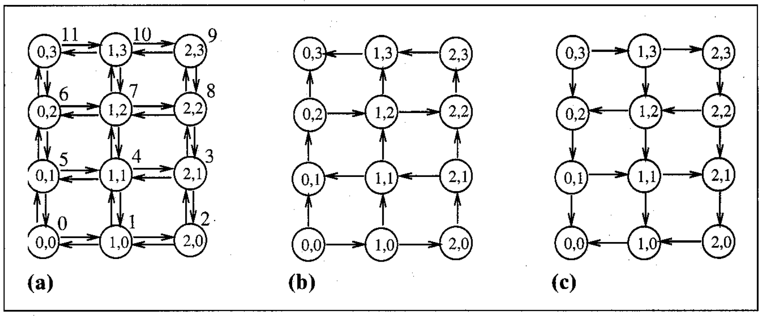


Figure 11. The labeling of a 4 × 3 mesh: (a) physical network; (b) high-channel network; (c) low-channel network.

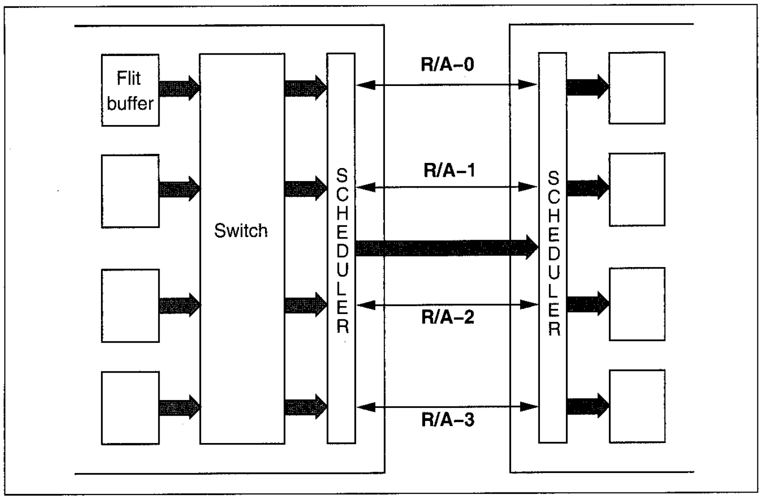


Figure 12. Four virtual channels share a unidirectional physical channel.

the destination address in a packet. Each routing table requires only four entries, one for each outgoing channel. For example, the routing table at node (1,1) will contain the following information. For $d \geq 7$, the packet will be routed using the $+Y$ channel. For $5 \leq d < 7$, $1 < d \leq 3$, and $d \leq 1$, the packet will be routed through channels $-X$, $+X$, and $-Y$, respectively.

In table-lookup routing, the most important issue is how to assign appropriate labels to nodes so that minimal, deadlock-free routing results. One such strategy for a 2D mesh topology with N nodes assigns a label to each node on the basis of its position in a particular Hamiltonian path of the network graph[5]; the first node in the path is labeled 0, and the last node in the path is labeled $N-1$. A label assignment function ℓ for an $m \times n$ mesh that results in minimal routing is

$$\ell(x,y)=\begin{cases} y*n+x & \text{if } y \text{ is even} \\ y*n+n-x-1 & \text{if } y \text{ is odd} \end{cases}$$

This labeling effectively divides the network into two subnetworks, shown in Figures 11b and 11c. The *high-channel subnetwork* contains all of the channels whose direction is from lower labeled nodes to higher labeled nodes, and the *low-channel subnetwork* contains all of the channels whose direction is from higher labeled nodes to lower labeled nodes. Since both subnetworks are acyclic, it is easily shown that this table-lookup routing algorithm is deadlock-free.

A deadlock-free table-lookup routing algorithm for the hypercube is also given by Lin and Ni.[5] For the hypercube, the label assignment function ℓ for a node with address $d_{n-1}\, d_{n-2} \ldots d_0$ is

$$\ell\left(d_{n-1}d_{n-2}\ldots d_0\right)=\sum_{i=0}^{n-1}\left(c_i\bar{d}_i 2^i+\bar{c}_i d_i 2^i\right)$$

where $c_{n-1}=0$, $c_{n-j}=d_{n-1}\oplus d_{n-2}\oplus \ldots \oplus d_{n-j+1}$ for $1 < j \leq n$. A similar technique is used in the Inmos IMS T9000 transputer, where it is referred to as interval labeling.

Virtual channels

Some adaptive routing algorithms require multiple pairs of channels between adjacent nodes. Implementing each channel in a wormhole-routed network with a separate set of physical wires is very expensive. Furthermore, in most applications the channel utilization is not high. One way to address this problem is to multiplex several virtual channels on a single physical communication channel. Each virtual channel has its own flit buffer, control, and data path.[11] In some designs, such as those of the Intel Touchstone and Intel/CMU iWarp, several unidirectional channels in the same direction share a single physical unidirectional channel; in other designs,[4] unidirectional virtual channels in opposite directions share a physical bidirectional channel.

Through virtual channels, a physical network can be divided into multiple disjoint logical networks, thereby facilitating adaptive routing algorithms. Virtual channels are useful in three other ways. First, by increasing the degree of connectivity in the network, they facilitate the mapping onto a particular physical topology of applications in which processes communicate according to another logical topology. For example, an application in which processes communicate according to a hexagonal array can be mapped onto a 2D mesh. Second, even when the application and the architecture have the same topology, extra connections may still be needed to route around congested or faulty nodes. Third, virtual channels provide the ability to deliver guaranteed communication bandwidth to certain classes of packets. For example, it is important that some bandwidth be reserved to support system-related functions, such as debugging, monitoring, and system diagnosis. By time-multiplexing virtual channels onto physical channels using a fair schedule, availability of some minimum bandwidth can be guaranteed to each virtual channel as long as the number of virtual channels sharing the same physical channel is bounded.

The most important issue concerning virtual channels is the multiplexing and arbitration of a physical channel among many virtual channels. The multiplexing technique should be designed to maximize channel utilization. Specifically, if m virtual channels share a physical channel with bandwidth W, and k virtual channels are active, where $1 \leq k \leq m$, then each active virtual channel should have an effective bandwidth of W/k. Since the number of active virtual channels is a function of time, the router should be able to dynamically allocate channel bandwidth to the active virtual channels.

Figure 12 illustrates the sharing of four virtual channels over a unidirectional physical channel. A dedicated single-bit Request/Acknowledge wire exists between an input virtual channel and an output virtual channel of two adjacent nodes, as shown in Figure 4. The scheduler multiplexes data from the virtual channels over the physical channel. A fair scheduling discipline, such as round-robin, can be used. To preserve bandwidth, only those virtual channels that have a nonempty flit buffer at the sending side and a nonfull flit buffer at the receiving side may participate in the scheduling decision. In other words, among all of the low R/A lines, the scheduler on the sending side will decide which output channel with a nonempty flit buffer can raise its R/A line to high and use the physical channel.

Using a pair of opposite unidirectional channels between two adjacent nodes simplifies control. However, if these two unidirectional channels are not fully utilized, one may be busy while the other is idle. Combining two unidirectional channels into a single bidirectional channel will increase channel utilization. Assuming the bandwidth of each unidirectional channel is B, the bandwidth of the corresponding bidirectional channel will be $2B$. If only one of the nodes has packets to transmit, it can use the full bandwidth. The design of a bidirectional channel must provide a fair and efficient arbitration scheme between two sides. One such arbitration method is based on the concept of token passing,[4] in which a single-bit arbitration line is used to transfer control of the channel between two adjacent nodes.

The virtual channel concept is not without drawbacks. As the number of virtual channels increases, the scheduling becomes more complicated, requiring additional hardware complexity and potentially increasing network latency. The sharing of bandwidth may also increase latency. Consider the following scenario in which a communication path traverses multiple physical channels, each of which supports many virtual channels. If the bandwidth of each physical channel is W and there is no sharing with other virtual channels, the effective bandwidth of the communication path is W. On the other hand, if one of the physical channels along the path is shared with three other packets, that channel becomes a bottleneck and the effective bandwidth of the entire path is reduced to $W/4$, even though the available bandwidth of all other channels in the path is W. The trade-off between increased network throughput and longer communication latency should be considered when deciding whether to use virtual channels.

Open issues

As we have described, wormhole routing algorithms have already been subjected to extensive research. However, we should briefly mention several related topics that have only recently received attention from the research community.

The primary research tools used thus far to study the performance of wormhole routing algorithms have been analysis and simulation, in which either uniform or generic parameterized workloads have been used to evaluate routing algorithms. To account for the characteristics of specific application software, traces of communication in actual parallel programs must be incorporated into such models. More research is needed in this direction before a realistic and practical performance comparison study on the algorithms presented in this article can be conducted. In addition, researchers need simulation programs that efficiently model variations of wormhole routing, including virtual channels, large flit buffers, and sophisticated input and output selection policies.

A major goal in the design of direct networks is to minimize the constituent elements of communication latency so that such systems can support a finer grain of parallelism. Start-up latency, which may include time for memory and buffer coping, can significantly degrade performance. Methods to reduce start-up latency deserve further investigation, although significant progress in this area has been reported recently. The MIT J-machine uses special hardware to achieve a start-up latency of 2 microseconds, and the Ncube-3 is claimed to exhibit a start-up latency of 5 microseconds.

A related issue concerns the number of internal channels connecting the local processor/memory to the router. Most commercial multiprocessors support a single pair of internal channels, which may become a bottleneck for packets entering and leaving the direct network. One system that supports multiple pairs of internal channels is the Intel/CMU iWarp. The appropriate number of internal channels and their cost/performance trade-offs require further study.

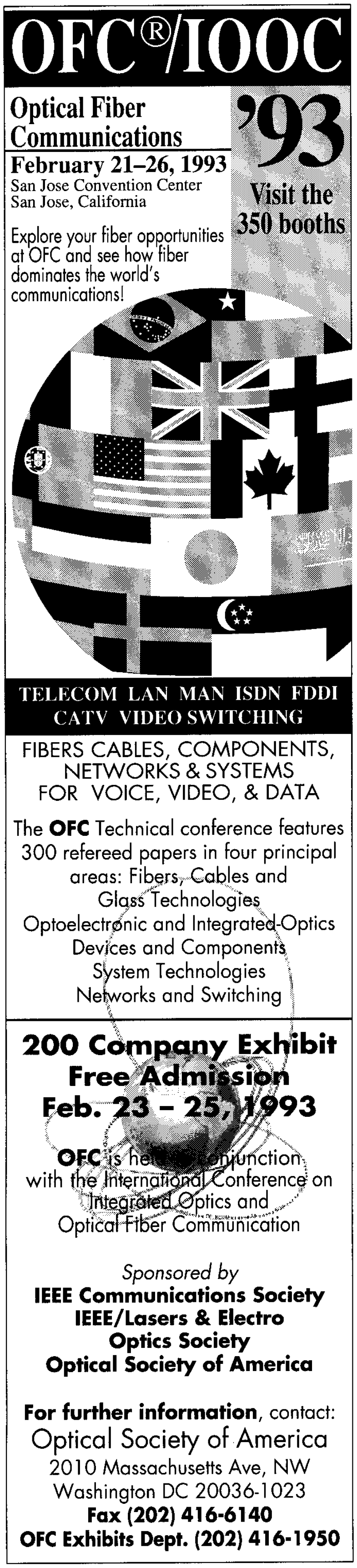
Since the normal behavior of wormhole-routed networks is still a subject of intense research, this article has not addressed the issues of fault tolerance or reliable routing, which are desirable in highly reliable systems. The traditional "replace-and-reboot" approach is used in most existing direct network systems. Investigation of routing and flow control methods for injured wormhole-routed direct networks will likely receive much attention when research in this area becomes more mature and demand increases for highly reliable parallel computing environments.

Finally, this article has surveyed routing algorithms for single-destination, or unicast, communication. Another area of intensive research concerns one-to-many, or multicast, communication. Broadcast is a special case of multicast in which a message is delivered to all nodes in the network. Efficient multicast communication has been shown to be useful in applications such as parallel simulation and parallel search, as well as in operations such as replication and barrier synchronization, found in data parallel languages. Ongoing research concerning multicast communication in wormhole-routed systems includes the study of deadlock-free, hardware-supported multicast routing algorithms[5] and software-based multicast communication.[12] In spite of these efforts, much of the wormhole multicast problem, especially performance evaluation of multicast protocols under actual workloads, remains open to study.

Direct network architectures are strong candidates for use in massively parallel computers, as evidenced by many successful commercial and experimental multicomputers and scalable shared-memory multiprocessors. The characteristics of direct networks, as reflected by the communication latency metric, are critical to the performance of such systems. Wormhole routing, the most promising switching technique, has been adopted in several new massively parallel computers. However, wormhole routing also raises unique technical challenges in routing and flow control — in particular, the development of routing algorithms that avoid deadlock. The problem is complicated by the need for adaptive routing and reconfigurable topologies. We have tried to elucidate such issues while surveying various strategies that have been used or proposed to address them. ■

Acknowledgments

We would like to thank Xiaola Lin and Christopher Glass for their invaluable contributions to this work. Thanks are also due to the anonymous reviewers for their many



insightful comments and suggestions for improvement.

This research was supported in part by National Science Foundation Grants ECS-8814027, CDA-9121641, and MIP-9204066, and by an Ameritech Faculty Fellowship.

References

1. W.C. Athas and C.L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *Computer*, Vol. 21, No. 8, Aug. 1988, pp. 9-25.

2. S. Thakkar et al., "Scalable Shared-Memory Multiprocessor Architectures," *Computer*, Vol. 23, No. 6, June 1990, pp. 71-83.

3. W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *J. Distributed Computing*, Vol. 1, No. 3, 1986, pp. 187-196.

4. W.J. Dally and P. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller," *Proc. Int'l Conf. Computer Design*, IEEE CS Press, Los Alamitos, Calif., Order No. 2473, 1987, pp. 230-234.

5. X. Lin and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks," *Proc. 18th Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., Order No. 2146, 1991, pp. 116-125.

6. W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, Vol. C-36, No. 5, May 1987, pp. 547-553.

7. D.H. Linder and J.C. Harden, "An Adaptive and Fault-Tolerant Wormhole Routing Strategy for *k*-ary *n*-cubes," *IEEE Trans. Computers*, Vol. 40, No. 1, Jan. 1991, pp. 2-12.

8. W.J. Dally and H. Aoki, "Adaptive Routing Using Virtual Channels," tech. report, MIT Laboratory for Computer Science, Sept. 1990. To appear in *IEEE Trans. Parallel and Distributed Systems.*

9. C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *Proc. 19th Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., Order No. 2940, 1992, pp. 278-287.

10. S. Konstantinidou, "Adaptive, Minimal Routing in Hypercubes," *Proc. Sixth MIT Conf. Advanced Research in VLSI*, MIT Press, Cambridge, Mass., 1990, pp. 139-153.

11. W.J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel and Distributed Systems*, Vol. 3, No. 2, Mar. 1992, pp. 194-205.

12. P.K. McKinley et al., "Unicast-Based Multicast Communication in Wormhole-Routed Networks," *Proc. 1992 Int'l Conf. Parallel Processing*, Vol. II, IEEE CS Press, Los Alamitos, Calif., Order No. 3155, 1992, pp. 10-19.

Lionel M. Ni is a professor in the Department of Computer Science and director of the Advanced Computer Systems Laboratory at Michigan State University. His research interests include computer architecture, parallel processing, and distributed computing.

Ni received a BS in electrical engineering from National Taiwan University in 1973, an MS in electrical and computer engineering from Wayne State University in 1977, and a PhD in electrical engineering from Purdue University in 1980. He is a member of the ACM, the Society for Industrial and Applied Mathematics, and the IEEE Computer Society, and he serves on the editorial board of the *Journal of Parallel and Distributed Computing* and *IEEE Transactions on Computers.*

Philip K. McKinley is an assistant professor in the Department of Computer Science at Michigan State University and was previously a member of technical staff at Bell Laboratories. His research interests include scalable architectures and software, optical communications, and multicast communication for parallel processing and computer networks.

McKinley received a BS in mathematics and computer science from Iowa State University in 1982, an MS in computer science from Purdue University in 1983, and a PhD in computer science from the University of Illinois at Urbana-Champaign in 1989. He is a member of ACM and the IEEE Computer Society.

The authors can be contacted at Michigan State University, Dept. of Computer Science, A714 Wells Hall, East Lansing, MI 48824-1027, e-mail {ni, mckinley}@cps.msu.edu.