Parallel Computer Architecture

A Hardware/Software Approach

David E. Culler Jaswinder Pal Singh

with Anoop Gupta



MORGAN KAUFMANN PUBLISHERS, INC.
San Francisco, California

Qualcomm Ex. 1015

Kurt F. Wendt Library
University of Wisconsin - Madison
215 N. Randall Avenue
Madison, WI 53706-1688

Senior Editor	Denise E.M. Penrose
Director of Production and Manufacturing	Yonie Overton
Senior Production Editor	Elisabeth Beller
Editorial Coordinator	Meghan Keeffe
Cover Design	Martin Heirakuji Graphic Design
Cover Photo	Image copyright © 1998 PhotoDisc, Inc.
Text Design	Mark Ong, Side by Side Studios
Copyeditor	Jennifer McClain
Proofreaders	Jennifer McClain, Jeff Van Beuren, Ken DellaPenta, Christine Sabooni
Compositor	Nancy Logan
Illustrators	Nancy Logan, Dartmouth Publishing, Inc., Cherie Plumlee
Indexer	Steve Rath
Printer	Quebecor Printing

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances where Morgan Kaufmann Publishers, Inc. is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Morgan Kaufmann Publishers, Inc.
Editorial and Sales Office
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
USA

Telephone	415/392-2665
Facsimile	415/982-2665
Email	mkp@mkp.com
WWW	http://www.mkp.com

Order toll free 800/745-7323

Advice, Praise, and Errors: Any correspondence related to this publication or intended for the authors should be addressed to the Editorial and Sales Office of Morgan Kaufmann Publishers, Inc., Dept. PCA APE or sent electronically to *pca@mkp.com*. Please report errors by email to *pcabugs@mkp.com*. Please check the errata page at *http://www.mkp.com/pca* to see if the bug has already been reported and fixed.

© 1999 Morgan Kaufmann Publishers, Inc.
All rights reserved
Printed in the United States of America

03 02 01 00 99 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, recording, or otherwise—without the prior written permission of the publisher.

Library of Congress Cataloging-in-Publication Data
Culler, David E.
Parallel computer architecture: a hardware/software approach /
David E. Culler, Jaswinder Pal Singh, with Anoop Gupta.
p. cm.
Includes bibliographical references and index.
ISBN 1-55860-343-3
1. Parallel computers. 2. Computer architecture. I. Singh, Jaswinder Pal. II. Gupta, Anoop. III. Title.
QA76.58.C85 1999
004'.35—dc21

98-28034
CIP

Qualcomm Ex. 1015

10

Interconnection Network Design

We have seen throughout this book that scalable high-performance interconnection networks lie at the core of parallel computer architecture. Our generic parallel machine has three basic components: the processor-memory nodes, the node-to-network interface, and the network that holds it all together. The previous chapters have given a general understanding of the requirements placed on the interconnection network of a parallel machine; this chapter examines in depth the design of high-performance interconnection networks for parallel computers. These networks share basic concepts and terminology with local area networks (LANs) and wide area networks (WANs), which may be familiar to many readers, but the design trade-offs are quite different because of the dramatic difference of time scale.

Parallel computer networks are a rich and interesting topic because they have so many facets, but this richness also makes the topic difficult to understand in an overall sense. For example, parallel computer networks are generally wired together in a regular pattern. The topological structure of these networks has elegant mathematical properties, and there are deep relationships between these topologies and the fundamental communication patterns of important parallel algorithms. However, pseudo-random wiring patterns have a different set of nice mathematical properties and tend to have more uniform performance without really good or really bad communication patterns. There is a wide range of interesting trade-offs to examine at this abstract level, and a huge volume of research papers focus completely on this aspect of network design. On the other hand, passing information between two independent asynchronous devices across an electrical or optical link presents a host of subtle engineering issues. These are the kinds of issues that give rise to major standardization efforts. From yet a third point of view, the interactions between multiple flows of information competing for communications resources have subtle performance effects that are influenced by a host of factors. The performance modeling of networks is another huge area of theoretical and practical research. Real network designs address issues at each of these levels. The goal of this chapter is to provide a holistic understanding of the many facets of parallel computer networks so that the reader may see the diverse network design space within the larger problem of parallel machine design as driven by application demands.

As with all other aspects of design, network design involves understanding trade-offs and making compromises so that the solution is near optimal in a global sense rather than optimized for a particular component of interest. The performance

Qualcomm Ex. 1015

impact of the many interacting facets can be quite subtle. Moreover, no clear consensus exists in the field on the appropriate cost model for networks since trade-offs can be made between very different technologies; for example, bandwidth of the links may be traded against complexity of the switches. It is also very difficult to establish a well-defined workload against which to assess network designs since program requirements are influenced by every other level of the system design before being presented to the network. This is the kind of situation that commonly gives rise to distinct "design camps" and rather heated debates, which often neglect to bring out the differences in base assumptions. In the course of developing the concepts of computer networks, this chapter points out how the choice of cost model and of workload lead to various important design points, which reflect key technological assumptions.

Previous chapters have illuminated the factors that drive network design. The communication-to-computation ratio of the program places a requirement on the data bandwidth the network must deliver if the processors are to sustain a given computational rate. However, this load varies considerably between programs; the flow of information may be physically localized or dispersed, and it may be bursty in time or fairly uniform. In addition, the waiting time of the program is strongly affected by the latency of the network, and the time spent waiting affects the bandwidth requirement. We have seen that different programming model implementations tend to communicate at different granularities (which impacts the size of data transfers seen by the network) and that they use different protocols at the network transaction level to realize the higher-level programming model.

This chapter begins with a set of basic definitions and concepts that underlie all networks. Simple models of communication latency and bandwidth are developed in Section 10.2 to reveal the core differences in network design styles. The key components that are assembled to form networks are described concretely in Section 10.3. Section 10.4 explains the rich space of interconnection topologies in a common framework, and Section 10.5 ties the design trade-offs back to cost, latency, and bandwidth under basic workload assumptions. Section 10.6 explains the various ways that messages are routed within the topology of the network in a manner that avoids deadlock and describes the further impact of routing on communication performance. Section 10.7 dives deeper into the hardware organization of the switches that form the basic building block of networks in order to provide a more precise understanding of the engineering trade-offs of various options and the mechanics underlying the more abstract network concepts. Then Section 10.8 explores the alternative approaches to flow control within a network. With this grounding in place, Section 10.9 brings together the entire range of issues in a collection of case studies and examines the transition of parallel computer network technology into other network regimes, including the emerging system area networks (SANs).

10.1 BASIC DEFINITIONS

The job of an interconnection network in a parallel machine is to transfer information from any source node to any desired destination node, in support of the net-

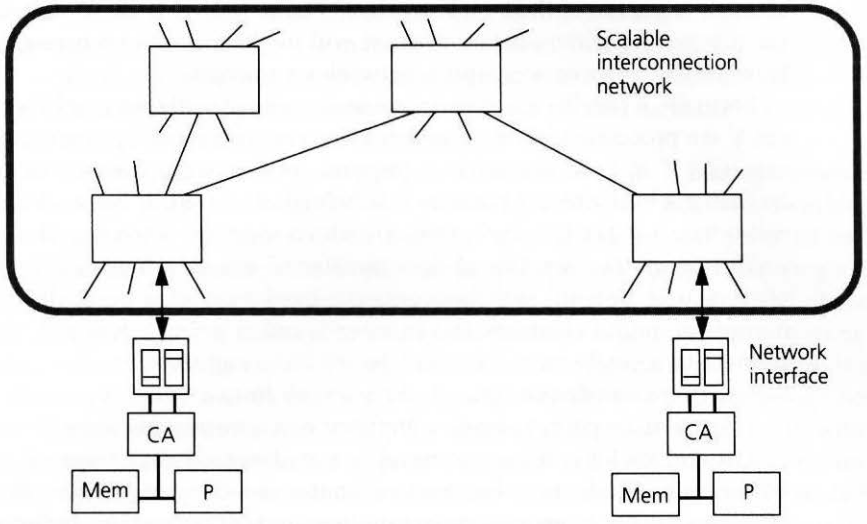


FIGURE 10.1 Generic parallel machine interconnection network. The communication assist initiates network transactions on behalf of the processor or memory controller through a network interface that causes information to be transmitted across a sequence of links and switches to a remote node where the network transaction takes place.

work transactions that are used to realize the programming model. It should accomplish this task with as small a latency as possible, and it should allow a large number of such transfers to take place concurrently. In addition, it should be inexpensive relative to the cost of the rest of the machine.

The expanded diagram for our generic large-scale parallel architecture in Figure 10.1 illustrates the structure of an interconnection network in a parallel machine. The communication assist on the source node initiates network transactions by pushing information through the network interface (NI). These transactions are handled by the communication assist, processor, or memory controller on the destination node, depending on the communication abstraction that is supported.

The network is composed of links and switches that provide a means to route the information from the source node to the destination node. A *link* is essentially a bundle of wires or fibers that carries an analog signal. For information to flow along a link, a transmitter converts digital information at one end into an analog signal that is driven down the link and converted back into digital symbols by the receiver at the other end. The *physical protocol* for converting between streams of digital symbols and an analog signal forms the lowest layer of the network design. The transmitter, link, and receiver collectively form a *channel* for digital information flow between switches (or NIs) attached to the link. The *link-level protocol* segments the stream of symbols crossing a channel into larger logical units, called *packets* or *messages*, that are interpreted by the switches in order to steer each unit arriving on an input channel to the appropriate output channel. Processing nodes communicate

Qualcomm Ex. 1015

across a sequence of links and switches. The *node-level protocol* embeds commands for the remote communication assist within the packets or messages exchanged between the nodes to accomplish network transactions.

Formally, a parallel machine interconnection network is a graph, where the vertices V are processing hosts or switch elements connected by communication channels $C \subseteq V \times V$. A *channel* is a physical link between host or switch elements, including a buffer to hold data as it is being transferred. It has a width w and a signaling rate $f = 1/\tau$ (for cycle time τ), which together determine the *channel bandwidth* $b = wf$. The amount of data transferred across a link in a cycle is called a physical unit, or *phit*.[1] Switches connect a fixed number of input channels to a fixed number of output channels; this number is called the switch *degree*. Hosts typically connect to a single switch but can be multiply connected with separate channels. Messages are transferred through the network from a source host node to a destination host along a path, or *route*, comprised of a sequence of channels and switches.

A useful analogy to keep in mind is a roadway system composed of streets and intersections. Each street has a speed limit and a number of lanes, determining its peak bandwidth. It may be either unidirectional (one-way) or bidirectional. Intersections allow travelers to switch among a fixed number of streets. In each trip, a collection of people travel from a source location along a route to a destination. They may use any of many potential routes, many modes of transportation, and many different ways of dealing with traffic encountered en route. A very large number of such trips may be in progress in a city concurrently, and their respective paths may cross or share segments of the route.

A network is characterized by its topology, routing algorithm, switching strategy, and flow control mechanism.

- The *topology* is the physical interconnection structure of the network graph; this may be regular, as with a two-dimensional grid (typical of many metropolitan centers), or it may be irregular. Most parallel machines employ highly regular networks. A distinction is often made between direct and indirect networks; *direct* networks have a host node connected to each switch whereas *indirect* networks have hosts connected only to a specific subset of the switches, which form the edges of the network. Many machines employ mixed strategies, so the more critical distinction is between the two types of nodes: hosts generate and remove traffic whereas switches only move traffic along.
- The *routing algorithm* determines which routes messages may follow through the network graph. The routing algorithm restricts the set of possible paths to a smaller set of legal paths. There are many different routing algorithms, providing different guarantees and offering different performance trade-offs. For example, continuing the traffic analogy, a city might eliminate gridlock by leg-

1. Since many networks operate asynchronously rather than being controlled by a single global clock, the notion of a network "cycle" is not as widely used as in dealing with processors. We could equivalently define the network cycle time as the time to transmit the smallest physical unit of information, a phit. For parallel architectures, it is convenient to think about the processor cycle time and the network cycle time in common terms. Indeed, the two technological regimes are becoming increasingly similar.

Qualcomm Ex. 1015

islating that cars must travel east-west before making a single turn north or south toward their destination rather than being allowed to zigzag across town. We will see that this does indeed eliminate deadlock, but it limits a driver's ability to avoid traffic en route. In a parallel machine, we are only concerned with routes from a host to a host.

- The *switching strategy* determines how the data in a message traverses its route. There are basically two switching strategies. In *circuit switching*, the path from the source to the destination is established and reserved until the message is transferred over the circuit. (This strategy is like reserving a parade route; it is good for moving a lot of people through, but advanced planning is required and it tends to be unpleasant for any traffic that might cross or share a portion of the reserved route, even when the parade is not in sight. It is also the strategy used in phone systems, which establish a circuit through possibly many switches for each call.) The alternative is *packet switching*, in which the message is broken into a sequence of packets. A packet contains routing and sequencing information as well as data. Packets are individually routed from the source to the destination. (The analogy to traveling as small groups in individual cars is obvious.) Packet switching typically allows better utilization of network resources because links and buffers are only occupied while a packet is traversing them.
- The *flow control mechanism* determines when the message, or portions of it, moves along its route. In particular, flow control is necessary whenever two or more messages attempt to use the same network resource (e.g., a channel) at the same time. One of the traffic flows could be stalled in place, shunted into buffers, detoured to an alternate route, or simply discarded. Each of these options places specific requirements on the design of the switch and influences other aspects of the communication subsystem. (Discarding traffic is clearly unacceptable in our traffic analogy.) The minimum unit of information that can be transferred across a link and either accepted or rejected is called a flow control unit, or *flit*. It may be as small as a phit or as large as a packet or message.

To illustrate the difference between a phit and a flit, consider the nCUBE case study in Section 7.1.4. The links are a single bit wide, so a phit is 1 bit. However, a switch accepts incoming messages in chunks of 36 bits (32 bits of data plus 4 parity bits). It only allows the next 36 bits to come in when it has a buffer to hold it, so the flit is 36 bits. In many more recent machines, such as the T3D, the phit and flit are the same.

An important property of a topology is the *diameter* of a network, which is the length of the maximum shortest path between any two nodes. The *routing distance* between a pair of nodes is the number of links traversed en route; this is at least as large as the shortest path between the nodes and may be larger. The *average distance* is simply the average of the routing distance over all pairs of nodes; this is also the expected distance between a random pair of nodes. In a direct network, routes must be provided between every pair of switches, whereas in an indirect network, it is

Qualcomm Ex. 1015

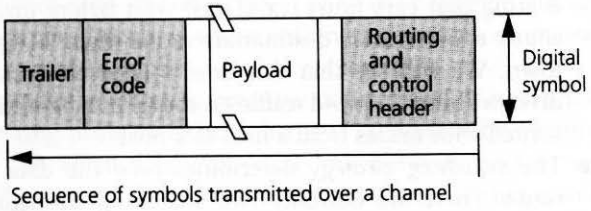


FIGURE 10.2 Typical packet format. A packet forms the logical unit of information that is steered along a route by switches. It is comprised of three parts: a header, a data payload, and a trailer. The header and trailer are interpreted by the switches as the packet progresses along the route, but the payload is not. The node-level protocol is carried in the payload.

only required that routes are provided between hosts. A network is *partitioned* if a set of links or switches are removed such that some hosts are no longer connected by routes.

Most of the discussion in this chapter centers on packets because packet switching is used in most modern parallel machine networks. Where specific important properties of circuit switching arise, they are pointed out. A *packet* is a self-delimiting sequence of digital symbols and logically consists of three parts, illustrated in Figure 10.2: a header, a payload, and a trailer. The *header* is the front of the packet and usually contains the routing and control information so that the switches and network interface can determine what to do with the packet as it arrives. The *payload* is the part of the packet containing data transmitted across the network. The *trailer* is the end of the packet and typically contains the error-checking code so that it can be generated as the message spools out onto the link. The header may also have a separate error-checking code.

The two basic mechanisms for building abstractions in the context of networks are encapsulation and fragmentation. *Encapsulation* involves carrying higher-level protocol information in an uninterpreted form within the message format of a given level. *Fragmentation* involves splitting the higher-level protocol information into a sequence of messages at a given level. Although these basic mechanisms are present in any network, the layers of abstraction in parallel computer networks tend to be much shallower than in, say, the Internet and are designed to fit together very efficiently. To make these notions concrete, observe that the header and trailer of a packet form an envelope that is interpreted by the switches and encapsulates the data payload. Information associated with the node-level protocol is contained within this payload. For example, a read request is typically conveyed to a remote memory controller in a single packet, and the cache line response is a single packet. The memory controllers are not concerned with the actual route followed by the packet or with the format of the header and trailer. At the same time, the network is not concerned with the format of the remote read request within the packet payload. A large bulk data transfer would not typically be carried out as a single packet; instead, it would be fragmented into several packets. Each would need to contain information to indicate where its data should be deposited or which fragment in the

Qualcomm Ex. 1015

overall data sequence it represents. Dropping down a layer, an individual packet is fragmented at the link level into a series of symbols that are transmitted in order across the link, so there is no need for sequencing information. This situation in which higher-level information is carried within an envelope, or multiple such envelopes, that is interpreted by the lower-level protocol occurs at every level (or layer) of network design.

10.2 BASIC COMMUNICATION PERFORMANCE

There is much to understand on each of the four major aspects of network design, but before going into these aspects in detail it is useful to have a general understanding of how they interact to determine the performance and functionality of the overall communication subsystem. Building on the brief discussion of networks in Chapter 7, let us look at performance from the latency and bandwidth perspectives.

10.2.1 Latency

To establish a basic performance model for understanding networks, we may expand the model for the communication time that we have used since Chapter 1. The time to transfer n bytes of information from its source to its destination has four components, as follows.

$$\text{Time}(n)_{S-D} = \text{Overhead} + \text{Routing Delay} + \text{Channel Occupancy} + \text{Contention Delay} \quad (10.1)$$

The *overhead* associated with getting the message into and out of the network on the ends of the actual transmission has been discussed extensively in dealing with the node-to-network interface in previous chapters. We have seen machines that are designed to move cache-line-sized chunks and others that are optimized for large-message DMA transfers. As to the remaining components, the routing delay and channel occupancy are effectively lumped together in previous chapters as the *unloaded latency* of the network for typical message sizes, and contention has been largely ignored. These other components are the focus of this chapter.

The *channel occupancy* provides a convenient lower bound on the communication latency, independent of where the message is going or what else is happening in the network. As we look at network design in more depth, we see below that the occupancy of each link is influenced by the channel width, the signaling rate, and the amount of control information, which is in turn influenced by the topology and routing algorithm. Whereas previous chapters were concerned with the channel occupancy seen from "outside" the network—the time to transfer the message across the bottleneck channel in the route—the view from within the network is that a channel occupancy is associated with each step along the route. The communication assist is occupied for a period of time accepting the communication request from the processor or memory controller and spooling a packet into the network.

Qualcomm Ex. 1015

Each channel the packet crosses en route is occupied for a period of time by the packet, as is the destination communication assist.

For example, an issue that we need to be aware of is the efficiency of the packet encoding. The packet envelope increases the occupancy because header and trailer symbols are added by the source and stripped off by the destination. Thus, for a payload of size n the occupancy of a channel is

$$\frac{n + n_E}{b}$$

where n_E is the size of the envelope and b is the raw bandwidth of the channel. This issue is addressed in the "outside view" by specifying the *effective bandwidth* of the link, derated from the raw bandwidth by

$$\frac{n}{n + n_E}$$

at least for fixed-size packets. However, within the network, packet efficiency remains a design issue. The effect is more pronounced with small packets, but it also depends on how routing is performed.

The *routing delay* is seen from outside the network as the time to move a given symbol, say, the first bit of the message, from the source to the destination. Viewed from within the network, each step along the route incurs a routing delay that accumulates into the delay observed from the outside. The routing delay is a function of the number of channels on the route, called the *routing distance*, h, and the delay Δ incurred at each switch as part of selecting the correct output port. (It is convenient to view the node-to-network interface as contributing to the routing delay like a switch.) The routing distance depends on the network topology, the routing algorithm, and the particular pair of source and destination nodes. The overall delay is strongly affected by switching and routing strategies.

With packet-switched, *store-and-forward* routing, the entire packet is received by a switch before it is forwarded on the next link, as illustrated in Figure 10.3(a). This strategy is used in most wide area networks and was used in several early parallel computers. The unloaded network latency for an n-byte packet, including envelope, with store-and-forward routing is

$$T_{sf}(n, h) = h\left(\frac{n}{b} + \Delta\right) \tag{10.2}$$

where Δ is the additional routing delay per hop.

Equation 10.2 would suggest that the network topology is paramount in determining network latency since the topology fundamentally determines the routing distance, h. In fact, the story is more complicated.

First, consider the switching strategy. With circuit switching, we expect a delay proportional to h to establish the circuit, configure each of the switches along the route, and inform the source that the route is established. After this time, the data should move along the circuit in time n/b plus an additional small delay propor-

Qualcomm Ex. 1015

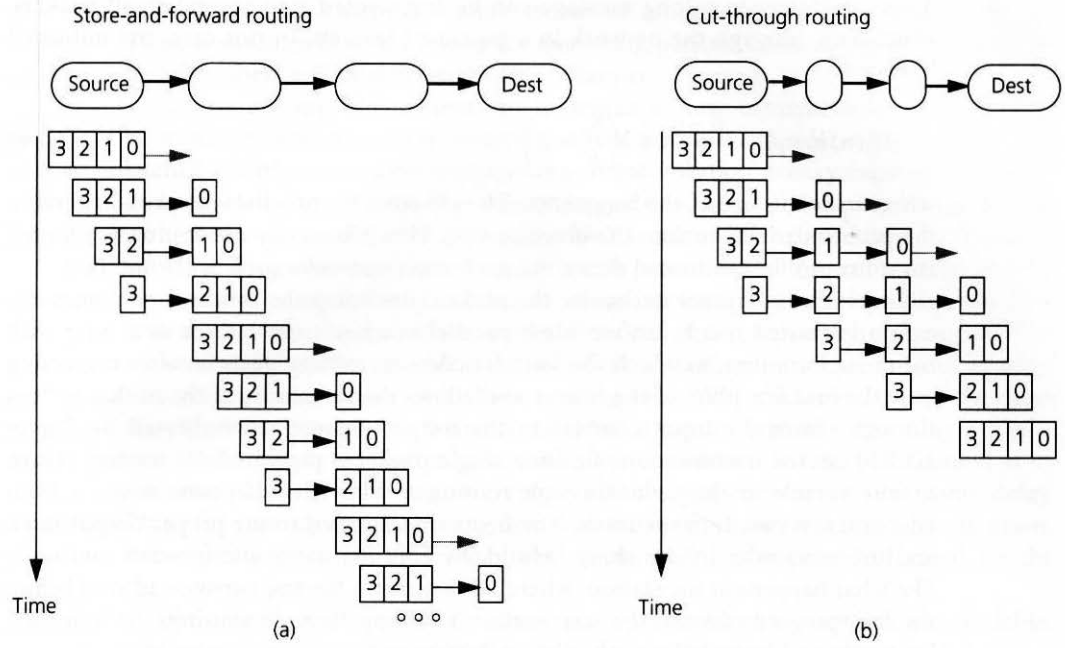


FIGURE 10.3 Store-and-forward versus cut-through routing for packet-switched networks. A four-flit packet traverses three hops from source to destination under store-and-forward and cut-through routing. Cut-through achieves lower latency by making the routing decision (gray arrow) on the first flit and pipelining the packet through a sequence of switches. Store-and-forward accumulates the entire packet before routing it toward the destination.

tional to h. Thus, the unloaded latency in units of the network cycle time, τ, for an n-byte message traveling distance h in a circuit-switched network is

$$T_{cs}(n, h) = \frac{n}{b} + h\Delta \tag{10.3}$$

In Equation 10.3, the setup and routing delay is an additive term, independent of the size of the message. Thus, as the message length increases, the routing distance, and hence the topology, becomes an insignificant fraction of the unloaded communication latency. Circuit switching is traditionally used in telecommunications networks since the call setup is short compared to the duration of the call. It is used in a minority of parallel computer networks, including the Meiko CS-2 and the BBN Butterfly. One important difference is that in parallel machines the circuit is established by routing the message through the network and holding open the route as a circuit. The more traditional approach is to compute the route on the side, configure the switches, and then transmit information on the circuit.

It is also possible to retain packet switching and yet reduce the unloaded latency from that of naive store-and-forward routing. The key concern with Equation 10.2 is that the delay is the product of the routing distance and the occupancy for the full

Qualcomm Ex. 1015

message. However, a long message can be fragmented into several small packets, which flow through the network in a pipelined fashion. In this case, the unloaded latency is

$$T_{sf}(n, h, n_p) = \frac{n - n_p}{b} + h\left(\frac{n_p}{b} + \Delta\right) \tag{10.4}$$

where n_p is the size of the fragments. The effective routing delay is proportional to the packet size rather than the message size. This is basically the approach adopted (in software) for traditional data communication networks such as the Internet.

In parallel computer networks, the idea of pipelining the routing and communication is carried much further. Most parallel machines use packet switching with *cut-through* routing, in which the switch makes its routing decision after inspecting only the first few phits of the header and allows the remainder of the packet to "cut through" from the input channel to the output channel, as indicated by Figure 10.3(b), so the transmission of even a single packet is pipelined. (If we may return to our vehicle analogy, cut-through routing is like what happens when a train encounters a switch in the track. The first car is directed to the proper output track and the remainder follow along behind. By contrast, store-and-forward routing is like what happens at the station, where all the cars in the train arrive and stop before the first proceeds toward the next station.) For cut-through routing, the unloaded latency has a form similar to the circuit switch case

$$T_{ct}(n, h) = \frac{n}{b} + h\Delta \tag{10.5}$$

although the routing coefficient, Δ, may differ since the mechanics of the process are rather different. Observe that with cut-through routing a single message may occupy the entire route from the source to the destination, much like circuit switching. The head of the message establishes the route as it moves toward its destination, and the route clears as the tail moves through.

The preceding discussion of communication latency addresses a message flowing from source to destination without running into traffic along the way. In this unloaded case, the network can be viewed simply as a pipeline, with a start-up cost, pipeline depth, and time per stage. The different switching and routing strategies change the effective structure of the pipeline whereas the topology, link bandwidth, and fragmentation determine the depth and time per stage. Of course, the reason that networks are so interesting is that they are not a simple pipeline but rather an interwoven fabric of portions of many pipelines. The whole motivation for using a network rather than a bus is to allow multiple data transfers to occur simultaneously. This means that one message flow may collide with others and contend for resources. Fundamentally, the network must provide a mechanism for dealing with contention. The behavior under contention depends on several facets of the network design—the topology, the switching strategy, and the routing algorithm—but the bottom line is that at any given time a channel can only be occupied by one message. If two messages attempt to use the same channel at once, one must be deferred. Typ-

ically, each switch provides some means of arbitration for the output channels. Thus, a switch will select one of the incoming packets contending for each output and the others will be deferred in some manner.

An overarching design issue for networks is how contention is handled. This issue recurs throughout the chapter, so let's first just consider what it means for latency. Clearly, contention increases the communication latency experienced by the nodes. Exactly how it increases latency depends on the mechanisms used for dealing with contention within the network, which in turn differ depending on the basic network design strategy. For example, with packet switching, contention may be experienced at each switch. Using store-and-forward routing, if multiple packets that are buffered in the switch need to use the same output channel, one will be selected and the others are blocked in buffers until they are selected. Thus, contention adds queuing delays to the basic routing delay. With circuit switching, the effect of contention arises when trying to establish a circuit; typically, a routing probe is extended toward the destination, and if it encounters a reserved channel it is retracted. The network interface retries establishing the circuit after some delay. Thus, the start-up cost in gaining access to the network increases under contention, but once the circuit is established the transmission proceeds at full speed for the entire message.

With cut-through packet switching, two packet blocking options are available. The *virtual cut-through* approach is to spool the blocked incoming packet into a buffer so that the behavior under contention degrades to that of store-and-forward routing. The *wormhole* approach buffers only a few flits in the switch and leaves the tail of the message in place along the route. The blocked portion of the message is very much like a circuit held open through a portion of the network.

Switches have limited buffering for packets, so under sustained contention the buffers in a switch may fill up. What happens to incoming packets if there is no buffer space to hold them in the switch? In traditional data communication networks, the links are long and little feedback occurs between the two ends of the channel, so the typical approach is to discard the packet. Thus, under contention the network becomes highly unreliable, and sophisticated protocols (e.g., TCP/IP slow-start) are used at the nodes to adapt the requested communication load to what the network can deliver without a high loss rate. Discarding on buffer overrun is also used with most ATM switches, even if they are employed to build closely integrated clusters. Like the wide area case, the source receives no indication that its packet was dropped, so it must rely on some kind of time-out mechanism to deduce that a problem has occurred.

In parallel computer networks, a packet headed for a full buffer is typically blocked in place, rather than discarded; this requires a handshake between the output port and input port across the link, that is, link-level flow control. Under sustained congestion, traffic "backs up" from the point in the network where contention for resources occurs toward the sources that are driving traffic into that point. Eventually, the sources experience back pressure from the network (when it refuses to accept packets), which causes the flow of data into the network to slow

Qualcomm Ex. 1015

down to a rate that can move through the bottleneck.[2] Increasing the amount of buffering within the network allows contention to persist longer without causing back pressure at the source, but it also increases the potential queuing delays within the network when contention does occur.

One of the concerns that arises in networks with message blocking is that the backup can impact traffic that is not headed for the highly contended output. Suppose that the network traffic favors one of the destinations, say, because it holds an important global variable that is widely used. This is often called a "hot spot." If the total amount of traffic destined for that output exceeds the bandwidth of the output, this traffic will back up within the network. If this condition persists, the backlog will propagate backward through the tree of channels directed at this destination, which is called *tree saturation* (Pfister and Norton 1985). Any traffic that crosses the tree will also be delayed. In a wormhole-routed network, an interesting alternative to blocking the message is to discard it but to inform the source of the collision. For example, in the BBN Butterfly machine the source held onto the tail of the message until the head reached the destination (i.e., formed the circuit) and if a collision occurred en route, the worm was retracted all the way back to the source (Rettberg and Thomas 1986). This greatly reduces the impact of tree saturation.

We can see from this brief discussion that all aspects of network design—link bandwidth, topology, switching strategy, routing algorithm, and flow control—combine to determine the latency per message. It should also be clear that a relationship exists between latency and bandwidth. If the communication bandwidth demanded by the program is low compared to the available network bandwidth, collisions will be few, buffers will tend to be empty, and latency will stay low, especially with cut-through routing. As the bandwidth demand increases, latency will increase due to contention.

An important and often overlooked point is that parallel computer networks are effectively a *closed system* with feedback from the network to its traffic sources. The load placed on the network depends on the rate at which processing nodes request communication, which in turn depends on how fast the network delivers this communication. Program performance is affected most strongly by latency when some kind of dependence is involved: the program must wait until a read completes or until a message is received to continue; while it waits, the load placed on the network drops and the latency decreases. This situation is very different from that of file transfers across the country contending with unrelated traffic. In a parallel machine, a program largely contends with itself for communication resources. If the machine is used in a multiprogrammed fashion, parallel programs may also contend with one another, but the request rate of each will be reduced as the service rate of the network is reduced due to the contention. Since low-latency communication is critical

2. This situation is exactly like multiple lanes of traffic converging on a narrow tunnel or bridge. When the traffic flow is less than the flow rate of the tunnel, almost no delay occurs, but when the inbound flow exceeds the tunnel bandwidth, traffic backs up. When the traffic jam fills the available storage capacity of the roadway, the aggregate traffic moves forward slowly enough that the aggregate bandwidth is equal to that of the tunnel.

Qualcomm Ex. 1015

to parallel program performance, the emphasis in this chapter is on cut-through packet-switched networks.

10.2.2 Bandwidth

Network bandwidth is critical to parallel program performance, in part because higher bandwidth decreases occupancy, in part because higher bandwidth reduces the likelihood of contention, and in part because phases of a program may push a large volume of data around without waiting for transmission of individual data items to be completed. Since networks behave like pipelines, it is possible to deliver high bandwidth even when the latency is large.

It is useful to look at bandwidth from two points of view: the "global" aggregate bandwidth available to all the nodes through the network and the "local" individual bandwidth available to a node. If the total communication volume of a program is M bytes and the aggregate communication bandwidth of the network is B bytes per second, then clearly the communication time is at least M/B seconds. On the other hand, if all of the communication is to or from a single node, this estimate is far too optimistic; the communication time would be determined by the bandwidth through that single node.

Let us look first at the bandwidth available to a single node and see how it may be influenced by network design choices. We have seen that the effective local bandwidth is reduced from the raw link bandwidth by the density of the packet

$$b\frac{n}{n+n_E}$$

Furthermore, if the switch blocks the packet for the routing delay of Δ cycles while it makes its routing decision, then the effective local bandwidth is further derated to

$$b\left(\frac{n}{n+n_E+w\Delta}\right)$$

since $w\Delta$ is the opportunity to transmit data that is lost while the link is blocked. Thus, network design issues such as the packet format and the routing algorithm will influence the bandwidth seen by even a single node. If multiple nodes are communicating at once and contention arises, the perceived local bandwidth will drop further (and the latency will rise). Contention at the endpoints happens in any network if multiple nodes send messages to the same node, but it may occur within the interior of the network as well. The choice of network topology and routing algorithm affects the likelihood of contention within the network.

If many of the nodes are communicating at once, it is useful to focus on the global bandwidth that the network can support rather than only the bandwidth available to each individual node. First, we should sharpen the concept of the aggregate communication bandwidth of a network. The most common notion of aggregate bandwidth is the *bisection bandwidth* of the network, which is the sum of the bandwidths of the minimum set of channels that, if removed, partition the network into two equal unconnected sets of nodes. This is a valuable concept because, if the

Qualcomm Ex. 1015

communication pattern is completely uniform, half of the messages are expected to cross the bisection in each direction. We will see in the following that the bisection bandwidth per node varies dramatically in different network topologies. However, bisection bandwidth is not entirely satisfactory as a metric of aggregate network bandwidth because communication is not necessarily distributed uniformly over the entire machine. If communication is localized rather than uniform, bisection bandwidth will give a pessimistic estimate of communication time. An alternative notion of global bandwidth that caters to localized communication patterns would be the sum of the bandwidth of the links from the nodes to the network. The concern with this notion of global bandwidth is that the internal structure of the network may not support it. Clearly, the available aggregate bandwidth of the network depends on the communication pattern; in particular, it depends on how far the packets travel, so we should look at this relationship more closely.

The total bandwidth of all the channels (or links) in the network is the number of channels, C, times the bandwidth per channel, that is, Cb bytes per second, Cw bits per cycle, or C phits per cycle. If each of N hosts issues a packet every M cycles with an average routing distance of h, then each packet occupies, on average, h channels for $l = n/w$ cycles, and the total load on the network is Nhl/M phits per cycle. The average link utilization is at least

$$\rho = M\frac{C}{Nhl} \tag{10.6}$$

and this obviously must be less than one. One way of looking at this is that the number of links per node, C/N, reflects the communication bandwidth (phits per cycle per node) available, on average, to each node. This bandwidth is consumed in direct proportion to the routing distance and the message size. The number of links per node is a static property of the topology. The average routing distance is determined by the topology, the routing algorithm, the program communication pattern, and the mapping of the program onto the machine. Good communication locality may yield a small h, whereas random communication will travel the average distance and really bad patterns may traverse the full diameter. The message size is determined by the program behavior and the communication abstraction. In general, the aggregate communication requirement in Equation 10.6 says that as the machine is scaled up, the channels per node must scale with the increase in expected latency.

In practice, several factors limit the channel utilization, ρ, well below unity. The load may not be perfectly balanced over all the links. Even if it is balanced, the routing algorithm may prevent all the links from being used for the particular communication pattern employed in the program. And even if all the links are usable and the load is balanced over the duration, stochastic variations in the load and contention for low-level resources may arise. All these factors affect the network's *saturation point*, which represents the total channel bandwidth it can usefully deliver. As illustrated in Figure 10.4, if the bandwidth demand placed on the network by the processors (called the *offered bandwidth*) is moderate, the latency remains low, and the delivered bandwidth increases with the offered bandwidth. However, at some point, demanding more bandwidth only increases the contention for resources and the

Qualcomm Ex. 1015

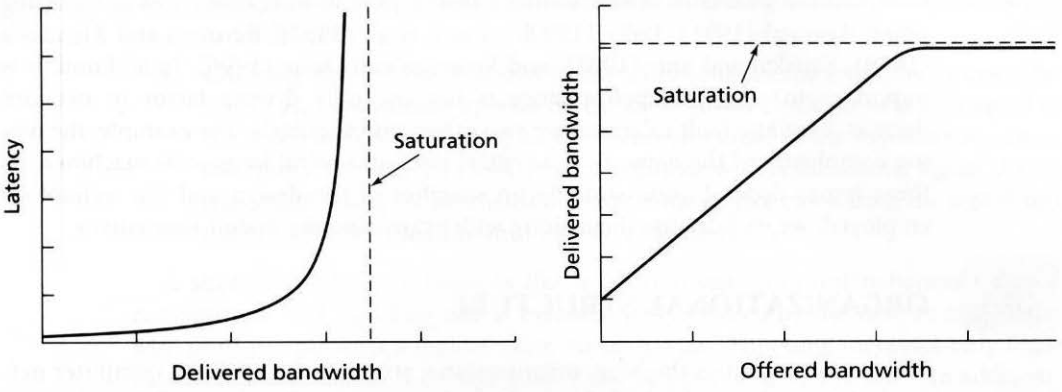


FIGURE 10.4 Typical network saturation behavior. Networks can provide low latency when the requested bandwidth is well below that which can be delivered. In this regime, the delivered bandwidth scales linearly with that requested. However, at some point, the network saturates and additional load causes the latency to increase sharply without yielding additional delivered bandwidth.

latency increases dramatically. The network is essentially moving as much traffic as it can, so additional requests just get queued up in the buffers. Increasing offered bandwidth does not increase what is delivered. We attempt to design parallel machines so that the network stays out of saturation, either by providing ample communication bandwidth or by limiting the demands placed by the processors.

A word of caution is in order regarding the dramatic increase in latency illustrated in Figure 10.4 as the network load approaches saturation. The behavior illustrated in the figure is typical of all queuing systems (and networks) under the assumption that the load placed on the system is independent of the response time. The sources keep pushing messages into the system faster than it can service them, so a queue of arbitrary length builds up somewhere and the latency grows with the length of this queue. In other words, this simple analysis assumes an *open system*, whereas in reality, parallel machines are closed systems. There is only a limited amount of buffering in the network and, usually, only a limited amount of communication buffering in the network interfaces. Thus, if these "queues" fill up, the sources will slow down, reducing their demand to the service rate since there is no place to put the next packet until one is removed. The flow control mechanisms affect this coupling between source and sink. Moreover, dependences within the parallel programs inherently embed some degree of end-to-end flow control because a processor must receive remote information before it can do additional work that depends on the information and generate additional communication traffic. Nonetheless, it is important to recognize that a shared resource, such as a network link, is not expected to be 100% utilized even in the best of circumstances.

This brief performance modeling of parallel machine networks shows that the latency and bandwidth of real networks depends on all aspects of the network design, which we will examine in some detail in the remainder of the chapter. Performance modeling of networks is itself a rich area with a voluminous literature

base, and the interested reader should consult the following references as a starting point: Agarwal (1991), Dally (1990b), Karol et al. (1987), Kermani and Kleinrock (1979), Kruskal and Snir (1983), and Peterson and Davie (1996). In addition, it is important to note that performance is not the only driving factor in network designs. Cost and fault tolerance are two other critical criteria. For example, the wiring complexity of the network is a critical issue in several large-scale machines. As these issues depend quite strongly on specifics of the design and the technology employed, we will discuss them along with examining the design alternatives.

10.3 ORGANIZATIONAL STRUCTURE

This section outlines the basic organizational structure of a parallel computer network. It is useful to think of this issue in the more familiar terms of the processor organization and the applicable engineering constraints. We normally think of the processor as being composed of datapath, control logic, and memory interface, including perhaps the on-chip portions of the memory hierarchy. The datapath is further broken down into ALU, register file, pipeline latches, and so forth. The control logic is built up from examining the data transfers that take place in the datapath. Local connections within the datapath are short and scale well with improvements in VLSI technology whereas control wires and buses are long and become slower relative to gates as chip density increases. A very similar notion of decomposition and assembly applies to the network. Scalable interconnection networks are composed of three basic components: links, switches, and network interfaces. A basic understanding of these components, their performance characteristics, and their inherent costs is essential for evaluating network design alternatives. The set of operations the components perform is quite limited, fundamentally moving packets toward their intended destination.

10.3.1 Links

A link is a cable of one or more electrical wires or optical fibers with a connector at each end that is attached to a switch or network interface port. It allows an analog signal to be transmitted from one end, received at the other, and sampled to obtain the original digital information stream. In practice, there is tremendous variation in the electrical and physical engineering of links; however, their essential logical properties can be characterized along three independent dimensions: length, width, and clocking.

1. A *short link* is one in which only a single logical value can be on the link at any time; a *long link* is viewed as a transmission line where a series of logical values propagate along the link simultaneously at a fraction of the speed of light (1–2 feet per ns, depending on the specific medium).
2. A *narrow link* is one in which data, control, and timing information are multiplexed onto each wire, such as on a single serial link; a *wide link* is one that can simultaneously transmit data and control information. In either case, net-

Qualcomm Ex. 1015

work links are typically narrower than to internal processor datapaths, say, 4 to 16 data bits.

3. Clocking may be synchronous or asynchronous. In the *synchronous* case, the source and destination operate on the same global clock, so data is sampled at the receiving end according to the common clock; in the *asynchronous* case, the source encodes its clock in some manner within the analog signal that is transmitted, and the destination recovers the source clock from the signal and transfers the information into its own clock domain.

A short electrical link behaves like a conventional connection between digital components. The signaling rate is essentially determined by the time to charge the wire until it represents a logical value on both ends. This time increases only logarithmically with length if enough power can be used to drive the link.[3] In addition, the wire must be terminated properly to avoid reflections, which is why it is important that the link be point to point, as opposed to multipoint, like a bus.

The CRAY T3D is a good example of a wide, short, synchronous link design. Each bidirectional link contains 24 bits in each direction: 16 for data, 4 for control, and 4 providing flow control for the link in the reverse direction so that a switch will not try to deliver flits into a full buffer. The entire machine operates under a single 150-MHz clock. A flit is a single phit of 16 bits. Two of the control bits identify the phit type (00 no info, 01 routing tag, 10 packet, 11 end of packet).

In a long wire or optical fiber, the signal propagates along the link from source to destination. For a long link, the delay is clearly linear in the length of the wire. The signaling rate is determined by the time to correctly sample the signal at the receiver, so the length is limited by the signal decay along the link. If more than one wire is in the link, the signaling rate and wire length are also limited by the signal skew across the wires.

A correctly sampled analog signal can be viewed as a stream of digital symbols (phits) delivered from source to destination over time. Logical values on each wire may be conveyed by voltage levels or voltage transitions. Typically, the encoding of digital symbols is chosen so that it is easy to identify common failures (such as stuck-at faults and open connections) and easy to maintain clocking. Within the stream of symbols, individual packets must be identified. Thus, part of the signaling convention of a link is its *framing*, which identifies the start and end of each packet. In a wide link, distinct control lines may identify the head and tail phits. For example, a packet line goes high with the first header phit and stays high until the last tail phit. In the T3D, the routing tag phit and end-of-packet phit provide packet framing. In a narrow link, special control symbols are inserted in the stream to provide framing. In an asynchronous serial link, the clock must be extracted from the

3. The RC delay of a wire increases with the square of the length, so for a fixed amount of signal drive, the network cycle time is strongly affected by length. However, if the driver strength is increased using a driver tree, the time to drive the load of a longer wire only increases logarithmically. (If τ_{inv} is the propagation delay of a basic gate, then the effective propagation delay of a short wire of length l grows as $t_s = K\tau_{inv}\log l$.)

incoming analog signal as well; this is typically done with a unique synchronization burst in the sequence of binary values (Peterson and Davie 1996).

The CRAY T3E network provides a convenient contrast to the T3D. It uses a long, wide, asynchronous link design. The link is 14 bits wide in each direction, operating at 375 MHz. Each "bit" is conveyed by a low-voltage differential signal (LVDS) with a nominal swing of 600 mV on a pair of wires; that is, the receiver senses the difference in the two wires rather than the voltage relative to ground. The clock is sent along with the data. The maximum transmission distance is approximately 1 meter, but even at this length multiple bits will be on the wire at a time. A flit contains five phits, so the switches operate at 75 MHz on 70-bit quantities containing one 64-bit word plus control information. Flow control information is carried on data packets and idle symbols over the link in the reverse direction. The sequence of flits is framed into single-word and eight-word read and write request packets, message packets, and other special packets. The maximum data bandwidth of a link is 500 MB/s.

In general, the encoding of the packet within the frame is interpreted by the nodes attached to the link. Typically, the envelope is interpreted by the switch to do routing and error checking. The payload is delivered uninterpreted to the destination host, at which point further layers or internal envelopes are interpreted and peeled away. However, the destination node may need to inform the source whether it was able to hold the data. This requires some kind of node-to-node information that is distinct from the actual communication, for example, an acknowledgment in the reverse direction. With wide links, control lines may run in both directions to provide this information. Narrow links are almost always bidirectional so that special flow control signals can be inserted into the stream in the reverse direction as in the T3E.[4]

The Scalable Coherent Interface (SCI) defines both a long, wide copper link and a long, narrow fiber link. The links are unidirectional and nodes are always organized into rings. The copper link comprises 18 pairs of wires using differential signaling on both edges of a 250-MHz clock. It carries 16 bits of data, the clock, and a flag bit. The fiber link is serial and operates at 1.25 Gb/s. Packets are a sequence of 16-bit phits, with the header consisting of a destination node number phit and a command phit. The trailer consists of a 32-bit CRC (cyclic redundancy check) word. The flag bit provides packet framing by distinguishing idle symbols from packet phits. At least one idle phit occurs between successive packets.

Many evaluations of networks treat links as having a fixed cost. Common sense would suggest that the cost increases with the length of the link and its width. This is actually a point of considerable debate within the field because the relative quality of different networks depends on the cost model that is used in the evaluation. Much of the cost is in the connectors and the labor involved in attaching them, so the fixed cost is substantial. The connector cost increases with width whereas the wire cost increases with width and length. In many cases, the key constraint is the

4. This view of flow control as inherent to the link is quite different from the view in more traditional networking applications, where flow control is realized on top of the link-level protocol by special packets.

Qualcomm Ex. 1015

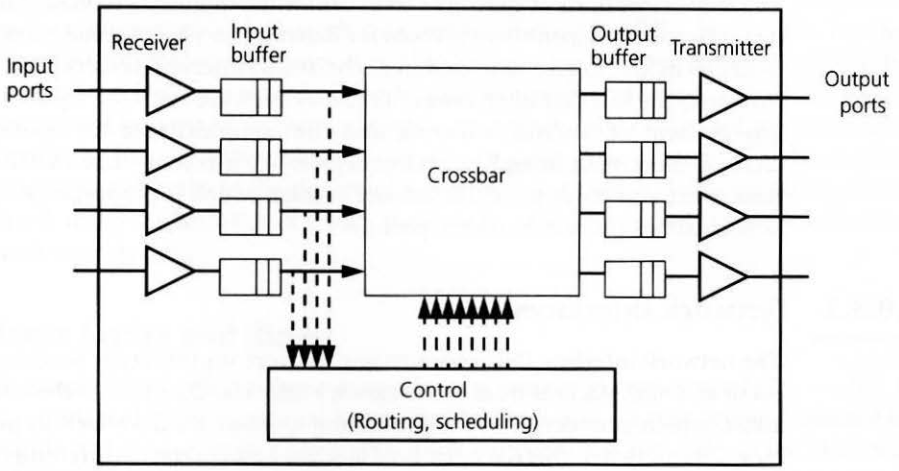


FIGURE 10.5 Basic switch organization. A set of input ports is connected to a set of output ports through a crossbar. The control logic affects the input/output connection at each point in time.

cross-sectional area of a bundle of links, say, at the bisection; this increases with width.

10.3.2 Switches

A switch consists of a set of input ports, a set of output ports, an internal "crossbar" connecting each input to every output, internal buffering, and control logic to effect the input/output connection at each point in time, as illustrated in Figure 10.5. Usually, the number of input ports is equal to the number of output ports, which is called the *degree* of the switch.[5] Each output port includes a transmitter to drive the link. Each input port includes a matching receiver. The input port has a synchronizer in most designs to align the incoming data with the local clock domain of the switch. This is essentially a FIFO, so it is natural to provide some degree of buffering with each input port. There may also be buffering associated with the outputs or shared buffering for the switch as a whole. The complexity of the control logic depends on the routing and scheduling algorithm, as we will discuss. At the very least, it must be possible to determine the output port required by each incoming packet and to arbitrate among input ports that need to connect to the same output port.

5. As with most rules, there are exceptions. For example, in the BBN Monarch design, two distinct kinds of switches were used that had an unequal number of input and output ports (Rettberg et al. 1990). Switches that routed packets to output ports based on routing information in the header could have more outputs than inputs. An alternative device, called a *concentrator*, routed packets to any output port, which were fewer than the number of input ports and all went to the same node.

Qualcomm Ex. 1015

Many evaluations of networks treat the switch degree as its cost. This is clearly a major factor, but again there is room for debate. The cost of some parts of the switch is linear in the degree, for example, the transmitters, receivers, and port buffers. However, the internal interconnect cost may increase with the square of the degree. The amount of internal buffering and the complexity of the routing logic also increase more than linearly with the degree. With recent VLSI switches, the dominant constraint tends to be the number of pins, which is proportional to the number of ports times the width of each port.

10.3.3 Network Interfaces

The network interface (NI) contains one or more input/output ports to source packets to and sink packets from the network under the direction of the communication assist, which connects it to the processing node as we have seen in previous chapters. The network interface, or host nodes, behave quite differently than switch nodes and may be connected via special links. The NI formats the packets and constructs the routing and control information. It may have substantial input and output buffering compared to a switch. It may perform end-to-end error checking and flow control. Clearly, its cost is influenced by its storage capacity, processing complexity, and number of ports.

10.4 INTERCONNECTION TOPOLOGIES

Now that we understand the basic factors determining the performance and the cost of networks, we can examine each of the major dimensions of the design space in relation to these factors. This section covers the set of important interconnection topologies. Each topology is really a class of networks scaling with the number of host nodes N, so we want to understand the key characteristics of each class as a function of N. In practice, the topological properties, such as distance, are not entirely independent of the physical properties, such as length and width, because some topologies fundamentally require longer wires when packed into a physical volume, so it is important to understand both aspects.

10.4.1 Fully Connected Network

A fully connected network is essentially a single switch, which connects all inputs to all outputs. The diameter is 1 link. The degree is N. The loss of the switch wipes out the whole network; however, the loss of a link removes only one node. One such network is simply a bus, and this provides a useful reference point to describe the basic characteristics. It has the nice property that the cost scales as $O(N)$. Unfortunately, only one data transmission can occur on a bus at once, so the total bandwidth is $O(1)$, as is the bisection. In fact, the bandwidth scaling is worse than $O(1)$ because the clock rate of a bus decreases with the number of ports due to RC delays. (An Ethernet is really a bit-serial, distributed bus; it just operates at a low

Qualcomm Ex. 1015

enough frequency that a large number of physical connections are possible.) Another fully connected network is a crossbar. It provides $O(N)$ bandwidth, but the cost of the interconnect is proportional to the number of cross-points, or $O(N^2)$. In either case, a fully connected network is not scalable in practice. This is not to say they are not important. Individual switches are often fully connected internally and provide the basic building block for larger networks. A key metric of technological advance in networks is the degree of a cost-effective switch. With increasing VLSI chip density, the number of nodes that can be fully connected by a cost-effective switch is increasing.

10.4.2 Linear Arrays and Rings

The simplest network is a linear array of nodes numbered consecutively $0, \ldots, N-1$ and connected by bidirectional links. The diameter is $N-1$, the average distance is roughly $2/3\ N$, and removal of a single link partitions the network, so the bisection width is 1 link. Routing in such a network is trivial since there is exactly one route between any pair of nodes. To describe the route from node A to node B, let us define $R = B - A$ to be the *relative address* of B from A. This signed log N bit number is the number of links to cross to get from A to B with the positive direction being away from node 0. Since there is a unique route between a pair of nodes, clearly the network provides no fault tolerance. The network consists of $N-1$ links and can easily be laid out in $O(N)$ space using only short wires. Any contiguous segment of nodes provides a subnetwork of the same topology as the full network.

A *ring* or *torus* of N nodes can be formed by simply connecting the two ends of an array. With unidirectional links, the diameter is $N-1$, the average distance is $N/2$, the bisection width is 1 link, and there is one route between any pair of nodes. The relative address of B from A is $(B-A)$ mod N. With bidirectional links, the diameter is $N/2$, the average distance is $N/3$, the degree of the node is 2, and the bisection is 2. There are two routes (two relative addresses) between pairs of nodes, so the network can function with degraded performance in the presence of a single faulty link. The network is easily laid out with $O(N)$ space using only short wires, as indicated by Figure 10.6, by simply folding the ring. The network can be partitioned into smaller subnetworks; however, the subnetworks are linear arrays rather than rings.

Although these one-dimensional networks are not scalable in any practical sense, they are an important building block conceptually and in practice. The simple routing and low hardware complexity of rings has made them very popular for local area interconnects, including FDDI, FiberChannel Arbitrated Loop, and Scalable Coherent Interface (SCI). Since they can be laid out with very short wires, it is possible to make the links very wide. For example, the KSR1 used a 32-node ring that was 128 bits wide as a building block. SCI obtains its bandwidth by using 16-bit links.

10.4.3 Multidimensional Meshes and Tori

Rings and arrays generalize naturally to higher dimensions, including 2D *grids* and 3D *cubes*, with or without end-around connections. A d-dimensional array consists

Qualcomm Ex. 1015

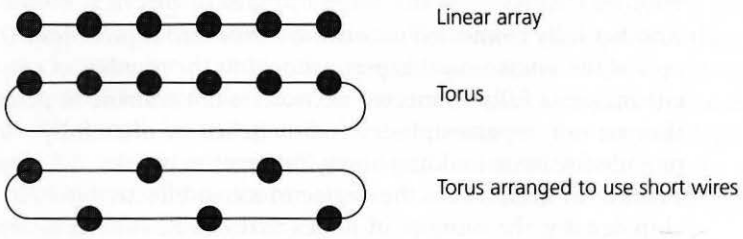


FIGURE 10.6 Linear and ring topologies. The linear array and torus are easily laid out to use uniformly short wires. The distance and cost grow as $O(N)$ whereas the aggregate bandwidth is only $O(1)$.

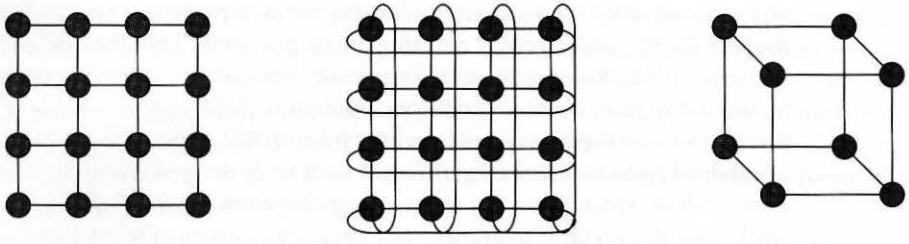


FIGURE 10.7 Grid, torus, and cube topologies. Grids, tori, and cubes are special cases of k-ary d-cube networks, which are constructed with k nodes in each of d dimensions. Low-dimensional networks pack well in physical space with short wires.

of $N = k_{d-1} \times \ldots \times k_0$ nodes, each identified by its d-vector of coordinates $(i_{d-1}, \ldots, i_0)$, where $0 \le i_j \le k_j - 1$ for $0 \le j \le d - 1$. Figure 10.7 shows the common cases of two and three dimensions. For simplicity, we will assume the length along each dimension is the same, so $N = k^d$ ($k = \sqrt[d]{N}$, $r = \log_d N$). This is called a d-dimensional k-ary mesh. (In practice, engineering constraints often result in nonuniform dimensions, but the theory is easily extended to handle that case.) Each node is a switch addressed by a d-vector of radix k coordinates and is connected to the nodes that differ by one in precisely one coordinate. The node degree varies between d and $2d$, inclusive, with nodes in the middle having full degree and the corners having minimal degree.

For a d-dimensional k-ary torus, the edges wrap around from each face, so every node has degree $2d$ (in the bidirectional case) and is connected to nodes differing by one (mod k) in each dimension. We will refer to arrays and tori collectively as meshes. The d-dimensional k-ary unidirectional torus is a very important class of networks, often called a k-ary d-cube, employed widely in modern parallel machines. These networks are usually configured as direct networks, so an additional switch degree is required to support the bidirectional host connection from each switch. They are generally viewed as having low degree, 2 or 3, so the network scales by increasing k along some or all of the dimensions.

Qualcomm Ex. 1015

To define the routes from node A to node B in a d-dimensional array, let $R = (b_{d-1} - a_{d-1}, \ldots, b_0 - a_0)$ be the relative address of B from A. A route must cross $r_i = b_i - a_i$ links in each dimension i, where the sign specifies the appropriate direction. The simplest approach is to traverse the dimensions in order, so for $i = 0 \ldots d - 1$, travel r_i hops in the ith dimension. This corresponds to traveling between two locations in a metropolitan grid by driving in, say, the east-west direction, turning once, and driving in the north-south direction. Of course, we can reach the same destination by first traveling north-south and then east-west or by zigzagging anywhere between these routes. In general, we may view the source and destination points as corners of a subarray and follow any path between the corners that reduces the relative address from the destination at each hop.

The diameter of the network is $d(k - 1)$. The average distance is simply the average distance in each dimension, roughly

$$d\frac{2}{3}k$$

If k is even, the bisection of a d-dimensional k-ary array is k^{d-1} bidirectional links. This is obtained by simply cutting the array in the middle by a (hyper)plane perpendicular to one of the dimensions. (If k is odd, the bisection may be a little bit larger.) For a unidirectional torus, the relative address and routing generalizes along each dimension just as for a ring. All nodes have degree d (plus the host degree), and k^{d-1} links cross the middle in each direction.

It is clear that a two-dimensional mesh can be laid out with $O(N)$ space in a plane with short wires and three-dimensional mesh in $O(N)$ volume in free space. In practice, engineering factors come into play. It is not really practical to build a huge 2D structure, and mechanical issues arise in how 3D is utilized, as illustrated by Example 10.1.

EXAMPLE 10.1 Using a direct 2D mesh topology, such as in the Intel Paragon, where a single cabinet holds 64 processors forming a 4-wide by 16-high array of nodes (each node containing a message processor and one to four compute processors), how might you configure cabinets to construct a large machine with only short wires?

Answer Although there are many possible approaches, the one used by Intel is illustrated in Figure 1.24 of Chapter 1. The cabinets stand on the floor, and large configurations are formed by attaching these cabinets side by side, forming a $16 \times k$ array. The largest configuration was a 1,824-node machine at Sandia National Laboratory configured as a 16×114 array. The bisection bandwidth is determined by the 16 links that cross between cabinets. ■

Other machines have found alternative strategies for dealing with real-world packaging restrictions. The MIT J-machine is a 3D torus where each board comprises an 8×16 torus in the first two dimensions. Larger machines are constructed by stacking these boards next to one another, with board-to-board connections providing the links in the third dimension. The Intel ASCI Red machine with 4,536 compute nodes is constructed as 85 cabinets. It allows long wires to run between

Qualcomm Ex. 1015

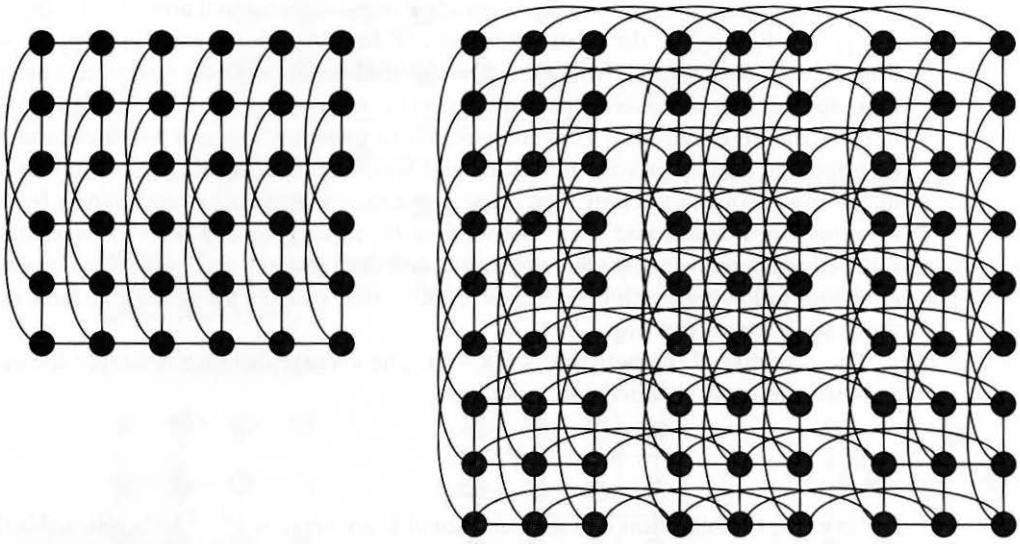


FIGURE 10.8 Embeddings of many logical dimensions in two physical dimensions. A higher-dimensional k-ary d-cube can be laid out in 2D by replicating a 2D slice and then connecting slices across the remaining dimensions. The wiring complexity of these higher-dimensional networks can be easily seen in the figure.

cabinets in each dimension. In general, with higher-dimensional meshes, several logical dimensions are embedded in each physical dimension using longer wires. Figure 10.8 shows a $6 \times 3 \times 2$ array and a four-dimensional 3-ary array embedded in a plane. It is clear that, for a given physical dimension, the average wire length and the number of wires increases with the number of logical dimensions.

10.4.4 Trees

In meshes, the diameter and average distance increases with the dth root of N. Many other topologies exist where the routing distance grows only logarithmically. The simplest of these is a tree. A binary tree has degree 3. Typically, trees are employed as indirect networks with hosts as the leaves, so for N leaves the diameter is $2 \log N$. (Such a topology could be used as a direct network of $N = k \log k$ nodes.) In the indirect case, we may treat the binary address of each node as a $d = \log N$ bit vector specifying a path from the root of the tree—the high-order bit indicates whether the node is below the left or right child of the root and so on down the levels of the tree. The levels of the tree correspond directly to the "dimension" of the network. One way to route from node A to node B would be to go all the way up to the root and then follow the path down specified by the address of B. Of course, we really only need to go up to the first common parent of the two nodes before heading down. Let $R = B \oplus A$, the bitwise xor of the node addresses, be the relative address of A and B

Qualcomm Ex. 1015

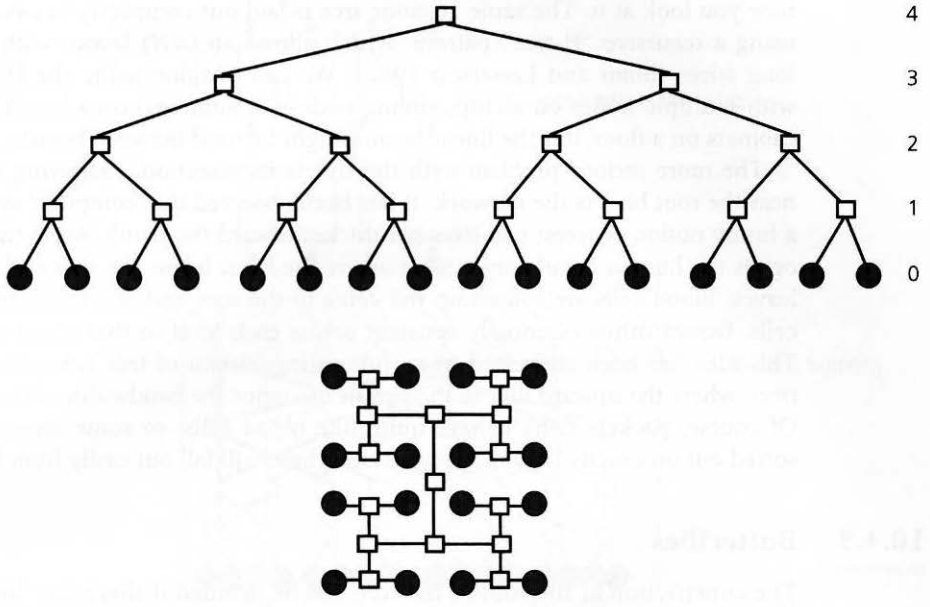


FIGURE 10.9 Binary trees. Trees are a simple network with logarithmic depth that can be laid out efficiently in 2D by using an H-tree configuration. Routing is simple, and they contain trees as subnetworks. However, the bisection bandwidth is only $O(1)$.

and i be the position of the most significant 1 in R. The route from node A to node B is simply $i + 1$ hops up followed by $i + 1$ hops down, with the direction at each branch specified by the low-order $i + 1$ bits of B.

Formally, a complete indirect binary tree is a network of $2N - 1$ nodes organized as $d + 1 = \log_2 N + 1$ levels. Host nodes occupy level 0 and are identified by a d-bit address $A = a_{d-1}, \ldots, a_0$. A switch node is identified by its level i and its $d - i$ bit address $A^{(i)} = a_{d-1}, \ldots, a_i$. A switch node $[i, A^{(i)}]$ is connected to a parent $[i + 1, A^{(i+1)}]$ and two children $[i - 1, A^{(i)}||\ 0]$ and $[i - 1, A^{(i)}||\ 1]$ where the vertical bars indicate bitwise concatenation. There is a unique route between any pair of nodes by going up to the least common ancestor, so no fault tolerance is present. The average distance is almost as large as the diameter and the tree partitions into subtrees. One virtue of the tree is the ease of supporting broadcast or multicast operations from one node to many.

Clearly, by increasing the branching factor of the tree the routing distance is reduced. In a k-ary tree, each node has k children, the height of the tree is $d = \log_k N$, and the address of a host is specified by a d-vector of radix k coordinates describing the path down from the root.

One potential problem with trees is that they seem to require long wires. After all, when we draw a tree in a plane, the lines near the root usually grow exponentially in length with the number of levels, as illustrated in the top portion of Figure 10.9, resulting in an $O(N \log N)$ layout with $O(N)$ long wires. This is really a matter of

Qualcomm Ex. 1015

how you look at it. The same 16-node tree is laid out compactly in two dimensions using a recursive "H-tree" pattern, which allows an $O(N)$ layout with only $O\sqrt{N}$ long wires (Bhatt and Leiserson 1982). We can imagine using the H-tree pattern with multiple nodes on a chip, among nodes (or subtrees) on a board, or between cabinets on a floor, but the linear layout might be used between boards.

The more serious problem with the tree is its bisection. Removing a single link near the root bisects the network. It has been observed that computer scientists have a funny notion of trees; real trees get thicker toward the trunk. An even better analogy is the human circulatory system where the heart forms the root and the cells the leaves. Blood cells are routed up the veins to the root and down the arteries to the cells. Bandwidth is essentially constant across each level so that blood flows evenly. This idea has been addressed in an interesting variant of tree networks, called *fat-trees*, where the upward link to the parent has twice the bandwidth of the child links. Of course, packets don't behave quite like blood cells, so some issues need to be sorted out on exactly how to wire this up. These will fall out easily from butterflies.

10.4.5 Butterflies

The constriction at the root of the tree can be avoided if there are "lots of roots." This is provided by an important logarithmic network called a butterfly. (The butterfly topology arises in many settings in the literature. It is the inherent communication pattern on an element-by-element level of the FFT, the Batcher odd-even merge sort, and other important parallel algorithms. It is isomorphic to topologies in the networking literature, including the Omega and SW-Banyan networks, and is closely related to the shuffle-exchange network and the hypercube, which we will discuss later.) Given 2×2 switches, the basic building block of the butterfly is obtained by simply crossing one of each pair of edges, as illustrated in the top of Figure 10.10. This is a tool for correcting one bit of the relative address—going straight leaves the bit the same, crossing flips the bit. These 2×2 butterflies are composed into a network of $N = 2^d$ nodes in $\log_2 N$ levels of switches by systematically changing the cross edges as shown by the 16-node butterfly illustrated in the bottom portion of Figure 10.10. This configuration shows an indirect network with unidirectional links going upward so that hosts deliver packets into level 0 and receive packets from level d. Each level corrects one additional bit of the relative address. Each node at level d forms the root of a tree with all the hosts as leaves, and from each host is a tree of routes reaching every node at level d.

A d-dimensional indirect butterfly has $N = 2^d$ host nodes and $d2^{d-1}$ switch nodes of degree 2 organized as d levels of $N/2$ nodes each. A switch node at level i, $[i, A]$ has its outputs connected to nodes $[i + 1, A]$ and $[i + 1, A \oplus 2^i]$. To route from A to B, compute the relative address $R = A \oplus B$ and at level i use the "straight edge" if r_i is 0 and the cross edge otherwise. The diameter is $\log N$. In fact, all routes are $\log N$ long. The bisection is $N/2$. (A slightly different formulation with only one host connected to each edge switch has bisection N but twice as many switches at each level and one additional level.)

Qualcomm Ex. 1015

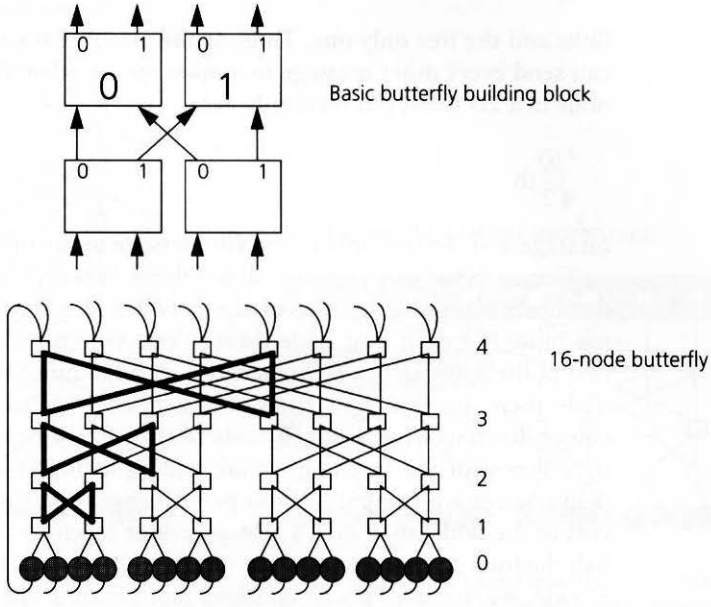


FIGURE 10.10 Butterfly. The butterfly is a logarithmic depth network constructed by composing 2 × 2 blocks that correct one bit in the relative address. It can be viewed as a tree with multiple roots.

A d-dimensional k-ary butterfly is obtained using switches of degree k, a power of two. The address of a node is then viewed as a d-vector of radix k coordinates, so each level corrects $\log k$ bits in the relative address. In this case, there are $\log_k N$ levels. In effect, this fuses adjacent levels into a higher radix butterfly.

There is exactly one route from each host output to each host input, so no inherent fault tolerance is present in the basic topology. However, unlike the 1D mesh or the tree where a broken link partitions the network, there is the potential for fault tolerance in the butterfly. For example, the route from A to B may be broken, but there is a path from A to another node C and from C to B. There are many proposals for making the butterfly fault tolerant by just adding a few extra links. One simple approach is to add an extra level to the butterfly so there are two routes to every destination from every source. This approach was used in the BBN T2000.

The butterfly appears to be a qualitatively more scalable network than meshes and trees because each packet crosses $\log N$ links and there are $N \log N$ links in the network, so on average it should be possible for all the nodes to send messages anywhere all at once. By contrast, a 2D torus or tree has only two links per node, so nodes can only send messages a long distance infrequently and very few nodes are close. A similar argument can be made in terms of bisection. For a random permutation of data among the N nodes, $N/2$ messages are expected to cross the bisection in each direction. The butterfly has $N/2$ links across the bisection whereas the d-dimensional mesh has only

Qualcomm Ex. 1015

$$N^{\frac{d-1}{d}}$$

links and the tree only one. Thus, as the machine scales, a given node in a butterfly can send every other message to a node on the other side of the machine whereas a node in a 2D mesh can send only every

$$\sqrt[d]{\frac{N}{2}}\text{th}$$

message and the tree only every Nth message to the other side.

This analysis has two potential problems, however. The first is cost. In a tree or d-dimensional mesh, the cost of the network is a fixed fraction of the cost of the machine. For each host node there is one switch and d links. In the butterfly, the cost of the network per node increases with the number of nodes since for each host node there are log N switches. Thus, neither scales perfectly. The real question comes down to what a switch costs relative to a processor and what fraction of the overall cost of the machine we are willing to invest in the network to be able to deliver a certain communication performance. If the switch and link is 10% of the cost of the node, then on a 1,024-processor machine the network will be only one-half the total cost with a butterfly. On the other hand, if the switch is equal in cost to a node, we are unlikely to consider more than a low-dimensional network. Conversely, if we reduce the dimension of the network, we may be able to invest more in each switch.

The second problem is that even though the butterfly has enough links to support the bandwidth × distance product of a random permutation, the topology of the butterfly will not allow an arbitrary permutation of N messages among the N nodes to be routed without conflict. A path from an input to an output blocks the paths of many other input/output pairs because there are shared edges. In fact, even when allowed to go through the butterfly twice, permutations exist that cannot be routed without conflicts. However, if two butterflies are laid back to back so that a message goes forward through one and in the reverse direction through the other, then for any permutation there exists a choice of intermediate positions that allows a conflict-free routing of the permutation. This back-to-back butterfly is called a Benes network (Benes 1965; Leighton 1992), and it has been extensively studied because of its elegant theoretical properties. It is often seen as having little practical significance because it is costly to compute the intermediate positions and the permutation has to be known in advance. On the other hand, there is another interesting theoretical result that says that on a butterfly any permutation can be routed with very few conflicts (with high probability) by first sending every message to a random intermediate node and then routing the messages to the desired destination (Leighton 1992). These two results come together in a very nice practical way in the fat-tree network, as follows.

A d-dimensional k-ary fat-tree is formed by taking a d-dimensional k-ary Benes network and folding it back on itself at the high-order dimension, as illustrated in Figure 10.11. The collection of $N/2$ switches at level i is viewed as N^{d-i} "fat nodes"

Qualcomm Ex. 1015

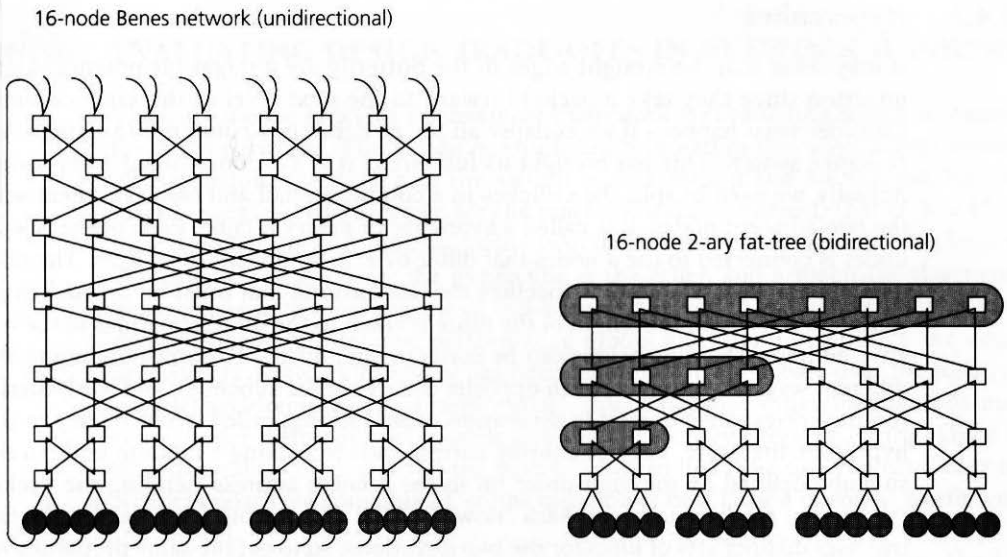


FIGURE 10.11 Benes network and fat-tree. A Benes network is constructed essentially by connecting two butterflies back to back. It has the interesting property that it can route any permutation in a conflict-free fashion, given the opportunity to compute the route off-line. Two forward-going butterflies do not have this property. The fat-tree is obtained by folding the second half of the Benes network back on itself and fusing the two directions so that it is possible to turn around at each level. Collections of switches serve as fat nodes.

of 2^{i-1} switches. The edges of the forward-going butterfly go up the tree toward the roots and the edges of the reverse butterfly go down toward the leaves. To route from *A* to *B*, pick a random node *C* in the least common ancestor fat node of *A* and *B* and take the unique tree route from *A* to *C* and the unique tree route back down from *C* to *B*. Let *i* be the highest dimension of difference in *A* and *B*; then there are 2^i root nodes to choose from, so the longer the routing distance the more the traffic can be distributed. This topology clearly has a great deal of fault tolerance—it has the bisection of the butterfly, the partitioning properties of the tree, and allows essentially all permutations to be routed with very little contention. It is used in the Connection Machine CM-5 and the Meiko CS-2. In the CM-5, the randomization on the upward path is done dynamically by the switches; in the CS-2, the source node chooses the ancestor. A particularly important practical property of butterflies and fat-trees is that the nodes have a fixed degree independent of the size of the network. This allows networks of any size to be constructed with the same switches. As is indicated in Figure 10.11, the physical wiring complexity of the higher levels of a fat-tree or any butterfly-like network becomes critical, since a large number of long wires connect to different places.

Qualcomm Ex. 1015

10.4.6 Hypercubes

It may seem that the straight edges in the butterfly are a target for potential optimization since they take a packet forward to the next level in the same column. Consider what happens if we collapse all the switches in a column into a single log N degree switch. This has brought us full circle; it is a d-dimensional 2-ary torus! Actually, we need to split the switches in a column in half and associate them with the two adjacent nodes. It is called a *hypercube* or *binary n*-cube. Each of the $N = 2^d$ nodes is connected to the d nodes that differ by exactly one bit in address. The relative address $R(A, B) = A \oplus B$ specifies the dimensions that must be crossed to go from A to B. Clearly, the length of the route is equal to the number of ones in the relative address. The dimensions can be corrected in any order (corresponding to the different ways of getting between opposite corners of the subcube), and the butterfly routing corresponds exactly to dimension order routing, called *e-cube routing* in the hypercube literature. Fat-tree routing corresponds to picking a random node in the subcube defined by the high-order bit in the relative address, sending the packet "up" to the random node and back "down" to the destination. Observe that the fat-tree uses distinct sets of links for the two directions, so to get the same properties we need a pair of bidirectional links between nodes in the hypercube.

The hypercube is an important topology that has received tremendous attention in the theoretical literature. For example, lower-dimensional meshes can be embedded one to one in the hypercube by choosing an appropriate labeling of the nodes. Recall from digital design that a graycode sequence orders the numbers from 0 to $2^d - 1$ so that adjacent numbers differ by 1 bit. This shows how to embed a 1D mesh into a d-cube, and it can be extended to any number of dimensions (see Exercise 10.7). Clearly, butterflies, shuffle-exchange networks, and the like embed easily. (Interestingly, a d-cube does not quite embed a $d - 1$ level tree because one extra node is in the d-cube.)

Practically speaking, the hypercube was used by many of the early large-scale parallel machines, including the Cal Tech research prototypes (Seitz 1985), the first three Intel iPSC generations (Ratner 1985), and three generations of nCUBE machines. Later large-scale machines, including the Intel Delta, the Intel Paragon, and the CRAY T3D, use low-dimensional meshes. One of the reasons for the shift is that in practice the hypercube topology forces the designer to use switches of a degree that supports the largest possible configuration. Ports are wasted in smaller configurations. The k-ary d-cube approach provides the practical scalability of allowing arbitrarily sized configurations to be constructed with a given set of components, that is, with switches of fixed degree. Nonetheless, this begs the question, what should the degree be?

The general trend in network design in parallel machines is toward switches that can be wired in an arbitrary topology. We see this, for example, in the IBM SP-2, SGI Origin, Myricom network, and most ATM switches. The designer may choose to adopt a particular regular topology or may wire together configurations of different sizes differently. At any point in time, technology factors such as pin-out and chip area limit the largest potential degree.

Qualcomm Ex. 1015

10.5 EVALUATING DESIGN TRADE-OFFS IN NETWORK TOPOLOGY

The k-ary d-cube provides a convenient framework for evaluating design alternatives for direct networks. The design question can be posed in two ways. Given a choice of dimension, the design of the switch is determined and we can ask how the machine scales. Alternatively, for the machine scale of interest, that is, $N = k^d$, we may ask what the best dimensionality is under salient cost constraints. We have the 2D torus at one extreme, the hypercube at the other, and a spectrum of networks between. As with most aspects of architecture, the key to evaluating trade-offs is to define the cost model and performance model and then to optimize the design accordingly. Network topology has been a point of lively debate over the history of parallel architectures. To a large extent, this is because different positions make sense under different cost models and the technology keeps changing. Once the dimensionality (or degree) of the switch is determined, the space of candidate networks is relatively constrained, so the question is how large a degree is worth working toward.

Let's collect what we know about this class of networks in one place. The total number of switches is N, regardless of degree; however, the switch degree is d, so the total number of links is $C = Nd$ and there are $2wd$ pins per node. The average routing distance is

$$d\left(\frac{k-1}{2}\right),$$

the diameter is $d(k-1)$, and $k^{d-1} = N/k$ links cross the bisection in each direction (for even k). Thus, there are $2Nw/k$ wires crossing the middle of the network.

If our primary concern is the routing distance, then we are inclined to maximize the dimension and build a hypercube. This would be the case with store-and-forward routing, assuming that the degree of the switch and the number of links were not a significant cost factor. In addition, we get to enjoy its elegant mathematical properties. Accordingly, this was the topology of choice for most of the first-generation large-scale parallel machines. However, with cut-through routing and a more realistic hardware cost model, the choice is much less clear. If the number of links or the switch degree is the dominant cost, we are inclined to minimize the dimension and build a mesh. For the evaluation to make sense, we want to compare the performance of design alternatives with roughly equal cost. Different assumptions about what aspects of the system are costly lead to very different conclusions.

The assumed communication pattern influences the decision too. If we look at the worst-case traffic pattern for each network, we will prefer high-dimensional networks where essentially all the paths are short. If we look at patterns where each node is communicating with only one or two near neighbors, we will prefer low-dimensional networks since only a few of the dimensions are actually used.

Qualcomm Ex. 1015

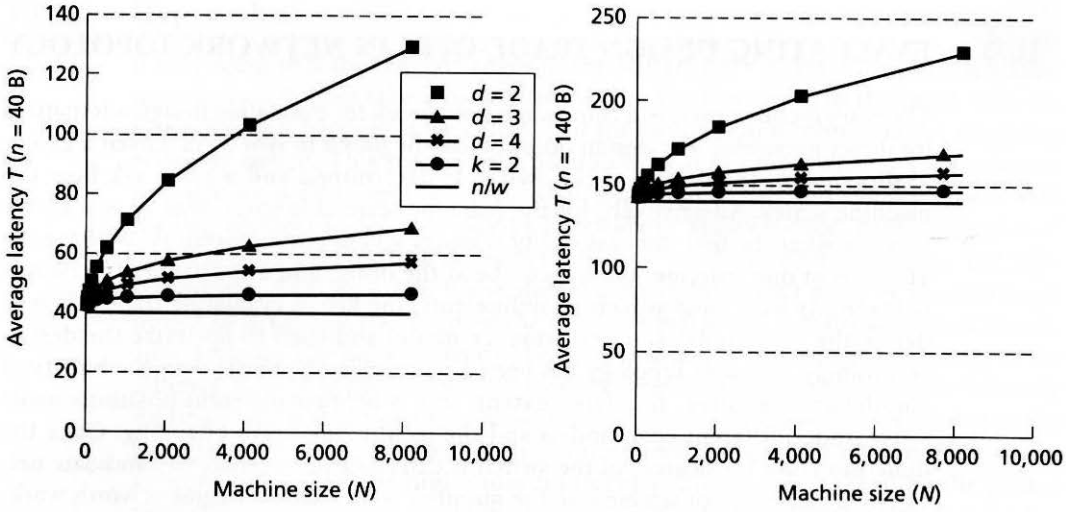


FIGURE 10.12 Unloaded latency scaling of networks for various dimensions with fixed link width. The *n/w* line shows the channel occupancy component of message transmission, that is, the time for the bits to cross a single channel, which is independent of network topology. The curves show the additional latency due to routing.

10.5.1 Unloaded Latency

Figure 10.12 shows the increase in average unloaded latency under our model of cut-through routing for 2-, 3-, and 4-cubes, as well as binary d-cubes ($k = 2$), as the machine size is scaled up. It assumes unit routing delay per stage ($\Delta = 1$) and shows message sizes of 40 and 140 bytes, with $w = 1$ byte. The bottom line shows the portion of the latency resulting from channel occupancy. As we should expect, for smaller messages (or larger routing delay per stage) the scaling of the low-dimension networks is worse because a message experiences more routing steps, on average. However, in making comparisons across the curves in this figure, we are tacitly assuming that the difference in degree is not a significant component of the system cost. In addition, 1 cycle routing is very aggressive; more typical values for high-performance switches are 4–8 network cycles (see Table 10.1). On the other hand, larger message sizes are also common.

To focus our attention on the dimensionality of the network as a design issue, we can fix the cost model and the number of nodes that reflects our design point and examine the performance characteristics of networks with fixed costs for a range of d. Figure 10.13 shows the unloaded latency for short messages as a function of the dimensionality for four machine sizes. For large machines, the routing delays in low-dimensionality networks dominate. For higher dimensionality, the latency approaches the channel time. This "equal number of nodes" cost model has been widely used to support the view that low-dimensional networks do not scale well.

Qualcomm Ex. 1015

Table 10.1 Link Width and Routing Delay for Various Parallel Machine Networks

Machine	Topology	Cycle Time (ns)	Channel Width (bits)	Routing Delay (cycles)	Flit (data bits)
nCUBE/2	Hypercube	25	1	40	32
TMC CM-5	Fat-Tree	25	4	10	4
IBM SP-2	Banyan	25	8	5	16
Intel Paragon	2D Mesh	11.5	16	2	16
Meiko CS-2	Fat-Tree	20	8	7	8
CRAY T3D	3D Torus	6.67	16	2	16
DASH	Torus	30	16	2	16
J-Machine	3D Mesh	31	8	2	8
Monsoon	Butterfly	20	16	2	16
SGI Origin	Hypercube	2.5	20	16	160
Myricom	Arbitrary	6.25	16	50	16

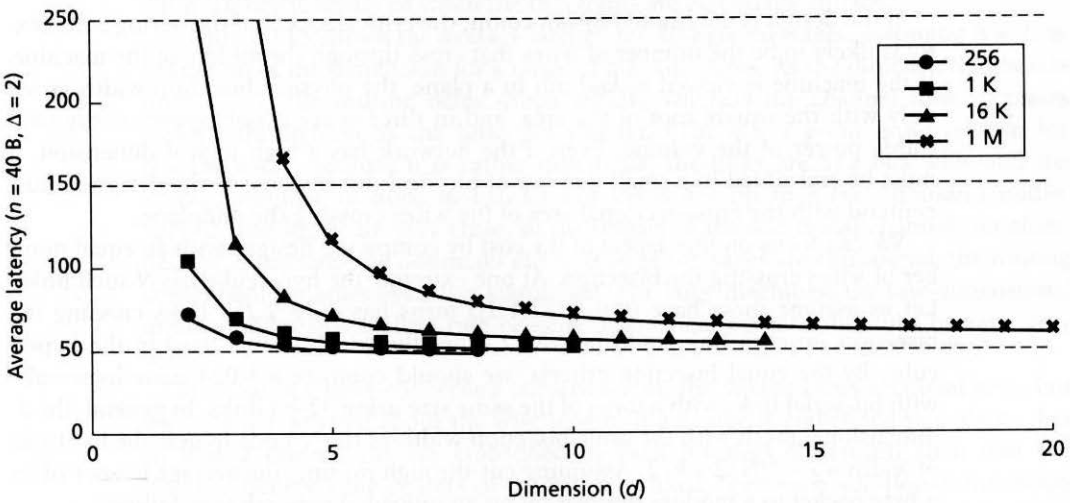


FIGURE 10.13 Unloaded latency for *k*-ary *d*-cubes with equal node count (n = 40 B, Δ = 2) as a function of degree. With the link width and routing delay fixed, the unloaded latency for large networks rises sharply at low dimensions due to routing distance.

Qualcomm Ex. 1015

It is not surprising that higher-dimensional networks are superior under the cost model of Figure 10.13 since the added switch degree, number of channels, and channel length come essentially for free. The high-dimension networks have a much larger number of wires and pins and bigger switches than the low-dimension networks. For the rightmost end of the graph in Figure 10.13, the network design is quite impractical. The cost of the network is a significant fraction of the cost of the large-scale parallel machine, so it makes sense to compare equal cost designs under appropriate technological assumptions. As chip size and density improve, the switch internals tend to become a less significant cost factor whereas pins and wires remain critical. In the extreme, the physical volume of the wires presents a fundamental limit on the amount of interconnection that is possible. So let's compare these networks under assumptions of equal wiring complexity.

One sensible comparison is to keep the total number of wires per node constant, that is, fix the number of pins, $2dw$. Let's take as our baseline a 2-cube with channel width $w = 32$, so there are a total of 128 wires per node. With more dimensions, there are more channels and they must each be thinner. In particular, $w_d = \lfloor 64/d \rfloor$. So in an 8-cube, the links are only 8 bits wide. Assuming 40- and 140-byte messages, a uniform routing delay of 2 cycles per hop, and uniform cycle time, the unloaded latency under equal pin scaling is shown in Figure 10.14. This figure shows a very different story. As a result of narrower channels, the channel time becomes greater with increasing dimension; this mitigates the reduction in routing delay stemming from the smaller routing distance. The very large configurations still experience large routing delays for low dimensions, regardless of the channel width, but all the configurations have an optimum unloaded latency at modest dimension.

If the design is not limited by pin count, the critical aspect of the wiring complexity is likely to be the number of wires that cross through the middle of the machine. If the machine is viewed as laid out in a plane, the physical bisection width grows only with the square root of the area, and in three-space it only grows as the two-thirds power of the volume. Even if the network has a high logical dimension, it must be embedded in a small number of physical dimensions, so the designer must contend with the cross-sectional area of the wires crossing the midplane.

We can focus on this aspect of the cost by comparing designs with an equal number of wires crossing the bisection. At one extreme, the hypercube has N such links. Let us assume these have unit size. A 2D torus has only $2\sqrt{N}$ links crossing the bisection, so each link could be $\sqrt{N}/2$ times the width of that used in the hypercube. By the equal bisection criteria, we should compare a 1,024-node hypercube with bit-serial links with a torus of the same size using 32-bit links. In general, the d-dimensional mesh with the same bisection width as the N-node hypercube has links of width $w_d = \sqrt[d]{N}/2 = k/2$. Assuming cut-through routing, the average latency of an n-byte packet to a random destination on an unloaded network is as follows.

$$T(n, N, d) = \frac{n}{w_d} + \Delta \cdot d\left(\frac{k-1}{2}\right)$$

$$= \frac{n}{k/2} + \Delta \cdot d\left(\frac{k-1}{2}\right) = \frac{n}{\sqrt[d]{N}/2} + \Delta \cdot d\left(\frac{\sqrt[d]{N}-1}{2}\right) \qquad (10.7)$$

Qualcomm Ex. 1015

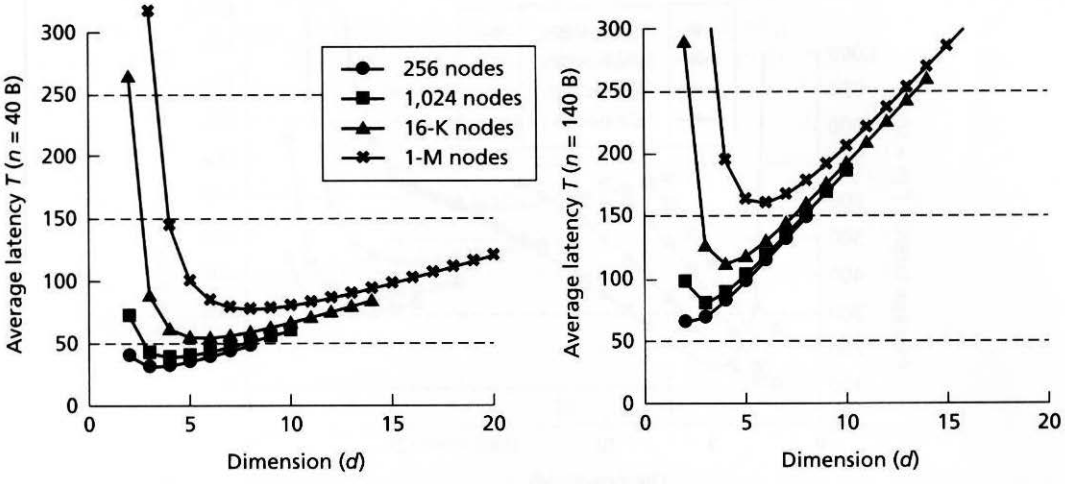


FIGURE 10.14 Unloaded latency for *k*-ary *d*-cubes with equal pin count (n = 40 B and n = 140 B, Δ = 2). With equal pin count, higher dimensions imply narrower channels, so the optimal design point balances the routing delay (which increases with lower dimension) against channel time (which increases with higher dimension).

Thus, increasing the dimension tends to decrease the routing delay but increase the channel time, as with equal pin count scaling. (The reader can verify that the minimum latency is achieved when the two terms are essentially equal.)

Figure 10.15 shows the average latency for 40-byte messages, assuming $\Delta = 2$, as a function of the dimension for a range of machine sizes. As the dimension increases from $d = 2$, the routing delay drops rapidly, whereas the channel time increases steadily throughout as the links get thinner. (The $d = 2$ point is not shown for $N = 1M$ nodes because it is rather ridiculous; the links are 512 bits wide and the average number of hops is 1,023.) For machines up to a few thousand nodes, $\sqrt[3]{N}/2$ and $\log N$ are very close, so the impact of the additional channels on channel width becomes the dominant effect. If large messages are considered, the routing component becomes even less significant. For large machines, the low-dimensional meshes under this scaling rule become impractical because the links become very wide.

Thus far, we have concerned ourselves with the wiring cross-sectional area, but we have not worried about the wire length. If a d-cube is embedded in a plane, that is, if $d/2$ dimensions are embedded in each physical dimension such that the distance between the centers of the nodes is fixed, then each additional dimension increases the length of the longest wire by a $\sqrt{k}$ factor. Thus, the length of the longest wire in a d-cube is $k^{n/2-1}$ times that in the 2-cube. Accounting for increased wire length further strengthens the argument for a modest number of dimensions. This accounting might be done in three ways. If we assume that multiple bits are pipelined on the wire, then the increased length effectively increases the routing delay. If

Qualcomm Ex. 1015

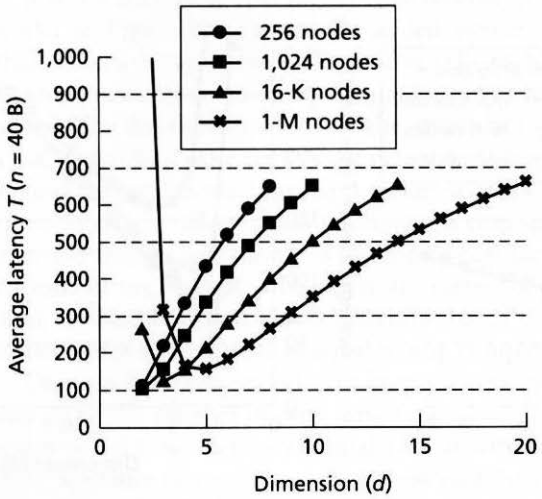


FIGURE 10.15 Unloaded latency for *k*-ary *d*-cubes with equal bisection width (*n* = 40 B, Δ = 2). The balance between routing delay and channel time shifts even more in favor of low-degree networks with an equal bisection width scaling rule.

the wires are not pipelined, then the cycle time of the network is increased as a result of the time to drive the wire, which is logarithmic in the wire length.

The embedding of the *d*-dimensional network into few physical dimensions introduces second-order effects that may enhance the benefit of low dimensions. If a high-dimension network is embedded systematically into a plane, the wire density tends to be highest near the bisection and low near the perimeter. A 2D mesh has uniform wire density throughout, so it makes better use of the area that it occupies.

We should also look at the trade-offs in network design from a bandwidth viewpoint. The key factor influencing latency with equal wire complexity scaling is the increased channel bandwidth at low dimension. Wider channels are beneficial if most of the traffic arises from or is delivered to one or a few nodes. If traffic is localized so that each node communicates with just a few neighbors, only a few of the dimensions are utilized, and again the higher-link bandwidth dominates. If a large number of nodes are communicating throughout the machine, then we need to model the effects of contention on the observed latency and see where the network saturates.

Before leaving the examination of trade-offs for latency in the unloaded case, we should note that the evaluation is rather sensitive to the relative time to cross a wire and to cross a switch. If the routing delay per switch is 20 times that of the wire, the picture is very different, as shown in Figure 10.16. This is the reason for using a higher-dimensionality network in the SGI Origin.

Qualcomm Ex. 1015

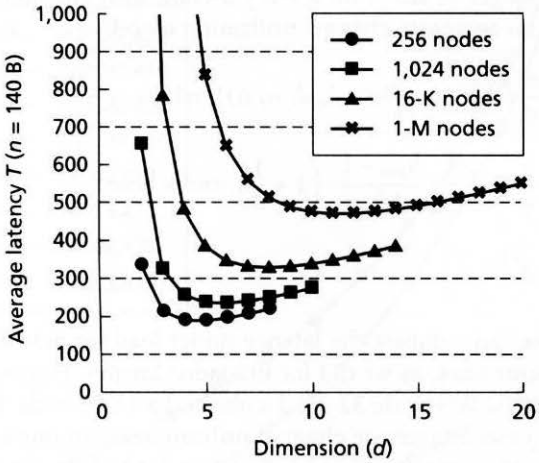


FIGURE 10.16 Unloaded latency for *k*-ary *d*-cubes with equal pin count and larger routing delays (n = 140 B, Δ = 20). When the time to cross a switch is significantly larger than the time to cross a wire, as is common in practice, higher-degree switches become much more attractive.

10.5.2 Latency under Load

In order to analyze the behavior of a network under load, we need to capture the effects of traffic congestion on all the other traffic that is moving through the network. These effects can be subtle and far reaching. Returning to the traffic analogy, notice when you are next driving down a loaded freeway, for example, that where a pair of freeways merge and then split, the traffic congestion is even worse than a series of on-ramps merging into a single freeway. There is far more driver-to-driver interaction in the interchange, and at some level of traffic load the whole thing just seems to stop. Networks behave in a similar fashion, but there are many more interchanges. In order to evaluate these effects in a family of topologies, we must take a position on the traffic pattern, the routing algorithm, the flow control strategy, and a number of detailed aspects of the internal design of the switch. We can then either develop a queuing model for the system or build a simulator for the proposed set of designs. The trick, as in most other aspects of computer design, is to develop models that are simple enough to provide intuition at the appropriate level of design yet accurate enough to offer useful guidance as the design refinement progresses.

We will use a closed-form model of contention delays developed in Agarwal (1991) for random traffic k-ary d-cubes using dimension-order cut-through routing and unbounded internal buffers, so flow control and deadlock issues do not arise. The model predictions correlate well with simulation results for networks meeting the same assumptions. This model is based on earlier work by Kruskal and Snir (1983) modeling the performance of indirect (Banyan) networks. Without going through the derivation, the main result is that we can model the latency for random

Qualcomm Ex. 1015

communication of messages of size n on a k-ary d-cube with channel width w at a load corresponding to an aggregate channel utilization of ρ by

$$T(n, k, d, w, \rho) = \frac{n}{w} + h_{ave}(\Delta + W(n, k, d, w, \rho)), \text{ where}$$

$$W(n, k, d, w, \rho) = \frac{n}{w} \cdot \frac{\rho}{1-\rho} \cdot \frac{h_{ave} - 1}{h_{ave}^2} \cdot \left(1 + \frac{1}{d}\right), \text{ and where} \tag{10.8}$$

$$h_{ave} = d\left(\frac{k-1}{2}\right)$$

Using this model, we can compare the latency under load for networks of low or high dimension of various sizes, as we did for unloaded latency. Figure 10.17 shows the predicted latency on a 1,024-node 32-ary 2-cube and a 1,000-node 10-ary 3-cube as a function of the requested aggregate channel utilization, assuming equal channel width, for relatively small message sizes of 4, 8, 16, and 40 phits. We can see from the right end of the curves that the two networks saturate at roughly the same channel utilization; however, this saturation point decreases rapidly with the message size. The left end of the curves indicates the unloaded latency. The higher-degree switch enjoys a lower base routing delay with the same channel time since there are fewer hops and equal channel widths. As the load increases, this difference becomes less significant. Notice how large the contended latency is compared to the unloaded latency. Clearly, in order to deliver low-latency communication to the user program, it is important that the machine is designed so that the network does not go into saturation easily, either by providing excess network bandwidth or by conditioning the processor load.

The data in Figure 10.17 raises a basic trade-off in network design. How large a packet should the network be designed for? The data shows clearly that networks move small packets more efficiently than large ones. However, smaller packets have worse packet efficiency due to the routing and control information contained in each one and require more network interface events for the same amount of data transfer. For any given technology and detailed design, there is an optimal point.

We must be a bit careful about the conclusions we draw from Figure 10.17 regarding the choice of network dimension. The curves in the figure show how efficiently each network utilizes the set of channels it has available to it. The figure suggests that both use their channels with roughly equal effectiveness. However, the higher-dimensional network has a much greater available bandwidth per node; it has 1.5 times as many channels per node and each message uses fewer channels. In a k-ary d-cube, the available phits per cycle under random communication are

$$\frac{Nd}{d\frac{(k-1)}{2}}$$

or $2/(k-1)$ phits per cycle per node ($2w/k - 1$ bits per cycle). The 3-cube in our example has almost four times as much available bandwidth at the same channel uti-

Qualcomm Ex. 1015

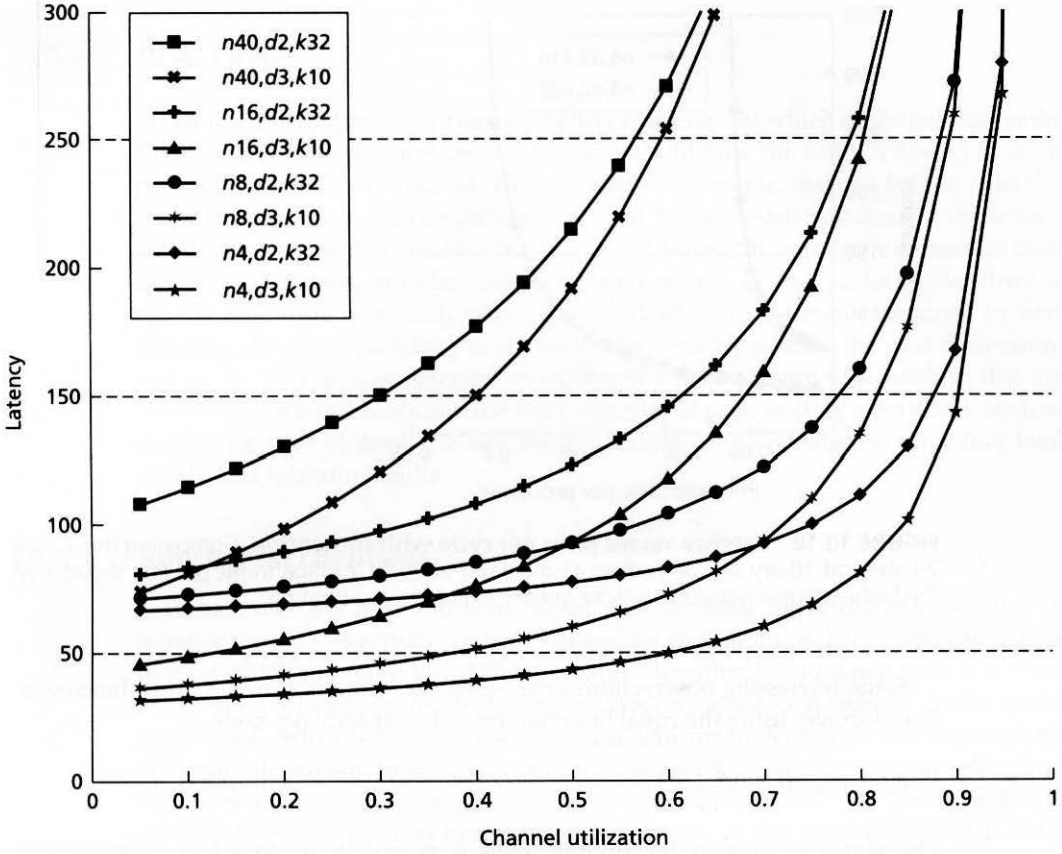


FIGURE 10.17 Latency with contention versus load for 32-ary 2-cube and 10-ary 3-cube with routing delay 2. At low channel utilization, the higher-dimensional network has significantly low latency with equal channel width, but as the utilization increases they converge toward the same saturation point.

lization, assuming equal channel width. Thus, if we look at latency against delivered bandwidth, the picture looks rather different, as shown in Figure 10.18. The 2-cube starts out with a higher base latency and saturates before the 3-cube begins to feel the load.

This comparison brings us back to the question of appropriate equal cost comparison. As an exercise, you can investigate the curves for equal pin-out and equal bisection comparison. Widening the channels shifts the base latency down by reducing channel time, increases the total available bandwidth, and reduces the waiting time at each switch since each packet is serviced faster. Thus, the results are quite sensitive to the cost model.

Qualcomm Ex. 1015

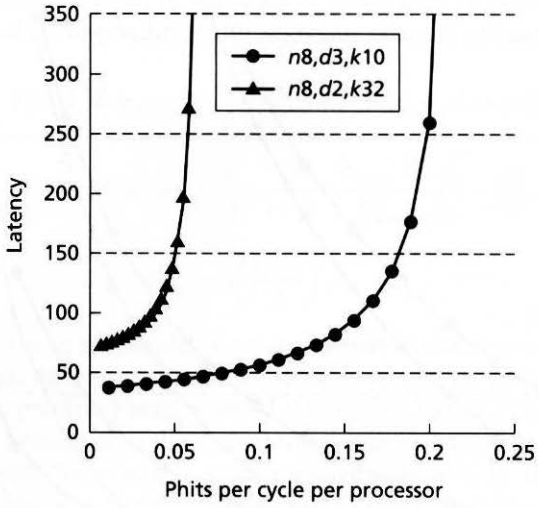


FIGURE 10.18 Latency versus phits per cycle with contention. Comparing the 32-ary 2-cube and 10-ary 3-cube with routing delay 2 at equal average traffic per link shows that the higher-degree networks handle greater load before saturating.

Some interesting observations arise when the width is scaled against dimension. For example, using the equal bisection rule, the capacity per node is

$$C(N,d) = W_d \cdot \frac{2}{k-1} \approx 1$$

The aggregate capacity for random traffic is essentially independent of dimension! Each host can expect to drive, on average, a fraction of a bit per network clock period. This observation yields a new perspective on low-dimension networks. Generally, the concern is that each of several nodes must route messages a considerable distance along one dimension. Thus, each node must send packets infrequently. Under the fixed bisection width assumption, with low dimension the channel becomes a shared resource pool for several nodes whereas high-dimension networks partition the bandwidth resource for traffic in each of several dimensions. When a node uses a channel in the low-dimension case, it uses it for a shorter amount of time. In most systems, pooling results in better utilization than partitioning.

In current machines, the area occupied by a node is much larger than the cross section of the wires, and the scale is generally limited to a few thousand nodes. In this regime, the bisection of the machine is often realized by bundles of cables. This represents a significant engineering challenge, but it is not a fundamental limit on the machine design. The tendency is to use wider links and faster signaling in topologies with short wires, as illustrated by Table 10.1, but it is not as dramatic as the equal bisection scaling rule would suggest.

Qualcomm Ex. 1015

10.6 ROUTING

Recall that the routing algorithm of a network determines which of the possible paths from source to destination are used as routes and how the route followed by each particular packet is determined. We have seen, for example, that in a k-ary d-cube the set of shortest routes is completely described by the relative address of the source and destination, which specifies the number of links that need to be crossed in each dimension. Dimension order routing restricts the set of legal paths so that there is exactly one route from each source to each destination—the one obtained by first traveling the correct distance in the low-order dimension, then the next dimension, and so on. This section describes the different classes of routing algorithms that are used in modern machines and the key properties of good routing algorithms, such as producing a set of deadlock-free routes, maintaining low latency, spreading load evenly, and tolerating faults.

10.6.1 Routing Mechanisms

Let's start with the nuts and bolts. Recall that the basic operation of a switch is to monitor the packets arriving at its inputs and for each input packet to select an output port on which to send it out. Thus, a routing algorithm is a function $R : N \times N \rightarrow C$, which at each switch maps the destination node n_d to the next channel on the route. High-speed switches basically use three mechanisms to determine the output channel from information in the packet header: arithmetic, source-based port select, and table lookup. In parallel computer networks, the switch needs to be able to make the routing decision for all its inputs every few cycles, so the mechanism needs to be simple and fast.

Simple arithmetic operations are sufficient to select the output port in most regular topologies. For example, in a 2D mesh, each packet can carry the signed distance to travel in each dimension $[\Delta x, \Delta y]$ in the packet header. The routing operation at switch ij is given by the following:

Direction	Condition
West $(-x)$	$\Delta x < 0$
East $(+x)$	$\Delta x > 0$
South $(-y)$	$\Delta x = 0, \Delta y < 0$
North $(+y)$	$\Delta x = 0, \Delta y > 0$
Processor	$\Delta x = 0, \Delta y = 0$

To accomplish this kind of routing, the switch needs to test the address in the header and decrement or increment one routing field. Typically, routes in a grid are determined by first moving in the Δx direction and then in the Δy. More generally, in a k-ary d-cube, the routes are determined by moving in each dimension from lowest numbered to highest, called *dimension order* routing. For a binary cube, the switch computes the position of the first bit that differs between the destination and the local node address (or the first nonzero bit if the packet carries the relative address

Qualcomm Ex. 1015

of the destination) and traverses the link in this dimension, called *e-cube* routing. This kind of mechanism is used in Intel and nCUBE hypercubes, the Paragon, the Cal Tech Torus Routing Chip (Seitz and Su 1993), and the J-machine, among others.

A more general approach is *source-based routing*, in which the source builds a header consisting of the output port number for each switch along the route, $p_0, p_1, \ldots, p_{h-1}$. Each switch simply strips off the port number from the front of the message and sends the message out on the specified channel. This allows a very simple switch with little control state and without even arithmetic units to support sophisticated routing functions on arbitrary topologies. All of the intelligence is in the host nodes. It has the disadvantage that the header tends to be large and usually of variable size. If the switch degree is d and routes of length h are permitted, the header may need to carry $h \log d$ routing bits. This approach is used in MIT Parc and Arctic routers, Meiko CS-2, and Myrinet.

A third approach, which is general purpose and allows for a small fixed-size header, is *table-driven routing*, in which each switch contains a routing table R and the packet header contains a routing field i so that the output port is determined by indexing into the table by the routing field, $o = R[i]$. This is used, for example, in HPPI and ATM switches. Generally, the table entry also gives the routing field for the next step in the route, $o, i' = R[i]$, to allow more flexibility in the table configuration. The disadvantage of this approach is that the switch must contain a sizable amount of routing state, and it requires additional switch-specific messages or some other mechanism for establishing the contents of the routing table. Fairly large tables are required to simulate even simple routing algorithms. This approach is better suited to LAN and WAN traffic where only a few of the possible routes among the collection of nodes are used at a time, most of which are long-lasting connections. By contrast, in a parallel machine there is often traffic among all the nodes.

Traditional networking routers contain a full processor that can inspect the incoming message, perform an arbitrary calculation to select the output port, and build a packet containing the message data for that output. This kind of approach is employed in routers and (sometimes) bridges that connect completely different networks (e.g., those that route between Ethernet, FDDI, ATM) or at least different data link layers; it really does not make sense at the time scale of the communication within a high-performance parallel machine.

10.6.2 Deterministic Routing

A routing algorithm is *deterministic* (or *nonadaptive*) if the route taken by a message is determined solely by its source and destination regardless of other traffic in the network. For example, dimension order routing is deterministic; the packet will follow its path regardless of whether a link along the way is blocked. Dimension order and e-cube routing are examples of deterministic algorithms. *Adaptive* routing algorithms allow the route for a packet to be influenced by traffic it meets along the way. For example, in a mesh, the route could zigzag toward its destination if links along the dimension order path were blocked or faulty. In a fat-tree, the upward path toward the common ancestor could steer away from blocked links rather than fol-

 Qualcomm Ex. 1015

lowing a specific path determined when the message was injected into the network. If a routing algorithm only selects shortest paths toward the destination, it is *minimal*; otherwise it is *nonminimal*. Allowing multiple routes between each source and destination is clearly required for adaptation (and fault tolerance), and it also provides a way to spread load over more links. These virtues are enjoyed by source-based and table-driven routing, but the choice is made when the packet is injected. Adaptive routing delays the choice until the packet is actually moving through the network, which clearly makes the switch more complex but has the potential of obtaining better link utilization.

We will concentrate first on deterministic routing algorithms and develop an understanding of some of the most popular routing algorithms and the techniques for proving them deadlock-free before investigating adaptive routing.

10.6.3 Deadlock Freedom

In our discussions of network latency and bandwidth, we have tacitly assumed that messages make forward progress, so it is meaningful to talk about performance. This section shows how to go about proving that a network is deadlock-free. Recall that *deadlock* occurs when a packet waits for an event that cannot occur, for example, when no message can advance toward its destination because the queues of the message system are full and each is waiting for another to make resources available. This can be distinguished from *indefinite postponement*, which occurs when a packet waits for an event that can occur but never does, and from *livelock*, which occurs when the routing of a packet never leads to its destination. Indefinite postponement is primarily a question of fairness, and livelock can only occur with adaptive nonminimal routing. Being free from deadlock is a basic property of well-designed networks that must be addressed from the very beginning.

Deadlock can occur in a variety of situations. A "head-on" deadlock may occur when two nodes attempt to send to each other and each begins sending before either receives. It is clear that if they both attempt to complete sending before receiving, neither will make forward progress. We saw this situation at the user message-passing layer using synchronous send and receive, and we saw it at the node-to-network interface layer. Within the network, it could potentially occur with half-duplex channels or if the switch controller were not able to transmit and receive simultaneously on a bidirectional channel. We should think of the channel as a shared resource that is acquired incrementally, first at the sending end and then at the receiver. In each case, the solution is to ensure that nodes can continue to receive while being unable to send. A reliable network can only be deadlock-free if the nodes are able to remove packets from the network even when they are unable to send packets. (Alternatively, we might recover from the deadlock by eventually detecting a time-out and aborting one or more of the packets, effectively preempting the claim on the shared resource. This does raise the possibility of indefinite postponement, which we will address later.) In this head-on situation there is no routing involved; instead, the problem is due to constraints imposed by the switch design.

Qualcomm Ex. 1015

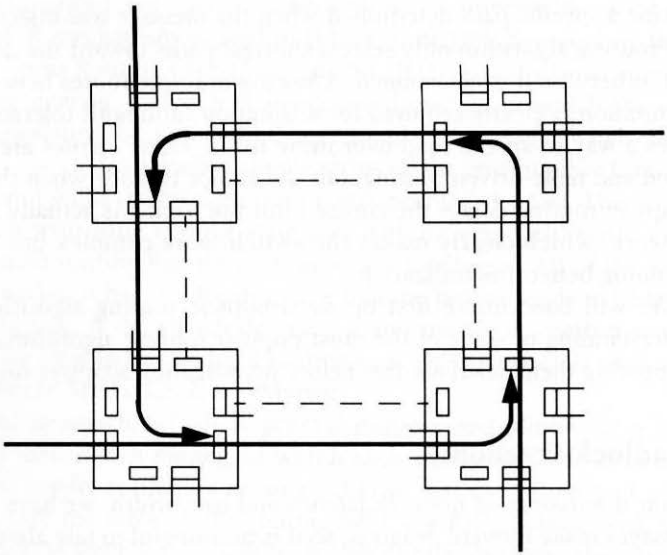


FIGURE 10.19 Examples of network routing deadlock. Each of four switches has four input and output ports. Four packets have each acquired an input port, an output buffer, and an input port, and all are attempting to acquire the next output buffer as they turn left. None will relinquish their output buffer until it moves forward, so none can progress.

A more interesting case of deadlock occurs when multiple messages are competing for resources within the network, as in the routing deadlock illustrated in Figure 10.19. Here we have several messages moving through the network where each message consists of several flits. We should view each channel in the network as having associated with it a certain amount of buffer resources; these may be input buffers at the channel destination, output buffers at the channel source, or both. In our example, each message is attempting to turn to the left, and all of the packet buffers associated with the four channels are full. No message will release a packet buffer until after it has acquired a new packet buffer into which it can move. One could make this example more elaborate by separating the switches with additional switches and channels, but it is clear that the channel resources are allocated incrementally within the network on a distributed basis as a result of messages being routed through, and the resources are nonpreemptible, at least without packet loss. Hence, there is a potential for deadlock.

This routing deadlock can occur with store-and-forward or with cut-through routing, although with cut-through there are greater opportunities for deadlock since each packet stretches over several flit buffers. Only the header flits of a packet carry routing information, so once the head of a message is spooled forward on a channel, all of the remaining flits of the message must spool along on the same channel. Thus, a single packet may hold onto channel resources across several switches. The essential point in these examples is that resources are logically associated with

Qualcomm Ex. 1015

channels and that messages introduce dependences between these resources as they move through the network.

The basic technique for proving a network deadlock-free is to articulate the dependences that can arise between channels as a result of messages moving through the network and to show that there are no cycles in the resulting channel dependence graph; this implies that no traffic patterns can lead to deadlock. The most common way of doing this is to number the channel resources such that each legal route follows a monotonically increasing (or decreasing) sequence; therefore, no dependence cycles can arise. For a butterfly, this is trivial because the network itself is acyclic. It is also simple for trees and fat-trees as long as the upward and downward channels are independent. For networks with cycles in the channel graph, the situation is more interesting.

To illustrate the basic technique for showing a routing algorithm to be deadlock-free, let us show that Δx, Δy routing on a k-ary 2D array is deadlock-free. To prove this, view each bidirectional channel as a pair of unidirectional channels numbered independently. Assign each positive-x channel $\langle i, y\rangle \rightarrow \langle i+1, y\rangle$ the number i, and similarly number the negative-x channels starting from 0 at the most positive edge. Number the positive-y channel $\langle x, j\rangle \rightarrow \langle x, j+1\rangle$ the number $N + j$, and similarly number the negative-y edges from the most positive edge. This numbering is illustrated in Figure 10.20. Any route consisting of a sequence of consecutive edges in one x direction, a 90-degree turn, and a sequence of consecutive edges in one y direction is strictly increasing. The channel dependence graph has a node for every unidirectional link in the network, and there is an edge from node A to node B if it is possible for a packet to traverse channel A and then channel B. All edges in the channel dependence graph go from lower-numbered nodes to higher-numbered ones, so there are no cycles in the channel dependence graph (even though there are many cycles in the network).

This proof easily generalizes to any number of dimensions and, since a binary d-cube consists of a pair of unidirectional channels in each dimension, to show that e-cube routing is deadlock-free on a hypercube. Observe, however, that the proof does not apply to k-ary d-cubes in general because the channel number decreases at the wrap-around edges. Indeed, it is not hard to show that for $k > 4$, dimension order routing will even introduce a dependence cycle on a unidirectional torus ($d = 1$).

Notice that the deadlock-free routing proof applies even if only a single flit of buffering is on each channel and that the potential for deadlock exists in a k-ary d-cube even with multiple packet buffers and store-and-forward routing since a single message may fill up all the packet buffers along its route. However, if the use of channel resources is restricted, it is possible to break the deadlock. For example, consider the case of a unidirectional torus with multiple packet buffers per channel and store-and-forward routing. Suppose that one of the packet buffers associated with each channel is reserved for messages destined for nodes with a larger number than their source, that is, packets that do not use wraparound channels. This means that it will always be possible for positive-going messages to make progress. Although wraparound messages may be postponed, the network does not deadlock. This solution is typical of the family of techniques for making store-and-forward

Qualcomm Ex. 1015

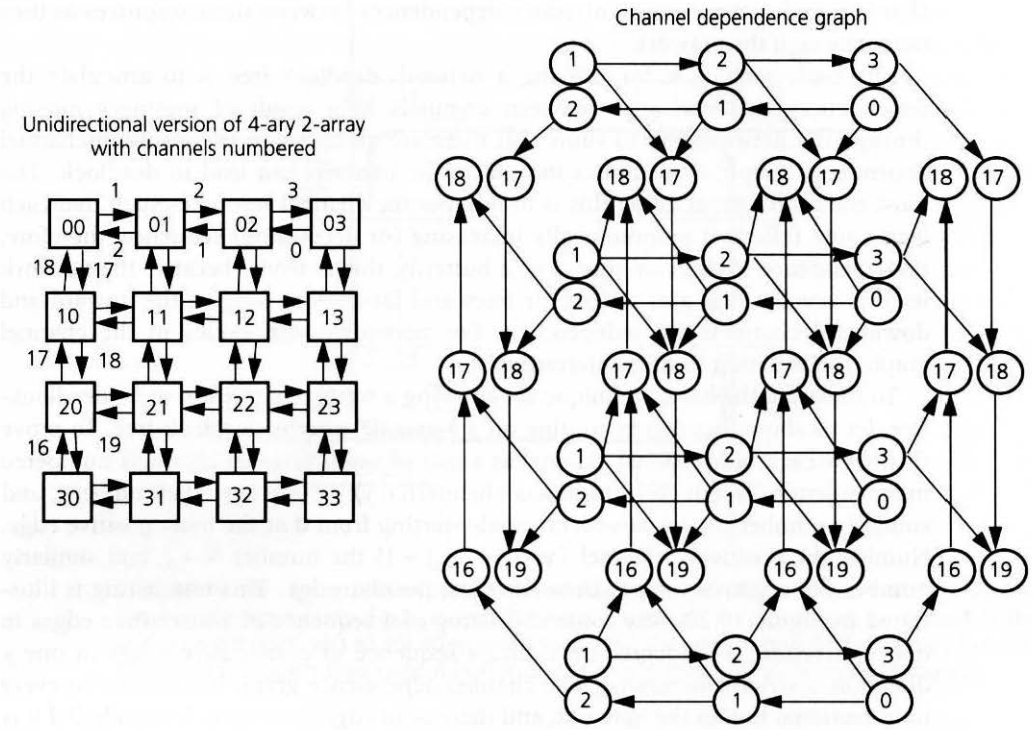


FIGURE 10.20 Channel ordering in the network graph and corresponding channel dependence graph. To show that the routing algorithm is deadlock-free, it is sufficient to demonstrate that the channel dependence graph has no cycles.

packet-switched networks deadlock-free; there is a concept of a *structured buffer pool* (in which certain buffers have specific functions) and the routing algorithm restricts the assignment of buffers to packets to break dependence cycles. This solution is not sufficient for wormhole routing since it tacitly assumes that packets of different messages can be interleaved as they move forward.

Observe that deadlock-free routing does not mean the system is deadlock-free. The network is deadlock-free only as long as it is drained into the NIs, even when the NIs are unable to send. If two-phase protocols are employed, we need to ensure that fetch deadlock is avoided. This means either providing two logically independent networks or ensuring that the two phases are decoupled through NI buffers as discussed in Chapter 7. Of course, the program may still have deadlocks, such as circular waits on locks or head-on collision using synchronous message passing. We have worked our way down from the top, showing how to make each of these layers deadlock-free as long as the next layer below is deadlock-free.

Given a network topology and a set of resources per channel, there are two basic approaches for constructing a deadlock-free routing algorithm: restrict the paths that packets may follow or restrict how resources are allocated. This observation

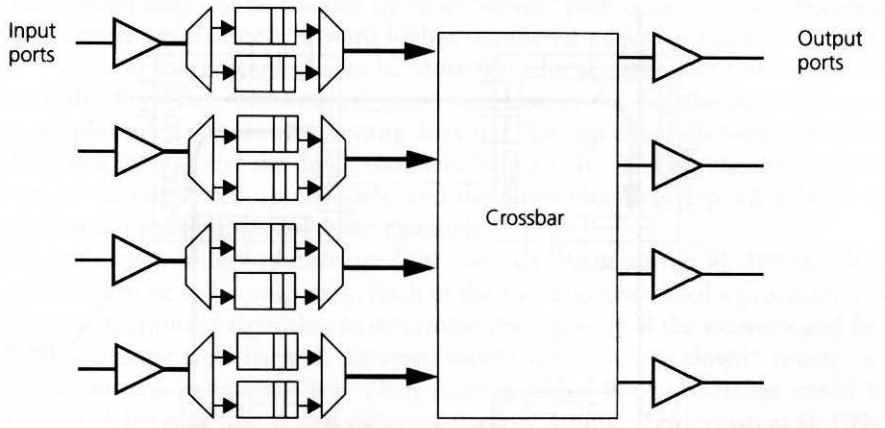


FIGURE 10.21 Multiple virtual channels in a basic switch. Each physical channel is shared by multiple virtual channels. The input ports of the switch split the incoming virtual channels into separate buffers; however, these are multiplexed through the switch to avoid expanding the crossbar.

raises a number of interesting questions. Is there a general technique for producing deadlock-free routes with wormhole routing on an arbitrary topology? Can such routes be adaptive? Is some minimum amount of channel resources required?

10.6.4 Virtual Channels

The basic technique for making networks with wormhole routing deadlock-free is to provide multiple buffers with each physical channel and to split these buffers into a group of *virtual channels*. Going back to our basic cost model for networks, this does not increase the number of links in the network nor the number of switches. In fact, it does not even increase the size of the crossbar internal to each switch since only one flit at a time moves through the switch for each output channel. As indicated in Figure 10.21, it does require additional selectors and multiplexers within the switch to allow the links and the crossbar to be shared among multiple virtual channels per physical channel.

Virtual channels are used to avoid deadlock by systematically breaking cycles in the channel dependence graph. Consider, for example, the four-way routing deadlock cycle of Figure 10.19. Suppose we have two virtual channels per physical channel, and messages at a node numbered higher than their destination are routed on the high channels while messages at a node numbered less than their destinations are routed on the low channels. As illustrated in Figure 10.22, the dependence cycle is broken. Applying this approach to the k-ary d-cube, treat the channel labeling as a radix $d + 1 + k$ number of the form ivx, where i is the dimension, x is the coordinate of the source node of the channel in dimension i, and v is the virtual channel number. In each dimension, if the destination node has a smaller coordinate than the source node in that dimension (i.e., if the message must use a wraparound edge),

Qualcomm Ex. 1015

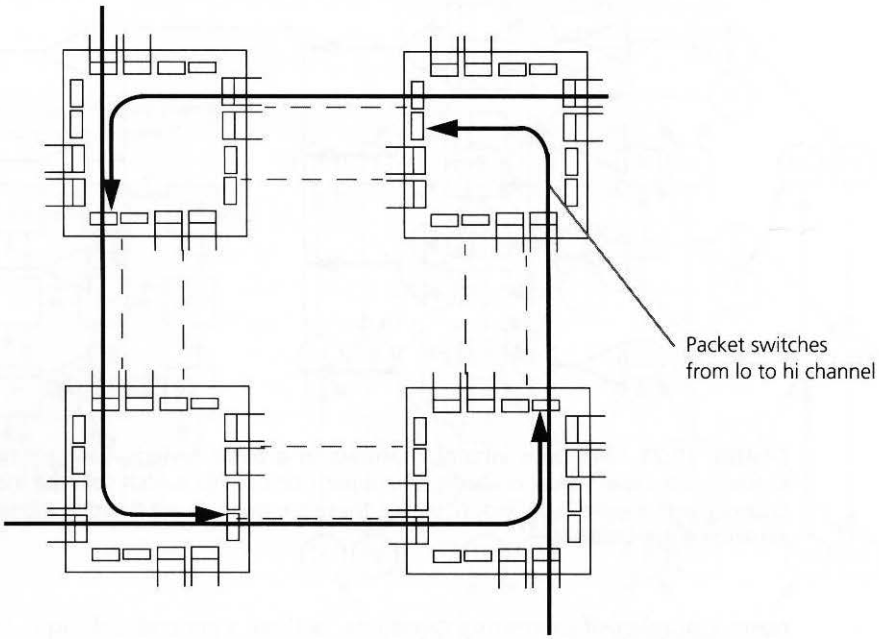


FIGURE 10.22 Breaking deadlock cycles with virtual channels. Each physical channel is broken into two virtual channels; call them lo and hi. The virtual channel "parity" of the input port is used for the output except on turns north to west, which make a transition from lo to hi.

use the $v = 1$ virtual channel in that dimension. Otherwise, use the $v = 0$ channel. You can verify that dimension order routing is deadlock-free with this assignment of virtual channels. Similar techniques can be employed with other popular topologies (Dally and Seitz 1987). Notice that with virtual channels we need to view the routing algorithm as a function $R : C \times N \rightarrow C$ because the virtual channel selected for the output depends on which channel it came in on.

10.6.5 Up*-Down* Routing

Are virtual channels required for deadlock-free wormhole routing on an arbitrary topology? No. If we assume that all the channels are bidirectional, there is a simple algorithm for deriving deadlock-free routes for an arbitrary topology. Not surprisingly, it restricts the set of legal routes. The general strategy is similar to routing in a tree, where routes go up the tree away from the source and then down to the destination. We assume the network consists of a collection of switches, some of which have one or more hosts attached to them. Given the network graph, we want to number the switches so that the numbers increase as we get farther away from the hosts. One approach is to construct a spanning tree of the graph with hosts at the leaves and numbers increasing toward the root. It is clear that for any source host,

any destination host can be reached by an *up*-down** path consisting of a sequence of zero or more up channels (toward higher-numbered nodes), a single turn, and a series of zero or more down channels. Moreover, the set of routes following such paths are deadlock-free. The network graph may have cycles, but the channel dependence graph under up*-down* routing does not. The up channels form a directed acyclic graph (DAG) and the down channels form a DAG. The up channels depend only on lower-numbered up channels, and the down channels depend only on up channels and higher-numbered down channels.

This style of routing was developed for Autonet (Anderson et al. 1992), which was intended to be self-configuring. Each of the switches contained a processor that could run a distributed algorithm to determine the topology of the network and find a unique spanning tree. Each of the hosts would compute up*-down* routes as a restricted shortest-paths problem. Then routing tables in the switches could be established. A breadth-first search variant is used by Atomic (Felderman et al. 1994) and Myrinet (Boden et al. 1995), where the switches are passive and the hosts determine the topology by probing the network. Each host runs an algorithm that partitions the network into levels with host nodes at level zero and each switch at the level corresponding to its maximum distance from a host. The numbering is given by a breadth-first search from the highest numbered switch and the algorithm determines the set of source-based routes from the host to the other nodes. A key challenge in automatic mapping of networks, especially with simple switches that only move messages through without any special processing, is determining when two distinct routes through the network lead to the same switch (Mainwaring et al. 1997). One solution is to try returning to the source by reversing routes from previously known switches; another is detecting when identical paths exist from two supposedly distinct switches to the same host.

10.6.6 Turn-Model Routing

We have seen that a deadlock-free routing algorithm can be constructed by restricting the set of routes within a network or by providing buffering with each channel that is used in a structured fashion. How much do we need to restrict the routes? Is there a minimal set of restrictions or a minimal combination of routing restrictions and buffers? An important development in this direction is turn-model routing (Glass and Ni 1992). Consider, for example, a 2D array. There are eight possible turns, which form two simple cycles, as shown in Figure 10.23. (The figure is illustrating cycles appearing in the network involving multiple messages. There is a corresponding cycle in the channel dependence graph.) Dimension order routing prevents the use of four of the eight turns—when traveling in $\mp x$ it is legal to turn in $\mp y$, but once a packet is traveling in $\mp y$ it can make no further turns. The illegal turns are indicated by gray lines in the figure. Intuitively, it seems possible to prevent cycles by eliminating only one turn in each cycle.

Of the 16 different ways to prohibit two turns in a 2D array, 12 prevent deadlock. These consist of the three unique algorithms shown in Figure 10.24 and rotations of these. The west-first algorithm is so named because no turn is allowed into the $-x$

Qualcomm Ex. 1015

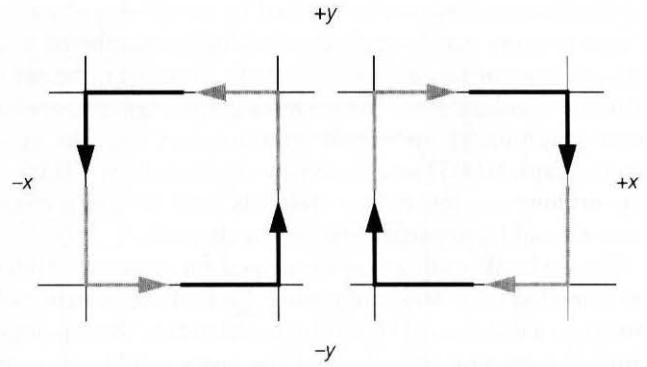


FIGURE 10.23 Turn restrictions of Δx, Δy routing. Dimension order routing on a 2D array prohibits the use of four of the eight possible turns, thereby breaking both of the simple dependence cycles. A deadlock-free routing algorithm can be obtained by prohibiting only two turns.

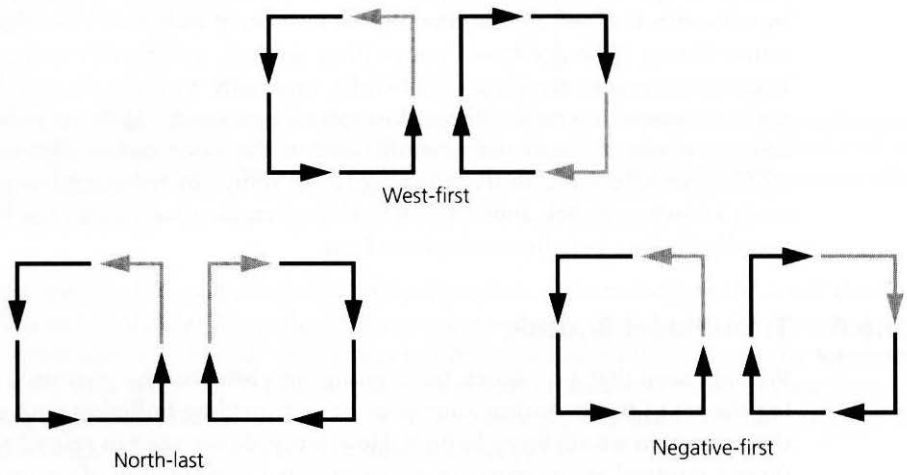


FIGURE 10.24 Minimal turn-model routing in 2D. Only two of the eight possible turns need be prohibited in order to obtain a deadlock-free routing algorithm. Legal turns are shown for three such algorithms.

direction; therefore, if a packet needs to travel in this direction, it must do so before making any turns. Similarly, in north-last there is no way to turn out of the $+y$ direction, so the route must make all its other adjustments before heading in this direction. Finally, negative-first prohibits turns from a positive direction into a negative direction, so the route must go as negative as its needs to before heading in either positive direction.

Each of these turn-model algorithms allows complex, even nonminimal routes. For example, Figure 10.25 shows some of the routes that might be taken under

Qualcomm Ex. 1015

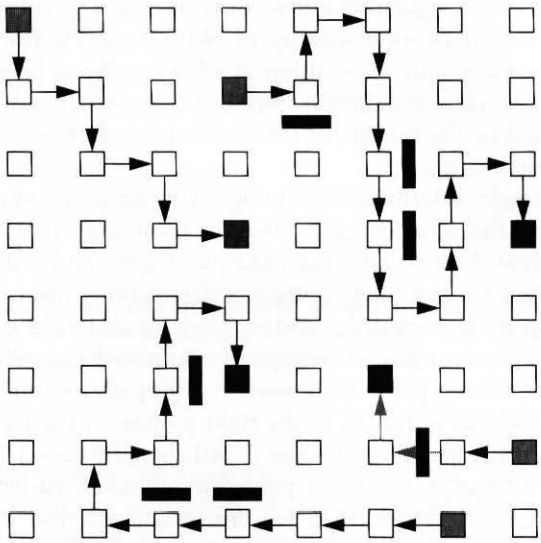


FIGURE 10.25 Examples of legal west-first routes in an 8 × 8 array. Substantial routing adaptation is obtained with turn-model routing, thus providing the ability to route around faults in a deadlock-free manner.

west-first routing. The elongated rectangles indicate blockages or broken links that might cause such a set of routes to be used. It should be clear that minimal turn models allow a great deal of flexibility in route selection. There are many legal paths between pairs of nodes.

The turn-model approach can be combined with virtual channels, and it can be applied in any topology. (In some networks, such as unidirectional d-cubes, virtual channels are still required.) The basic method is as follows: (1) partition the channels into sets according to the direction they route packets (excluding wraparound edges); (2) identify the potential cycles formed by "turns" between directions; and (3) prohibit one turn in each abstract cycle, being careful to break all the complex cycles as well. Finally, wraparound edges can be incorporated as long as they do not introduce cycles. If virtual channels are present, treat each set of channels as a distinct virtual direction.

Up* down* is essentially a turn-model algorithm with the assumption of bidirectional channels, using only two directions. Indeed, in reviewing the up*-down* algorithm, many shortest paths may conform to the up*-down* restriction, and certainly many nonminimal up*-down* routes are in most networks. Virtual channels allow the routing restrictions to be loosened even further.

10.6.7 Adaptive Routing

The fundamental advantage of loosening the routing restrictions is that it allows multiple legal paths between pairs of nodes. This is essential for fault tolerance. If

Qualcomm Ex. 1015

the routing algorithm allows only one path, failure of a single link will effectively leave the network disconnected. With multipath routing, it may be possible to steer around the fault. In addition, it allows traffic to be spread more broadly over available channels and thereby improves the utilization of the network. When a vehicle is parked in the middle of the street, it is often nice to have the option of driving around the block.

Simple deterministic routing algorithms can introduce tremendous contention within the network, even when the communication load is spread evenly over independent destinations. For example, Figure 10.26 shows a simple case where four packets are traveling to distinct destinations from distinct sources in a 2D mesh; under dimension order routing they are all forced to travel through the same link. The communication is completely serialized through the bottleneck while links for other shortest paths are unused. A multipath routing algorithm could use alternative channels, as indicated in the right portion of the figure. For any network topology there exist bad permutations (Gottlieb and Kruskal 1984), but simple deterministic routing makes these bad permutations much easier to run across. The particular example in Figure 10.26 has important practical significance. A common global communication pattern is a transpose. On a 2D mesh with dimension order routing, all the packets in a row must go through a single switch before filling in the column.

Multipath routing can be incorporated as an extension of any of the basic switch mechanisms. With source-based routing, the source simply chooses among the legal routes and builds the header accordingly. No change to the switch is required. With table-driven routing, this can be accomplished by setting up table entries for multiple paths. For arithmetic routing, additional control information would need to be incorporated in the header and interpreted by the switch.

Adaptive routing is a form of multipath routing where the choice of routes is made dynamically by the switch in response to traffic encountered en route. Formally, an adaptive routing function is a mapping of the form $R_A : C \times N \times \Sigma \rightarrow C$, where Σ represents the switch state. In particular, if one of the desired outputs is blocked or failed, the switch may choose to send the packet on an alternative channel. Minimal adaptive routing will only route packets along shortest paths to their destination, that is, every hop must reduce the distance to the destination. An adaptive algorithm that allows all shortest paths to be used is *fully adaptive*, otherwise it is *partially adaptive*. An interesting extreme case of nonminimal adaptive routing is what is called "hot potato" routing. In this scheme the switch never buffers packets. If more than one packet is destined for the same output channel, the switch sends one toward its destination and "misroutes" the rest onto other channels.

Adaptive routing is not widely used in current parallel machines, although it has been studied extensively in the literature (Ngai and Seitz 1989; Linder and Harden 1991), especially through the Chaos router (Kostantantindou and Snyder 1991). The CRAY T3E provides minimal adaptive routing in a cube. The nCUBE/3 is to provide minimal adaptive routing in a hypercube. The network proposed for the Tera machine (Alverson et al. 1990) is to use hot potato routing, with 128-bit packets delivered in one 3-ns cycle.

Qualcomm Ex. 1015

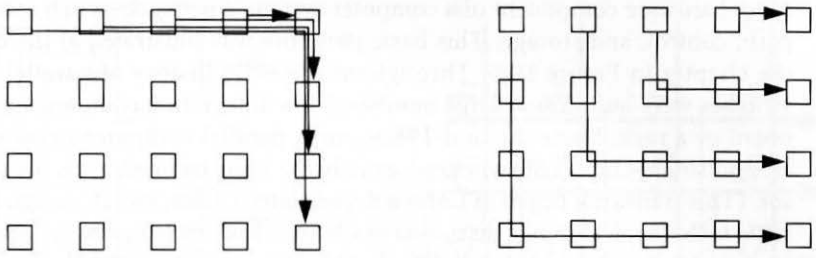


FIGURE 10.26 Routing path conflicts under deterministic dimension order routing. Several messages from distinct source to distinct destination contend for resources under dimension order routing, whereas an adaptive routing scheme may be able to use disjoint paths.

Although adaptive routing has clear advantages, it is not without its disadvantages. Clearly, it adds to the complexity of the switch, which can only make the switch slower. The reduction in bandwidth can outweigh the gains of the more sophisticated routing—a simple deterministic network in its linear operating regime is likely to outperform a clever adaptive network in saturation. In nonuniform networks, such as a d-dimensional array, adaptivity hurts performance on uniform random traffic. Stochastic variations in the load introduce temporary blocking in any network. For switches at the boundary of the array, this will tend to propel packets toward the center. As a result, contention forms in the middle of the array that is not present under deterministic routing. Adaptive routing can cause problems with certain kinds of nonuniform traffic as well, as we will see in Section 10.8.3.

Nonminimal adaptive routing tends to perform poorly as the network reaches saturation because packets traverse extra links and hence consume more bandwidth. The throughput of the network tends to drop off as load is increased rather than flattening at the saturation point, as illustrated in Figure 10.17.

Recently, there have been a number of proposals for low-cost partially and fully adaptive routing that use a combination of a limited number of virtual channels and restrictions on the set of turns (Chien and Kim 1992; Schwiebert and Jayasimha 1995). It appears that most of the advantages of adaptive routing, including fault tolerance and channel utilization, can be obtained with a very limited degree of adaptability.

10.7 SWITCH DESIGN

Ultimately, the design of a network boils down to the design of the switch and how the switches are wired together. The degree of the switch, its internal routing mechanisms, and its internal buffering determine what topologies can be supported and what routing algorithms can be implemented. Now that we understand the higher-level network design issues, let us return to switch design in more detail. Like any

Qualcomm Ex. 1015

other hardware component of a computer system, a network switch comprises datapath, control, and storage. This basic structure was illustrated at the beginning of the chapter in Figure 10.5. Throughout the early history of parallel computing, switches were built from a large number of low-integration components occupying a board or a rack. Since the mid-1980s, most parallel computer networks are built around single-chip VLSI switches—exactly the same technology as the microprocessor. (This transition began in LANs a decade later.) Thus, switch design is tied to the same technological trends discussed in Chapter 1: decreasing feature size, increasing area, and increasing pin count. We should view modern switch design from a VLSI perspective.

10.7.1 Ports

The total number of pins is essentially the total number of input and output ports times the channel width. Since the perimeter of the chip grows slowly compared to area, switches tend to be pin limited. This pushes designers toward narrow, high-frequency channels. Very high-speed serial links are especially attractive because they use the least pins and eliminate problems with skew across the bit lines in a channel. However, with serial links the clock and all control signals must be encoded within the framing of the serial bit stream. With parallel links, one of the wires is essentially a clock for the data on the others. Flow control is realized using an additional wire, providing a ready/acknowledge handshake.

10.7.2 Internal Datapath

The datapath is the connectivity between each of a set of input ports (i.e., input latches, buffers, or FIFO) and every output port. This is generally referred to as the internal crossbar, although it can be realized in many different ways. A *nonblocking crossbar* is one in which each input port can be connected to a distinct output in any permutation simultaneously. Logically, for an $n \times n$ switch the nonblocking crossbar is nothing more than an n-way multiplexer associated with each destination, as shown in Figure 10.27(a). The multiplexer may be implemented in a variety of different ways, depending on the underlying technology. For example, in VLSI it is typically realized as a single bus with n tristate drivers, shown in Figure 10.27(b). In this case, the control path provides n enable points per output. A technique that is becoming increasingly common is to use a memory as a crossbar by writing for each input port and reading for each output port; see Figure 10.27(c).

It is clear that the hardware complexity of the crossbar is determined by the wires. There are nw data wires in each direction, requiring $(nw)^2$ area. There are also n^2 control wires, which add to this significantly. How do we expect switches to track improvements in VLSI technology? Assume that the area of the crossbar stays constant, but the feature size decreases. The ideal VLSI scaling law says that if the feature size is reduced by a factor of s (including the gate thickness) and the voltage level is reduced by the same factor, then the speed of the transistors improves by a factor of $1/s$, the propagation delay of wires connecting neighboring transistors

Qualcomm Ex. 1015

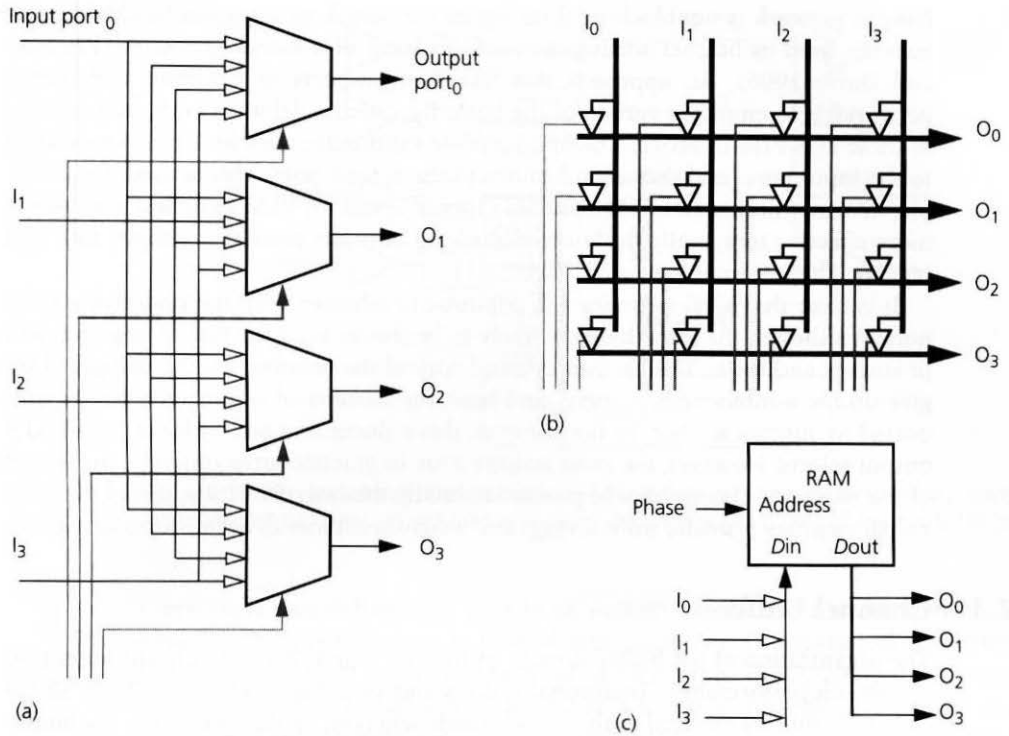


FIGURE 10.27 Crossbar implementations. The crossbar internal to a switch can be implemented as (a) a collection of multiplexers, (b) a grid of tristate drivers, or (c) via a contentional static RAM that time-multiplexes across the ports.

improves by a factor of $1/s$, and the total number of transistors per unit area increases by $1/s^2$ with the same power density. For switches, this means that the wires get thinner and closer together, so the degree of the switch can increase by a factor of $1/s$. Notice that the switch degree improves as only the square root of the improvement in logic density. The bad news is that these wires run the entire length of the crossbar, hence the length of the wires stays constant. The wires get thinner, so they have more resistance. The capacitance is reduced and the net effect is that the propagation delay is unchanged (Bakoglu 1990). In other words, ideal scaling gives us an improvement in switch degree for the same area but no improvement in speed. The speed does improve by a factor of $1/s$ if the voltage level is held constant, but then the power increases with $1/s^2$. Increases in chip area will allow larger degree, but the wire lengths increase and the propagation delays will increase.

Some degree of confusion exists over the term "crossbar." In the traditional switching literature, multistage interconnection networks that have a single controller are sometimes called crossbars, even though the datapath through the switch is organized as an interconnection of small crossbars. In many cases these are connected in a manner similar to a butterfly topology, called a Banyan network. A

Qualcomm Ex. 1015

Banyan network is nonblocking if its inputs are sorted, so some nonblocking crossbars are built as batcher sorting networks in front of a Banyan network (Peterson and Davie 1996). An approach that has many aspects in common with Benes networks is to employ a variant of the butterfly, called a delta network, and use two of these networks in series. The first serves to randomize a packet's position relative to the input ports and the second routes to the output port. This is used, for example, in some commercial ATM switches (Turner 1988). In VLSI switches, it is usually more effective to actually build a nonblocking crossbar since it is simple, fast, and regular. The key limit is pins anyway.

It is clear that VLSI switches will continue to advance with the underlying technology, although the growth rate is likely to be slower than the rate of improvement in storage and logic. The hardware complexity of the crossbar can be reduced if we give up the nonblocking property and limit the number of inputs that can be connected to outputs at once. In the extreme, this reduces to a bus with n drivers and n output selects. However, the most serious issue in practice turns out to be the length of the wires and the number of pins, so reducing the internal bandwidth of the individual switches provides little savings and a significant loss in network performance.

10.7.3 Channel Buffers

The organization of the buffer storage within the switch has a significant impact on the switch performance. Traditional routers and switches tend to have large SRAM or DRAM buffers external to the switch fabric whereas, in VLSI switches, the buffering is internal to the switch and comes out of the same silicon budget as the datapath and the control section. There are four basic options: no buffering (just input and output latches), buffers on the inputs, buffers on the outputs, or a centralized shared buffer pool. A few flits of buffering on input and output channels decouples the switches on either end of the link and tends to provide a significant improvement in performance. As chip size and density increase, more buffering is available and the network designer has more options, but still the buffer real estate comes at a premium and its organization is important. Like so many other aspects in network design, the issue is not just how effectively the buffer resources are utilized but how the buffering affects the utilization of other components of the network.

Intuitively, we might expect sharing of the switch storage resources to be harder to implement but to allow better utilization of these resources than partitioning the storage among ports. All of the communication ports need to access the shared pool simultaneously, requiring a very high-bandwidth memory. More surprisingly, sharing the buffer pool on demand can hurt the network utilization in some cases because a single congested output port can hog most of the buffer pool and thereby prevent other traffic from moving through the switch.

Input Buffering

One attractive approach is to provide independent FIFO buffers with each input port, as illustrated in Figure 10.28. Each buffer needs to be able to accept a phit

Qualcomm Ex. 1015

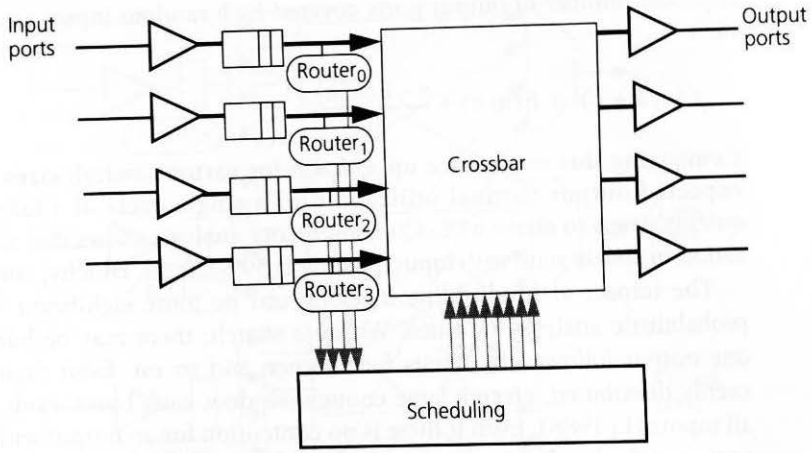


FIGURE 10.28 Input buffered switch. A FIFO is provided at each of the input ports, but the controller can only inspect and service the packets at the heads of the input FIFOs.

every cycle and deliver one phit to an output, so the internal bandwidth of the switch is easily matched to the flow of data coming in. The operation of the switch is relatively simple; it monitors the head of each input FIFO, computes the desired output port of each, and schedules packets to move through the crossbar accordingly. Typically, routing logic is associated with each input port to determine the desired output. This is trivial for source-based routing; it requires an arithmetic unit per input for algorithmic routing and, typically, a routing table per input with table-driven routing. With cut-through routing, the decision logic does not make an independent choice every cycle but only every packet. Thus, the routing logic is essentially a finite state machine, which spools all the flits of a packet to the same output channel before making a new routing decision at the packet boundary (Seitz and Su 1993).

One problem with the simple input buffered approach is the occurrence of "head-of-line" blocking. Suppose that two ports have packets destined for the same output port. One of them will be scheduled onto the output and the other will be blocked. The packet just behind the blocked packet may be destined for one of the unused outputs (there are guaranteed to be unused outputs), but it will not be able to move forward. This head-of-line blocking problem is familiar in our vehicular traffic analogy; it corresponds to having only one lane approaching an intersection. If the car ahead is blocked in attempting to turn, there is no way to proceed down the empty street ahead.

We can easily estimate the effect of head-of-line blocking on channel utilization. If we have two input ports and randomly pick an output for each, the first succeeds and the second has a 50/50 chance of picking the unused output. Thus, the expected number of packets per cycle moving through the switch is 1.5 and, hence, the expected utilization of each output is 75%. Generalizing this, if $E(n, k)$ is the

Qualcomm Ex. 1015

expected number of output ports covered by k random inputs to an n-port switch, then

$$E(n, k+1) = E(n, k) + \frac{n - E(n, k)}{n}$$

Computing this recurrence up to $k = n$ for various switch sizes reveals that the expected output channel utilization for a single cycle of a fully loaded switch quickly drops to about 65%. Queuing theory analysis shows that the expected utilization in steady state with input queuing is 59% (Karol, Hluchyj, and Morgan 1987).

The impact of head-of-line blocking can be more significant than this simple probabilistic analysis indicates. Within a switch, there may be bursts of traffic for one output followed by bursts for another, and so on. Even though the traffic is evenly distributed, given a large enough window, each burst results in blocking on all inputs (Li 1988). Even if there is no contention for an output within a switch, the packet at the head of an input buffer may be destined for an output that is blocked due to congestion elsewhere in the network. Still, the packets behind it cannot move forward. In a wormhole routed network, the entire worm will be stuck in place, effectively consuming link bandwidth without going anywhere. A more flexible organization of buffer resources might allow packets to slide ahead of packets that are blocked.

Output Buffering

The basic enhancement we need to make to the switch is to provide a way for it to consider multiple packets at each input as candidates for advancement to the output port. A natural option is to expand the input FIFOs to provide an independent buffer for each output port so that packets sort themselves by destination upon arrival, as indicated by Figure 10.29. (This is the kind of switch assumed by the conventional delay analysis in Section 10.5; the analysis is simplified because the switch does not introduce additional contention effects internally.) With a steady stream of traffic on the inputs, the outputs can be driven at essentially 100%. However, the advantages of such a design are not without a cost; additional buffer storage and internal interconnect are required.[6] Along with the sorting stage and the wider multiplexers, this may increase the switch cycle time or increase its routing delay.

It is a matter of perspective whether the buffers in Figure 10.29 are associated with the input or the output ports. If viewed as output port buffers, the key property is that each output port has enough internal bandwidth to receive a packet from every input port in one cycle. This could be obtained with a single output FIFO, but it would have to run at an internal clock rate of n times that of the input ports.

---

6. It is possible to provide the capability of the output buffered switch but avoid the storage and interconnect penalty (Joerg 1994). The set of buffers at each input forms a pool and each output has a list of pointers to packets destined for it. The timing requirement in this design is the ability to push n pointers per cycle into the output port buffer rather than n packets per cycle.

Qualcomm Ex. 1015

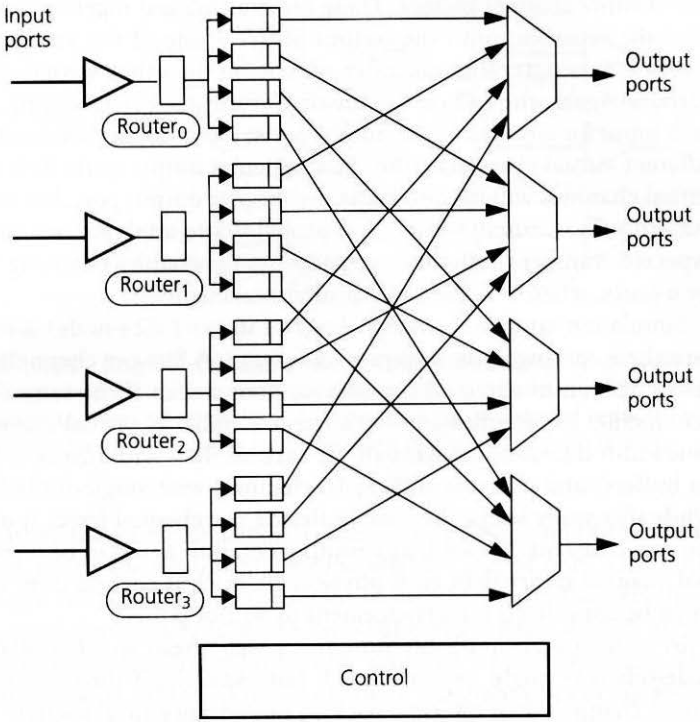


FIGURE 10.29 Switch design to avoid head-of-line blocking. Packets are sorted by output port at each input port so that the controller can schedule a packet for an output port if any input has a packet destined for that output.

Shared Pool

With a shared pool, each of the input ports deposits data into a central memory, and each of the output buffers reads from it. Head-of-line blocking is avoided because input ports can write to the pool regardless of output port, assuming space is available. The challenge is to match the bandwidth of the n-input and n-output ports. A common trick is to make the internal datapath to the pool $2n$ times as wide as the links. Each input port buffers $2n$ phits before writing it to the pool, and each output port gets $2n$ phits at a time. Often these shared pools are built from the SRAM technology used for caches.

Virtual Channels Buffering

Virtual channels suggest an alternative way of organizing the internal buffers of the switch. Recall that a set of virtual channels provides transmission of multiple independent packets across a single physical link. As illustrated in Figure 10.21, to support virtual channels the flow across the link is split upon arrival at an input port

Qualcomm Ex. 1015

into distinct channel buffers. These are multiplexed together again, either before or after the crossbar, onto the output ports. If one of the virtual channel buffers is blocked, it is natural to consider advancing the other virtual channels toward the outputs. Again, the switch has the opportunity to select among multiple packets at each input for advance to the output ports; however, in this case the choice is among different virtual channels rather than different output ports. It is possible that all the virtual channels will need to route to the same output port, but expected coverage of the outputs is much better. In a probabilistic analysis, we can ask: what is the expected number of distinct output ports covered by choosing among vn requests for n ports, where v is the number of virtual channels?

Simulation studies show that large (256- to 1,024-node) 2-ary butterflies using wormhole routing with moderate buffering (16 flits per channel) saturate at a channel utilization of about 25% under random traffic. If the same 16 flits of buffering per channel are distributed over a larger number of virtual channels, the saturation bandwidth increases substantially. It exceeds 40% with just two virtual channels (8-flit buffers) and is nearly 80% at 16 channels with single-flit buffers (Dally 1990a). While this study keeps the total buffering per channel fixed, it does not really keep the cost constant. Notice that a routing decision needs to be computed for each virtual channel rather than each physical channel, if packets from any of the channels are to be considered for advancement to output ports.

By now you are probably jumping a step ahead to additional cost-performance trade-offs that might be considered. For example, if the crossbar has an input per virtual channel, then multiple packets can advance from a single input at once. This increases the probability of each packet advancing and, hence, the channel utilization. The crossbar increases in size in only one dimension, and the multiplexers are eliminated since each output port is logically a vn-way multiplexer. Switch design allows a great deal of room for innovation.

10.7.4 Output Scheduling

We have seen routing mechanisms that determine the desired output port for each input packet, datapaths that provide a connection from the input ports to the outputs, and buffering strategies that allow multiple packets per input port to be considered as candidates for advancement to the output port. A key missing component in the switch design is the *scheduling algorithm*, which selects the packets to advance in each cycle. Given a selection, the remainder of the switch control asserts the control points in the crossbars or multiplexers and the buffers or latches to effect the register transfer from each selected input to the associated output. As with the other aspects of switch design, there is a spectrum of solutions varying from simple to complex.

A simple approach is to view the scheduling problem as n independent arbitration problems, one for each output port. Each candidate input buffer has a request line to each output port and a grant line from each port, as indicated by Figure 10.30. (The figure shows four candidate input buffers driving three output ports to indicate that routing logic and arbitration input is on a per-input-buffer

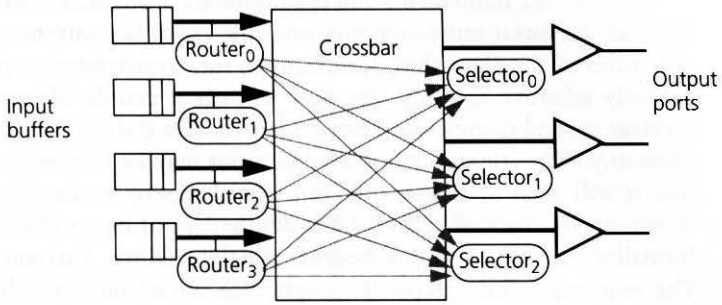


FIGURE 10.30 Control structure for output scheduling. Associated with each input buffer is routing logic to determine the output port, a request line, and a grant line per output. Each output has selection logic to arbitrate among asserted requests and to assert one grant, causing a flit to advance from the input buffer to the output port.

basis rather than per input port.) The routing logic computes the desired output port and asserts the request line for the selected output port. The output port scheduling logic arbitrates among the requests, selects one, and asserts the corresponding grant signal. Specifically, with the crossbar using tristate drivers in Figure 10.27(b), output port j enables input buffer i by asserting control enable e_{ij}. The input buffer logic advances its FIFO as a result of one of the grant lines being asserted.

An additional design question is the arbitration algorithm used for scheduling flits onto the output. Options include static priority, random, round-robin, and oldest-first scheduling. Each of these have different performance characteristics and implementation complexity. Clearly, static priority is the simplest; it is simply a priority encoder. However, in a large network it can cause indefinite postponement. In general, scheduling algorithms that provide fair service to the inputs perform better. Round-robin requires extra state to change the order of priority in each cycle. Oldest-first tends to have the same average latency as random assignment but significantly reduces the variance in latencies (Dally 1990a). One way to implement oldest-first scheduling is to use a control FIFO of input port numbers at each output port. When an input buffer requests an output, the request is enqueued. The oldest request at the head of the FIFO is granted.

It is useful to consider the implementation of various routing algorithms and topologies in terms of Figure 10.30. For example, in a direct d-cube there are $d + 1$ inputs (let's number them $i_0, \ldots, i_d$) and $d + 1$ outputs (numbered $o_1, \ldots, o_{d+1}$) with the host connected to input i_0 and output o_{d+1}. The straight path $i_j \rightarrow o_j$ corresponds to routing in the same dimension; other paths are a change in dimension. With dimension order routing, packets can only increase in dimension as they cross the switch. Thus, the full complement of request/grant logic is not required. Input j need only request outputs $j, \ldots, d + 1$ and output j need only grant inputs $0, \ldots, j$. Obvious static priority schemes assign priority in increasing or decreasing numerical order.

Qualcomm Ex. 1015

What are the implementation requirements of adaptive routing? First, the routing logic for an input must compute multiple candidate outputs according to the specific rules of the algorithm, for example, turn restrictions or plane restrictions. For partially adaptive routing, this may be only a couple of candidates. Each output receives several requests and can grant one. (Or if the output is blocked, it may not grant any.) The tricky point is that an input may be selected by more than one output. It will need to choose one, but what happens to the other outputs? Should it iterate on its arbitration and select another input or go idle? This problem can be formalized as one of *on-line bipartite matching* (Karp, Vazirani, and Vazirani 1990). The requests define a bipartite graph with inputs on one side and outputs on the other. The grants (one per output) define a matching of input/output pairs within the request graph. The maximum matching would allow the largest number of inputs to advance in a cycle, which ought to give the highest channel utilization. Viewing the problem in this way, the switch scheduling logic should approximate a fast parallel matching algorithm (Anderson et al. 1992). The basic idea is to form a tentative matching using a simple greedy algorithm, such as random selection among requests at each output followed by selection of grants at each input; then, for each unselected output, try to make an improvement to the tentative matching. In practice, the improvement diminishes after a couple of iterations. Clearly, this is another case of sophistication versus speed. If the scheduling algorithm increases the switch cycle time or routing delay, it may be better to accept a little extra blocking and get the job done faster.

This maximum matching problem applies to the case where multiple virtual channels are multiplexed through each input of the crossbar, even with deterministic routing. (Indeed, the technique was proposed for the AN2 ATM switch to address the situation of scheduling cells from several "virtual circuits."[7]) Each input port has multiple buffers that it can schedule onto its crossbar input, and these may be destined for different outputs. The selection of the outputs determines which virtual channel to advance. If the crossbar is widened, rather than multiplexing the inputs, the matching problem vanishes and each output can make a simple independent arbitration.

10.7.5 Stacked Dimension Switches

Many aspects of switch design are simplified if there are only two inputs and two outputs, including the control, arbitration, and datapaths. Several designs, including the torus routing chip (Seitz and Su 1993), the J-machine, and the CRAY T3D, have used a simple 2×2 building block and stacked these to construct switches of higher dimension, as illustrated in Figure 10.31. If we have in mind a d-cube, traffic continuing in a given dimension passes straight through the switch in that dimension, whereas if it needs to turn into another dimension it is routed vertically through the

7. Virtual circuits should not be confused with virtual channels. The former is a technique for associating routing of resources along an entire source-to-destination route. The latter is a strategy for structuring the buffering associated with each link.

Qualcomm Ex. 1015

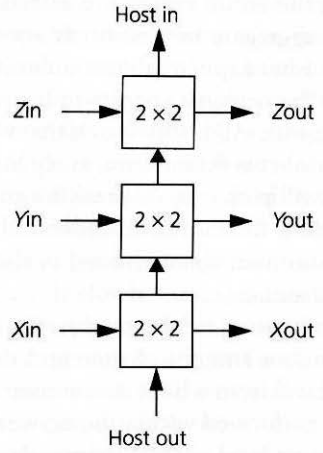


FIGURE 10.31 Stacked dimension switch. Traffic continuing in a dimension passes straight through one 2 × 2 switch, whereas when it turns to route in another dimension it is routed vertically up the stack.

switch. Notice that this adds a hop for all but the lowest dimension. This same technique yields a topology called *cube-connected cycles* when applied to a hypercube. Each $n \times n$ node of the hypercube is replaced by a ring of n 2 × 2 nodes.

10.8 FLOW CONTROL

In this section, we consider in detail what happens when multiple flows of data in the network attempt to use the same shared network resources at the same time. Some action must be taken to control these flows. If no data is to be lost, some of the flows must be blocked while others proceed. The problem of flow control arises in all networks and at many levels, but it is qualitatively different in parallel computer networks from that in local and wide area networks. In parallel computers, network traffic needs to be delivered about as reliably as traffic across a bus, and a very large number of concurrent flows occur on very small time scales. No other networking regime has such stringent demands. We will look briefly at some of these differences and then examine in detail how flow control is addressed at the link level and end to end in parallel machines.

10.8.1 Parallel Computer Networks versus LANs and WANs

To build intuition for the unique flow control requirements of the parallel machine networks, let us take a little digression and examine the role of flow control in the networks we deal with every day for file transfer and the like. We will look at three examples: Ethernet-style collision-based arbitration, FDDI-style global arbitration, and unarbitrated wide area networks.

Qualcomm Ex. 1015

In an Ethernet, the entire network is effectively a single shared wire, like a bus, only longer. (The aggregate bandwidth is equal to the link bandwidth.) However, unlike a bus, there is no explicit arbitration unit. A host attempts to send a packet by first checking that the network appears to be quiet and then (optimistically) driving its packet onto the wire. All nodes watch the wire, including the hosts attempting to send. If only one packet is on the wire, every host will "see" it, and the host specified as the destination will pick it up. If there is a collision, every host will detect the garbled signal, including the multiple senders. The minimum that a host may drive a packet (i.e., the minimum channel time) is about 50 μs; this is to allow time for all hosts to detect collisions.

The flow control aspect is how the retry is handled. On a collision, each sender backs off for a random amount of time and then retries. With each repeated collision, the retry interval from which the random delay is chosen is increased. The collision detection is performed within the network interface hardware, and the retry is handled by the lowest level of the Ethernet driver. If there is no success after a large number of tries, the Ethernet driver gives up and drops the packet. However, the message operation originated from some higher-level communication software layer that has its own delivery contract. For example, the TCP/IP layer will detect the delivery failure via a time-out and engage its own adaptive retry mechanism, just as it does for wide area connection, which we will look at next. The UDP layer will ignore the delivery failure, leaving it to the user application to detect the event and retry. The basic concept of the Ethernet rests on the assumption that the wire is very fast compared to the communication capability of the hosts. (This assumption was reasonable in the mid-1970s when Ethernet was developed.) A great deal of study has been given to the properties of collision-based media access control, but basically as the network reaches saturation, the delivered bandwidth drops precipitously.

Ring-based LANs, such as token ring and FDDI, use a distributed form of global arbitration to control access to the shared medium. A special arbitration token circulates the ring when there is an empty slot. A host that desires to send a packet waits until the token comes around, grabs it, and drives the packet onto the ring. After the packet is sent, the arbitration token is returned to the ring. In effect, flow control is performed at the hosts on every packet as part of gaining access to the ring. Even if the ring is idle, a host must wait for the token to traverse half the ring, on average. (This is why the unloaded latency for FDDI is generally higher than that of Ethernet.) However, under high load, the full link capacity can be used. Again, the basic assumption underlying this global arbitration scheme is that the network operates on a much smaller timescale than the communication operations of the hosts.

In the wide area case, each TCP connection (and each UDP packet) follows a path through a series of switches, bridges, and routers across media of varying speeds between source and destination. Since the Internet is a graph rather than a simple linear structure, at any point a set of incoming flows may have a collective bandwidth greater than the outgoing link. Traffic will back up as a result. Wide area routers provide a substantial amount of buffering to absorb stochastic variations in the flows, but if the contention persists, these buffers will eventually fill up. In this case, most routers will just drop the packets. Wide area links may be stretches of

Qualcomm Ex. 1015

fibers many miles long, so when the packet is driven onto a link it is hard to know if it will find buffer space available when it arrives at the other end. Furthermore, the flows of data through a switch are usually unrelated transfers. They are not, in general, a collection of flows from within a single parallel program, which imposes a degree of inherent end-to-end flow control by waiting for data it needs before continuing. The TCP layer provides end-to-end flow control and adapts dynamically to the perceived characteristics of the route occupied by a connection. It assumes that packet loss (detected via time-out) is a result of contention at some intermediate point, so when it experiences a loss, it sharply decreases the send rate (by reducing the size of its burst window). It slowly increases this rate (i.e., the window size) as data is transferred successfully (detected via acknowledgments from destination to source) until it once again experiences a loss. Thus, each flow is controlled at the source governed by the occurrence of time-outs and acknowledgments.

Of course, the wide area case operates on a timescale of fractions of a second whereas parallel machine networks operate on a scale of nanoseconds. Thus, one should not expect the techniques to carry over directly. Interestingly, the TCP flow control mechanisms do work well in the context of collision-based arbitration such as Ethernet (partly because the software overhead time tends to give a chance for the network to clear). However, with the emergence of high-speed, switched local and wide area networks, especially ATM, the flow control problem has taken on many more of the characteristics of the parallel machine case. Most commercial ATM switches provide a sizable amount of buffering per link (typically 64 to 128 cells per link) but drop cells when this is exceeded. Each cell is 53 bytes, so at the 155-Mb/s OC-3 rate, a cell transfer time is 2.7 μs. Buffers fill very rapidly compared to typical LAN/WAN end-to-end times. The TCP mechanisms can be ineffective in the ATM settings when contending with more aggressive protocols, such as UDP, prompting the ATM standardization efforts to include link-level flow and rate control measures.

10.8.2 Link-Level Flow Control

Essentially all parallel machine interconnection networks provide link-level flow control. The basic problem is illustrated in Figure 10.32. Data is to be transferred from an output port of one node across a link to an input port of a node operating autonomously. The storage may be a simple latch, a FIFO, or buffer memory. The link may be short or long, wide or narrow, synchronous or asynchronous. The key point is that, as a result of circumstances at the destination node, storage at the input port may not be available to accept the transfer, so the data must be retained at the source until the destination is ready. This may cause the buffers at the source to fill, and it in turn may exert pressure back on its sources.

The implementation of link-level flow control differs depending on the design of the link, but the main idea is the same. The destination node provides feedback to the source, indicating whether it is able to receive additional data on the link. The source holds onto the data until the destination indicates that it is able. Before we examine how this feedback is incorporated into the switch operation, let us look at how the flow control is implemented on different kinds of links.

Qualcomm Ex. 1015

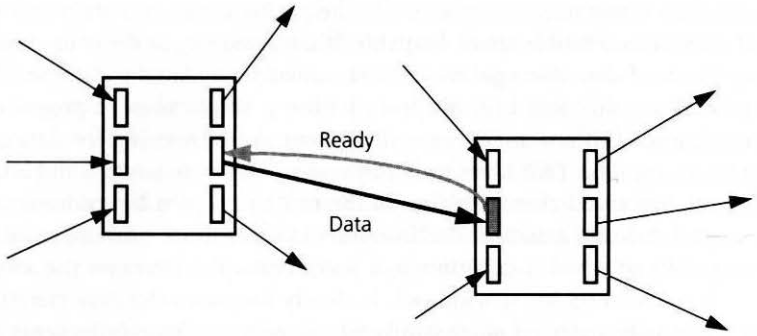


FIGURE 10.32 Link-level flow control. As a result of circumstances at the destination node, the storage at the input port may not be available to accept the data transfer, so the data must be retained at the source until the destination is ready.

With short-wide links, the transfer across the link is essentially like a register transfer within a machine, extended with a couple of control signals. We may view the source and destination registers as being extended with a full-empty bit, as illustrated in Figure 10.33. If the source is full and the destination is empty, the transfer occurs, the destination becomes full, and the source becomes empty (unless it is refilled from its source). With synchronous operation (e.g., in the CRAY T3D, IBM SP-2, TMC CM-5, and MIT J-machine), the flow control determines whether a transfer occurs for the clock cycle. It is easy to see how this is realized with edge-triggered or multiphase level-sensitive designs. If the switches operate asynchronously, the behavior is much like a register transfer in a self-timed design. The source asserts the request (req) signal when it is full and ready to transfer; the destination uses this signal to accept the value (when the input port is available) and asserts an acknowledgment (ack) when it has accepted the data. With short-narrow links, the behavior is similar, except that a series of phits is transferred for each req/ack handshake.

The req/ack handshake can be viewed as the transfer of a single token or credit between the source and the destination. When the destination frees the input buffer, it passes the token to the source (i.e., increases its credit). The source uses this credit when it sends the next flit and must wait until its account is refilled. For long links, this credit scheme is expanded so that the entire pipeline associated with the link propagation delay can be filled. Suppose that the link is sufficiently long that several flits are in transit simultaneously. As indicated by Figure 10.34, it will also take several cycles for acks to propagate in the reverse direction, so a number of acks (credits) may be in transit as well. The obvious credit-based flow control is for the source to keep account of the available slots in the destination input buffer. The counter is initialized to the buffer size. It is decremented when a flit is sent, and the output is blocked if the counter reaches zero. When the destination removes a flit from the input buffer, it returns a credit to the source, which increments the counter. The input buffer will never overflow; there is always room to drain the link into the buffer. This approach is most attractive with wide links, which have dedicated control lines

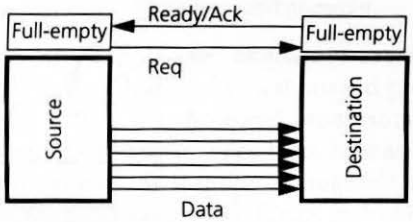


FIGURE 10.33 Simple link-level handshake. The source asserts its request when it has a flit to transmit; the destination acknowledges the receipt of a flit when it is ready to accept the next one. Until that time, the source repeatedly transmits the flit.

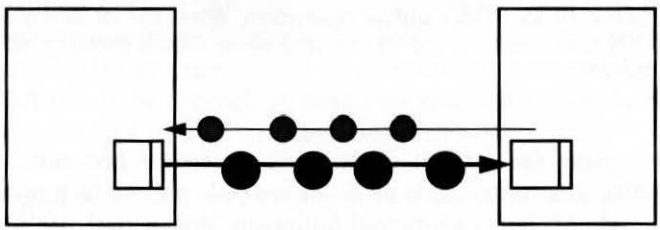


FIGURE 10.34 Transient flits and acks with long links. With longer links, the flow control scheme needs to allow more slack in order to keep the link full. Several flits can be driven onto the wire before waiting for acks to come back.

for the reverse ack. For narrow links that multiplex ack symbols onto the opposite-going channel, the ack per flit can be reduced by transferring bigger chunks of credit. However, the problem remains that the approach is not very robust to the loss of credit tokens.

Ideally, when the flows are moving forward smoothly there is no need for flow control. The flow control mechanism should be a governor that gently nudges the input rate to match the output rate. The propagation delays on the links give the system momentum. An alternative approach to link-level credits is to view the destination input buffer as a staging tank with a low-water mark and a high-water mark, as in Figure 10.35. When the fill level drops below the low mark, a GO symbol is sent to the source, and when it goes over the high mark, a STOP symbol is generated. There must be enough room below the low mark to withstand a full round-trip delay (the GO propagating to the source, being processed, and the first of a stream of flits propagating to the destination). In addition, there must be enough headroom above the high mark to absorb the round-trip's worth of flits that may be in flight. A nice property of this approach is that redundant GO symbols may be sent anywhere below the high mark and STOP symbols may be sent anywhere above the low mark with no harmful effect, so they are simply sent periodically in the two regimes. The fraction of the link bandwidth used by flow control symbols can be reduced by

Qualcomm Ex. 1015

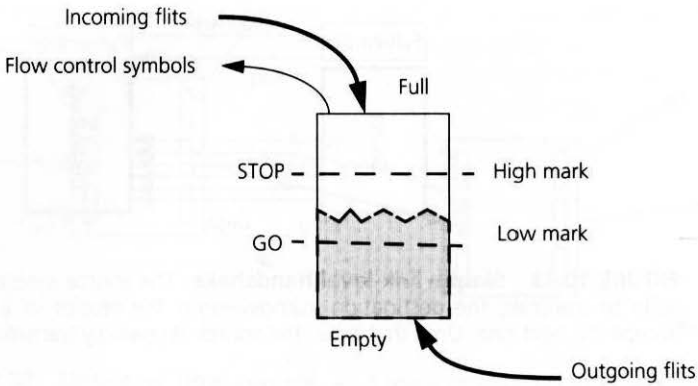


FIGURE 10.35 Slack buffer operation. When the fill level drops below the low mark, a GO symbol is sent to the source, and when it goes over the high mark, a STOP symbol is generated.

increasing the amount of storage between the low and high marks in the input buffer. This approach is used, for example, in Cal Tech routers (Seitz and Su 1993) and the Myrinet commercial follow-on (Boden et al. 1995). Similar techniques are used in modems.

It is worth noting that the link-level flow control is used on host-switch links as well as switch-switch links. In fact, it is generally carried over to the processor-NI interface as well. However, the techniques may vary for these different kinds of interfaces. For example, in the Intel Paragon the 175-MB/s network links are all short with very small flit buffers. However, the NIs have large input and output FIFOs. The communication assist includes a pair of DMA engines that can burst at 300 MB/s between memory and the network FIFOs. It is essential that the output (input) buffer not hit full (empty) in the middle of a burst because holding the bus transaction would hurt performance and potentially lead to deadlock. Thus, the burst is matched to the size of the middle region of the buffer and the high/low marks to the control turnaround between the NI and the DMA controller.

10.8.3 End-to-End Flow Control

Link-level flow control exerts a certain amount of end-to-end control because if congestion persists, buffers will fill up and the flow will be controlled all the way back to the source host nodes, called *back pressure*. For example, if k nodes are sending data to a single destination, they must eventually all slow to an average bandwidth of $1/k$th the output bandwidth. If the switch scheduling is fair and all the routes through the network are symmetric, back pressure will be enough to affect this. The problem is that by the time the sources feel the back pressure and regulate their output flow, all the buffers in the tree from the hot spot to the sources are full.

Qualcomm Ex. 1015

Hot Spots

The hot spot problem received quite a bit of attention as the technology reached a point where machines of several hundred to a thousand processors became reasonable (Pfister and Norton 1985). If a thousand processors deliver on average any more than 0.1% of their traffic to any one destination, that destination becomes saturated. If this situation persists, as it might if frequent accesses are made to an important shared variable, a saturation tree forms in front of this destination that will eventually reach all the way to the sources. At this point, all the remaining traffic in the system is severely impeded. The problem is particularly pernicious in butterflies since there is only one route to each destination and a great deal of sharing takes place among routes from one destination. Adaptive routing proves to make the hot spot problem even worse because traffic destined for the hot spot is sure to run into contention and to be directed to alternate routes. Eventually, the entire network clogs up. Large network buffers do not solve the problem; they just delay the onset. The time for the hot spot to clear is proportional to the total amount of hot spot traffic buffered in the network, so adaptivity and large buffers increase the time required for the hot spot to clear after the load is removed.

Various mechanisms have been developed to mitigate the causes of hot spots, such as having all nodes that need to increment a shared variable perform a parallel scan operation, as discussed in Chapter 7, or to implement combining fetch&add operations within the network (Pfister et al. 1985; Gottlieb, Lubachevsky, and Rudolph 1983). However, these only address situations where the problematic traffic is logically related. The more fundamental problem is that link-level flow control is like stopping on the freeway. Once the traffic jam forms, you are stuck. The better solution is not to get on the freeway at such times. With fewer packets inside the network, the normal mechanisms can do a better job of getting traffic to its destinations. This is one of the reasons that the BBN butterfly retracts the circuit established by a message upon collision.

Global Communication Operations

Problems with simple back pressure have been observed with completely balanced communication patterns, such as each node sending k packets to every other node. This occurs in many situations, including transposing a global matrix, converting between blocked and cyclic layouts, or the decimation step of an FFT (Brewer and Kuszmaul 1994; Dusseau et al. 1996). Even if the topology is robust enough to avoid serious internal bottlenecks on these operations, which is true of a fat-tree but not a low-degree dimension order mesh (Leighton 1992), a temporary backlog can have a cascading effect. When one destination falls behind in receiving packets from the network, a backlog begins to form. If priority is given to draining the network, this node will fall behind in sending to the other nodes. They, in turn, will send more than they receive and the backlog will tend to grow.

Simple end-to-end protocols in the global communication routines have been shown to mitigate this problem in practice. For example, a node may wait after

Qualcomm Ex. 1015

sending a certain amount of data until it has also received this amount, or it may wait for chunks of its data to be acknowledged. These precautions keep the processors more closely in step and introduce small gaps in the traffic flow, which decouples the processor-to-processor interaction through the network. (Interestingly, this technique is employed with the metering lights on heavily used bridges. Periodically, a wavefront of cars is injected into the bridge, separated by small gaps. This reduces the stochastic temporary blocking and avoids cascading blockage.)

Admission Control

With shallow, cut-through networks, the latency is low below saturation. Indeed, in most modern parallel machine networks, a single small message (or perhaps a few messages) will occupy an entire path from source to destination. If the remote network interface is not ready to accept the message, it is better to keep it within the source NI rather than blocking traffic within the network. One approach to achieving this is to perform NI-to-NI credit-based flow control. One study examining such techniques for a range of networks (Callahan and Goldstein 1995) indicates that allowing a single outstanding message per pair of NIs gives good throughput and maintains low latency.

10.9 CASE STUDIES

Networks are a fascinating area of study from a practical design and engineering viewpoint because they do one simple operation—move information from one place to another—and yet there is a huge space of design alternatives. Although the most apparent characteristics of a network are its topology, link bandwidth, switching strategy, and routing algorithm, several more characteristics must be specified to completely describe the design. These include the cycle time, link width, switch buffer capacity and allocation strategy, routing mechanism, switch output selection algorithm, and flow control mechanism. Each component of the design can be understood and optimized in isolation, but they all interact in interesting ways to determine the network performance on any particular traffic pattern in the context of the node architecture and the dependences embedded in the programs running on the platform.

This section summarizes a set of concrete network design points in important commercial and research parallel architectures. Using the framework established in this chapter, it systematically outlines the key design parameters.

10.9.1 CRAY T3D Network

The CRAY T3D network consists of a three-dimensional bidirectional torus of up to 1,024 switch nodes, each connected directly to a pair of processors,[8] with a data rate

8. Special I/O gateway nodes attached to the boundary of the cube include two processors each attached to a two-dimensional switch node connected into the x and z dimensions.

Qualcomm Ex. 1015

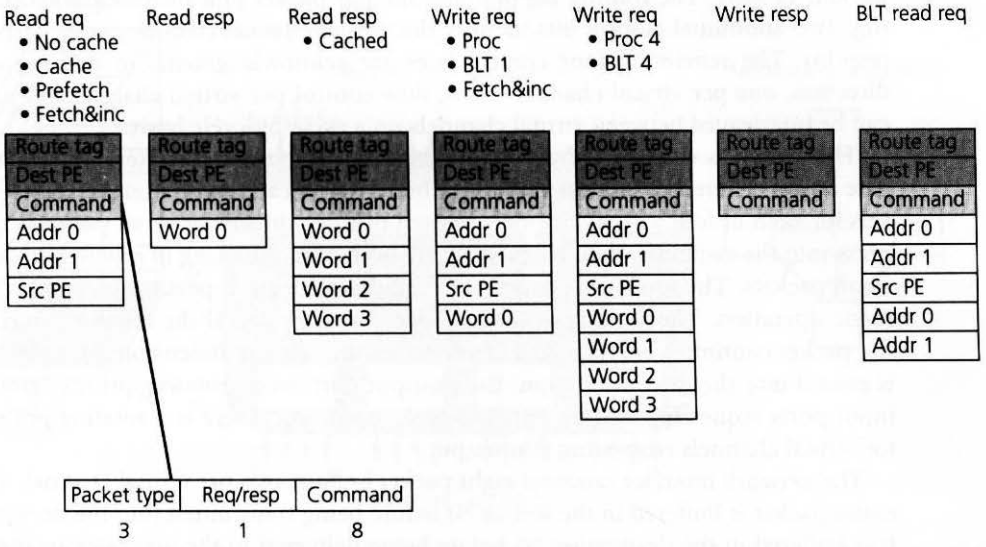


FIGURE 10.36 CRAY T3D packet formats. All packets consist of a series of 16-bit phits, with the first three being the route and tag, the destination processor number, and the command. Parsing and processing of the rest of the packet is determined by the tag and command.

of 300 MB/s per channel. Each node is on a single board, along with two processors and memory. There are up to 128 nodes (256 processors) per cabinet; larger configurations are constructed by cabling together cabinets. Dimension order, cut-through, packet-switched routing is used. The design of the network is very strongly influenced by the higher-level design of the system. Logically independent request and response networks are supported, with two virtual channels each to avoid deadlock, over a single set of physical links. Packets are variable length in multiples of 16 bits, as illustrated by Figure 10.36, and network transactions include various reads and writes plus the ability to start a remote block transfer engine (BLT), which is basically a DMA device. The first phit always contains the route, followed by the destination address, and the packet type or command code. The remainder of the payload depends on the packet type, consisting of relative addresses and data. All of the packet is parity protected, except the routing phit. If the route is corrupted, it will be misrouted, and the error will be detected at the destination because the destination address will not match the value in the packet.[9]

T3D links are short, wide, and synchronous. Each unidirectional link is 16 data and 8 control bits wide, operating under a single 150-MHz clock. Flits and phits are 16 bits. Two of the control bits identify the phit type (00 no info, 01 routing tag, 10

9. This approach to error detection reveals a subtle aspect of networks. If the route is corrupted, there is a very small probability that it will take a wrong turn at just the wrong time and collide with another packet in a manner that causes deadlock within the network; in this unlikely case the end node error detection will not be engaged.

Qualcomm Ex. 1015

packet, 11 last). The routing tag phit and the last-packet phit provide packet framing. Two additional control bits identify the virtual channel (req-hi, req-lo, resp-hi, resp-lo). The remaining four control lines are acknowledgments in the opposite direction, one per virtual channel. Thus, flow control per virtual channel and phits can be interleaved between virtual channels on a cycle-by-cycle basis.

The switch is constructed as three independent dimension routers, in six 10-K gate arrays. There is a modest amount of buffering in each switch (eight 16-bit parcels for each of four virtual channels in each of three dimensions), so packets compress into the switches when blocked. There is enough buffering in a switch to store small packets. The input port determines the desired output port by a simple arithmetic operation. The routing distance is decremented, and if the result is nonzero the packet continues in its current direction on the current dimension; otherwise, it is routed into the next dimension. Each output port uses a rotating priority among input ports requesting that output. For each input port, there is a rotating priority for virtual channels requesting that output.

The network interface contains eight packet buffers, two per virtual channel. The entire packet is buffered in the source NI before being transmitted into the network. It is buffered in the destination NI before being delivered to the processor or memory system. This store-and-forward delay effectively decouples the network and node operations. In addition to the main data communication network, separate treelike networks for logical-AND (Barrier) and logical-OR (Eureka) are provided.

The presence of bidirectional links provides two possible options in each dimension. A table lookup is performed in the source network interface to select the (deterministic) route consisting of the direction and distance in each of the three dimensions. An individual program occupies a partition of the machine consisting of a logically contiguous subarray of arbitrary shape (under operating system configuration). Shift and mask logic within the communication assist maps the partition-relative virtual node address into a machinewide logical $\langle X,Y,Z \rangle$ coordinate address. The machine can be configured with spare nodes, which can be brought in to replace a failed node. The $\langle X,Y,Z \rangle$ is used as an index for the route lookup, so the NI routing tables provide the final level of translation, identifying the physical node by its $\langle \pm\Delta x, \pm\Delta y, \pm\Delta z \rangle$ route from the source. This routing lookup also identifies which of the four virtual channels is used. To avoid deadlock within either the request or response (virtual) network, the high channel is used for packets that cross the wraparound links and the low channel otherwise.

10.9.2 IBM SP-1, SP-2 Network

The network used in the IBM SP-1 and SP-2 parallel machines (Abali and Aykanat 1994; Stunkel et al. 1994) is in some ways more versatile than that in the CRAY T3D but of lower performance and without support for two-phase, request-response operations. It is packet switched, with cut-through, source-based routing and no virtual channels. The switch has eight bidirectional 40-MB/s ports and can support a wide variety of topologies. However, in the SP machines, a collection of switches are

Qualcomm Ex. 1015

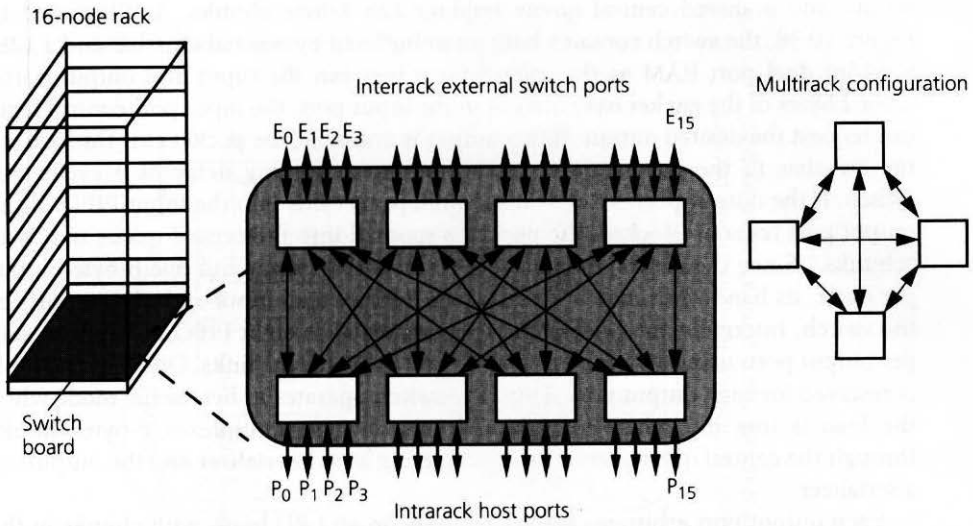


FIGURE 10.37 SP switch packaging. The collection of switches within a rack provides the bidirectional connection between 16 internal ports and 16 external ports.

packaged on a board as a 4-ary 2-dimensional butterfly with 16 internal connections to hosts in the same rack as the switch board and 16 external connections to other racks, as illustrated in Figure 10.37. The rack-level topology varies from machine to machine but is typically a variant of a butterfly. Figure 1.23 in Chapter 1 shows the large IBM SP-2 configuration at the Maui High-Performance Computing Center. Individual cabinets have the first level of routing at the bottom connecting the 16 internal nodes to 16 external links. Additional cabinets provide connectivity between collections of cabinets. The wiring for these additional levels is located beneath the machine room floor. Since the physical topology is not fixed in hardware, the network interface inserts the route for each outgoing message via a table lookup.

Packets consist of a sequence of up to 255 bytes; the first byte is the packet length, followed by one or more routing bytes and then the data bytes. Each routing byte contains two 3-bit output specifiers with an additional selector bit. The links are synchronous, wide, and long. A single 40-MHz clock is distributed to all the switches, with each link tuned so that its delay is an integral number of cycles. (Interboard signals are 100-K ECL differential pairs. Onboard clock trees are also 100-K ECL.) The links consist of 10 wires: 8 data bits, a framing "tag" control bit, and a reverse flow control bit. Thus, the phit is 1 byte. The tag bit identifies the length and routing phits. The flit is 2 bytes; two cycles are used to signal the availability of 2 bytes of storage in the receiver buffer. At any time, a stream of data/tag flits can be propagating down the link while a stream of credit tokens propagate in the other direction.

The switch provides 31 bytes of FIFO buffering on each input port, allowing links to be 16 phits long. In addition, there are 7 bytes of FIFO buffering on each

Qualcomm Ex. 1015

output and a shared central queue holding 128 8-byte chunks. As illustrated in Figure 10.38, the switch contains both an unbuffered byte-serial crossbar and a 128- × 64-bit dual port RAM as the interconnect between the input and output ports. After 2 bytes of the packet have arrived in the input port, the input port control logic can request the desired output. If this output is available, the packet cuts through via the crossbar to the output port, with a minimum routing delay of 5 cycles per switch. If the output port is not available, the packet fills into the input FIFO. If the output port remains blocked, the packet is spooled into the central queue in 8-byte "chunks." Since the central queue accepts one 8-byte input and one 8-byte output per cycle, its bandwidth matches that of the 8-byte serial input and output ports of the switch. Internally, the central queue is organized as eight FIFO linked lists, one per output port, using an auxiliary 128- × 7-bit RAM for the links. One 8-byte chunk is reserved for each output port. Thus, the switch operates in byte-serial mode when the load is low, but when contention forms, it time-multiplexes 8-byte chunks through the central queue, with the inputs acting as a deserializer and the outputs as a serializer.

Each output port arbitrates among requests on an LRU basis, with chunks in the central queue having priority over bytes in input FIFOs. Output ports are served by the central queue in LRU order. The central queue gives priority to inputs with chunks destined for unblocked output ports.

The SP network has three unusual aspects. First, since the operation is globally synchronous, instead of including CRC information in the envelope of each packet, time is divided into 64-cycle "frames." The last two phits of each frame carry the CRC. The input port checks the CRC and the output port generates it (after stripping it of used routing phits). Second, the switch is a single chip, and every switch chip is shadowed by an identical switch. The pins are bidirectional I/O pins, so one of the chips merely checks the operation of the other. (This will detect switch errors but not link errors.) Finally, the switch supports a circuit-switched "service mode" for various diagnostic purposes. The network is drained free of packets before changing modes.

10.9.3 Scalable Coherent Interface

The Scalable Coherent Interface provides a well-specified case study in high-performance interconnects because it emerged through a standards process rather than as a proprietary design or academic proposal. It was a standard long time in coming but has gained popularity as implementations have gotten under way. It has been adopted by several vendors, although in many cases only a portion of the specification is followed. Essentially, the full SCI specification is used in the interconnect of the HP/Convex Exemplar and in the Sequent NUMA-Q. The CRAY SCX I/O network is based heavily on SCI.

A key element of the SCI design is that it builds around the concept of unidirectional rings, called *ringlets*, rather than bidirectional links. Ringlets are connected by switches to form large networks. The specification defines three layers: a physical

Qualcomm Ex. 1015

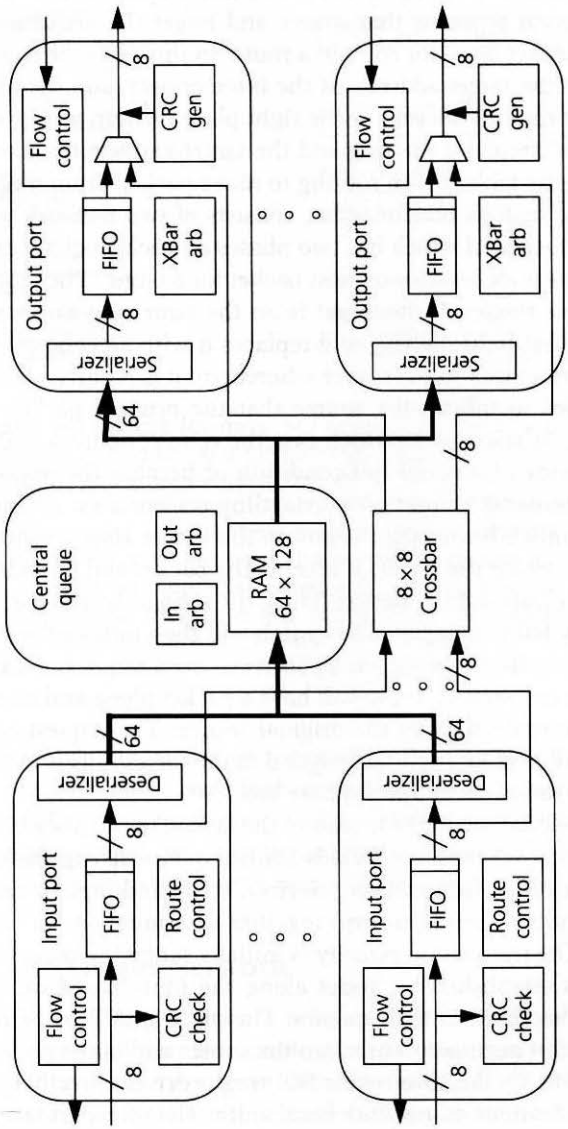


FIGURE 10.38 IBM SP (Vulcan) switch design. The IBM SP switch uses a crossbar for unbuffered packets passed directly from input port to output port, but all buffered packets are shunted into a common memory pool. Arbitration logic schedules flits into and out of the central queue RAM.

Qualcomm Ex. 1015

layer, a packet layer, and a transaction layer. The physical layer is specified in two 1-GB/s forms. Packets consist of a sequence of 16-bit units, much like the CRAY T3D packets. As illustrated in Figure 10.39, all packets consist of a TargetID, Command, and SourceID, followed by zero or more command-specific units and finally a 32-bit CRC. One unusual aspect is that source and target IDs are arbitrary 16-bit node addresses. The packet does not contain a route. In this sense, the design is like a bus: the source puts the target address on the interconnect, and the interconnect determines how to get the information to the right place. Within a ring, this is simple because the packet circulates the ring and the target extracts the packet. In the general case, switches use table-driven routing to move packets from ring to ring.

An SCI transaction, such as read or write, consists of two network transactions (request and response), each of which has two phases on each ringlet. Let's take this step by step. The source node issues a request packet for a target. The request packet circulates on the source ringlet. If the target is on the same ring as the source, the target extracts the request from the ring and replaces it with an echo packet, which continues around the ring back to the source whereupon it is removed from the ring. The echo packet serves to inform the source that the original packet was either accepted by the target, or rejected, in which case the echo contains a NACK. It may be rejected either because of a buffer-full condition or because the packet was corrupted. The source maintains a timer on outstanding packets so it can detect if the echo gets lost or corrupted. If the target is not on the source ring, a switch node on the ring serves as a proxy for the target. It accepts the packet and provides the echo once it has successfully buffered the packet. Thus, the echo only tells the source that the packet successfully left the ringlet. The switch will then initiate the packet onto another ring along the route to the target. Upon receiving a request, the target node will initiate a response transaction. It too will have a packet phase and an echo phase on each ringlet on the route back to the original source. The request echo packet informs the source of the Transaction ID assigned to the target; this is used to match the eventual response, much as on a split-phase bus.

Rather than have a clean envelope for each of the layers, in SCI they blur together a bit in the packet format where several fields control queue management and retry mechanisms. The control tpr (transaction priority), command mpr (maximum ring priority), and command spr (send priority) together determine one of four priority levels for the packet. The transaction priority is initially set by the requestor, but the actual send priority is established by nodes along the route based on what other blocked transactions they have in their queues. The phase and busy fields are used as part of the flow control negotiation between the source and target nodes.

In the Sequent NUMA-Q, the 18-bit-wide SCI ring is driven directly by the DataPump in a quad at 1-GB/s node-to-network bandwidth. The transport layer follows a strict request-reply protocol. When the DataPump puts a packet on the ring, it keeps a copy of the packet in its outgoing buffer until an *echo* is returned. When a DataPump removes an incoming packet from the ring, it replaces it by a *positive echo*. If a DataPump detects a packet destined for it but does not have space to remove that packet from the ring, it sends a *negative echo*, which causes the sender to retry its transmission (still holding the space in its outgoing buffer).

 Qualcomm Ex. 1015

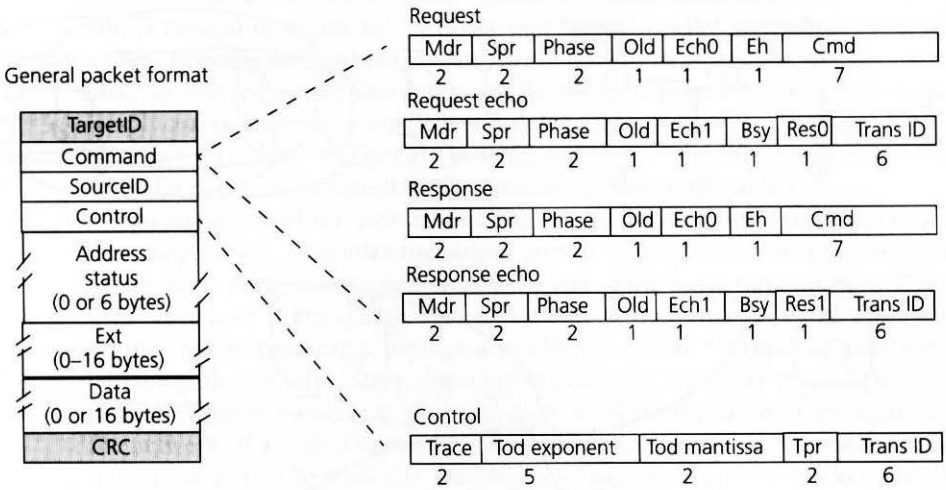


FIGURE 10.39 SCI packet formats. SCI operations involve a pair of transactions: a request and a response. Owing to its ring-based underpinnings, each transaction involves conveying a packet from the source to the target and an echo (going the rest of the way around the ring) from the target back to the source. All packets are a sequence of 16-bit units, with the first three being the destination node, command, and source node, and the final being the CRC. Requests contain a 6-byte address, optional extension, and optional data. Responses have a similar format, but the address bytes carry status information. Both kinds of echo packets contain only the minimal four units. The command unit identifies the packet type through the phase and ech fields. It also describes the operation to be performed (request cmd) or matched (trans id). The remaining fields and the control unit address lower-level issues of how packet queuing and retry are handled at the packet layer.

Since the latency of communication on a ring increases linearly with the number of nodes on it, large high-performance SCI systems are expected to be built out of smaller rings interconnected in arbitrary network topologies. For example, a performance study indicates that a single ring can effectively support four to eight high-performance processing nodes (Scott, Vernon, and Goodman 1992).

10.9.4 SGI Origin Network

The SGI Origin network is based on a flexible switch, called SPIDER, that supports six pairs of unidirectional links, each pair providing over 1.56 GB/s of total bandwidth in the two directions. Two nodes (four processors) are connected to each switch so there are four pairs of links to connect to other switches. This building block is configured in a family of topologies related to hypercubes, as illustrated in Figure 10.40. The links are flexible (long, wide) cables that can be up to 3 meters long. Messages are pipelined through the network, and the latency through the router itself is 41 ns from pin to pin. Routing is table driven, so as part of the network initialization the routing tables are set up in each switch. This allows routing to be programmable so that it is possible to support a range of configurations, to

Qualcomm Ex. 1015

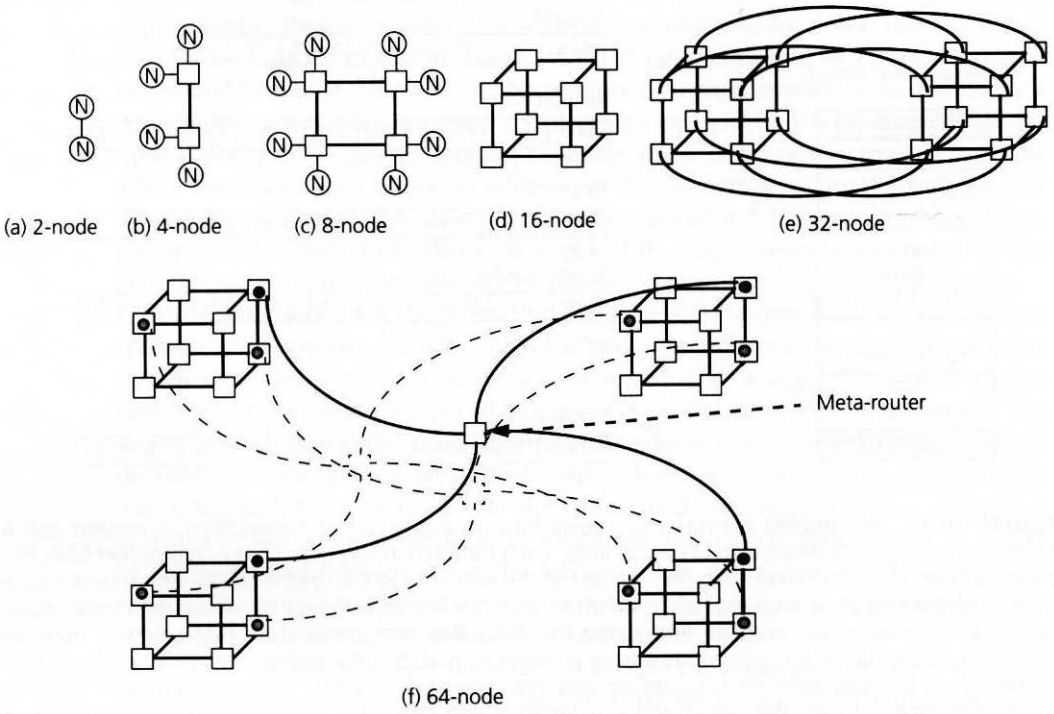


FIGURE 10.40 Network topology and router connections in the SGI Origin multiprocessor. Hypercube topologies are used for up to 32 nodes, as in (a)–(e), and beyond that a fat-tree variant is employed. Configurations (d)–(f) show only the routers and omit the two nodes connected to each for simplicity. Configuration (f) also shows only a few of the fat-cube connections across hypercubes; the routers that connect 16-node subcubes are called meta-routers. For a 512-node (1,024-processor) configuration, each meta-router itself would be replaced by a 5-*d* router hypercube.

have a partially populated network (i.e., not all ports are used), and to route around faulty components. Availability is also aided by the fact that the routers are separately powered and provide per-link error protection with hardware retry upon error. The switch provides separate virtual channels for requests and replies and supports 256 levels of message priority, with packet aging.

10.9.5 Myricom Network

As a final case study, we briefly examine the Myricom network (Boden et al. 1995) used in several cluster systems. The communication assist within the network interface card was described in Section 7.7. Here we are concerned with the switch that is used to construct a scalable interconnection network. Perhaps the most interesting aspect of its design is its simplicity. The basic building block is a switch with eight bidirectional ports of 160 MB/s each. The physical link is a 9-bit-wide long wire. It

Qualcomm Ex. 1015

can be up to 25 m, and 23 phits can be in transit simultaneously. The encoding allows control flow, framing symbols, and "gap" (or idle) symbols to be interleaved with data symbols. Symbols are constantly transmitted between switches across the links so that the switch can determine which of its ports are connected. A packet is simply a sequence of routing bytes followed by a sequence of payload bytes and a CRC byte. The end of a packet is delimited by the presence of a gap symbol. The start of a packet is indicated by the presence of a nongap symbol.

The operation of the switch is extremely simple: it removes the first byte from an incoming packet, computes the output port by adding the contents of the route byte to the input port number, and spools the rest of the packet to that port. The switch uses wormhole routing with a small amount of buffering for each input and each output. The routing delay through a switch is less than 500 ns. The switches can be wired together in an arbitrary topology. It is the responsibility of the host communication software to construct routes that are valid for the physical interconnection of the network. If a packet attempts to exit a switch on an invalid or unconnected port, it is discarded. When all the routing bytes are used up, the packet should have arrived at an NIC. The first byte of the message has a bit to indicate that it is not a routing byte, and the remaining bits indicate the packet type. All higher-level packet formatting and transactions are realized by the NIC and the host; the interconnection network only moves the bits.

10.10 CONCLUDING REMARKS

Parallel computer networks present a rich and diverse design space that brings together several levels of design. The physical link-level issues represent some of the most interesting electrical engineering aspects of computer design. In a constant effort to keep pace with the increasing rate of processors, link rates are ever improving. Currently, we are seeing multigigabit rates on copper pairs, and parallel fiber technologies offer the possibility of multigigabyte rates per link in the near future. One of the major issues at these rates is dealing with errors. If bit error rates of the physical medium are in the order of 10^{-19} per bit and data is being transmitted at a rate of 10^9 bytes per second, then an error is likely to occur on a link roughly every 10 minutes. With thousands of links in the machine, errors occur every second. These must be detected and corrected rapidly.

The switch-to-switch layer of design also offers a rich set of trade-offs, including how aspects of the problem are pushed down into the physical layer and how they are pushed up into the packet layer. For example, flow control can be built into the exchange of digital symbols across a link, or it can be addressed at the next layer in terms of packets that cross the link or even one layer higher, in terms of messages sent from end to end. There is a huge space of alternatives for the design of the switch itself, and these are again driven by engineering constraints from below and design requirements from above.

Even the basic topology, switching strategy, and routing algorithm reflect a compromise of engineering requirements and design requirements from below and above. We have seen, for example, how a basic question like the degree of the switch

depends heavily on the cost model associated with the target technology and the characteristics of the communication pattern in the target workload. Finally, as with many other aspects of architecture, there is significant room for debate as to how much of the higher-level semantics of the node-to-node communication abstraction should be embedded in the hardware design of the network itself. The entire area of network design for parallel computers is bound to be exciting for many years to come. In particular, there is a strong flow of ideas and technology between parallel computer networks and advancing, scalable networks for local area and system area communication.

10.11 EXERCISES

10.1 Consider a packet format with 10 bytes of routing and control information and 6 bytes of CRC and other trailer information. The payload contains a 64-byte cache block, along with 8 bytes of command and address information. If the raw link bandwidth is 500 MB/s, what is the effective data bandwidth on cache block transfers using this format? How would this change with 32-byte cache blocks? 128-byte blocks? 4-KB page transfers?

10.2 Suppose the links are 1 byte wide and operating at 300 MHz in a network where the average routing distance between nodes is $\log_4 P$ for P nodes. Compare the unloaded latency for 80-byte packets under store-and-forward and cut-through routing, assuming 4 cycles of delay per hop to make the routing decision and P ranging from 16 to 1,024 nodes.

10.3 Perform the comparison in Exercise 10.2 for 32-KB transfers fragmented into 1-KB packets.

10.4 Find an optimal fragmentation strategy for an 8-KB message, as is common for page-sized transfers under the following assumptions. There is a fixed overhead of o cycles per fragment, there is a store-and-forward delay through the source and the destination NI, packets cut through the network with a routing delay of R cycles, and the limiting data transfer rate is b bytes per cycle.

10.5 Suppose an $n \times n$ matrix of double-precision numbers is laid out over P nodes by rows. In order to compute on the columns you desire to transpose it to have a column layout. How much data will cross the bisection during this operation?

10.6 Consider a 2D torus direct network of N nodes with a link bandwidth of b bytes per second. Calculate the bisection bandwidth and the average routing distance. Compare the estimate of aggregate communication bandwidth using Equation 10.6 with the estimate based on bisection. In addition, suppose that every node communicates only with nodes 2 hops away. Then what bandwidth is available? What if each node communicates only with nodes in its row?

10.7 Show how an N-node torus is embedded in an N-node hypercube such that neighbors in the torus (i.e., nodes of distance one) are neighbors in the hypercube. [Hint: observe what happens when the addresses are ordered in a graycode sequence.] Generalize this embedding for higher-dimensional meshes.

Qualcomm Ex. 1015

10.8 Perform the analysis of Equation 10.6 for a hypercube network. In the last step, treat the row as being defined by the graycode mapping of the grid into the hypercube.

10.9 Calculate the average distance for a linear array of N nodes (assume N is even) and for a 2D mesh and 2D torus of N nodes.

10.10 A more accurate estimate of the communication time can be obtained by determining the average number of channels traversed per packet, h_{ave}, for the workload of interest and the particular network topology. The effective aggregate bandwidth is at most

$$\frac{\|C\|}{h_{ave}} B$$

Suppose the orchestration of a program treats the processors as a $\sqrt{p} \times \sqrt{p}$ logical mesh where each node communicates n bytes of data with its eight neighbors in the four directions and on the diagonal. Give an estimate of the communication time on a grid and a hypercube.

10.11 Show how an N-node tree can be embedded in an N-node hypercube by stretching one of the tree edges across two hypercube edges.

10.12 Use a spreadsheet to construct a comparison of Figure 10.12 with a design point of 10 cycle routing delays and 16-bit-wide links. Extend this comparision for 1,024-byte messages. What conclusions can you draw?

10.13 Verify that the minimum latency is achieved under equal pin scaling in Figure 10.14 when the routing delay is equal to the channel time.

10.14 Derive a formula for dimension that achieves the minimum latency under equal bisection scaling based on Equation 10.7.

10.15 Under the equal bisection scaling rule, how wide are the links of a 2D mesh with 1 million nodes?

10.16 Compare the behavior under load of 2D and 3D cubes of roughly equal size, as in Figure 10.17, for equal pin and equal bisection scaling.

10.17 Specify the Boolean logic used to compute the routing function in the switch with Δx, Δy routing on a 2D mesh.

10.18 If Δx, Δy routing is used, what effect does decrementing the count in the header have on the CRC in the trailer? How can this issue be addressed in the switch design?

10.19 Specify the Boolean logic used to compute the routing function in the switch dimension order routing on a hypercube.

10.20 Prove that the e-cube routing in a hypercube is deadlock-free.

10.21 Construct the table-based equivalent of Δx, Δy routing in a 2D mesh.

10.22 Show that permitting the pair of turns not allowed in Figure 10.24 leaves complex cycles. [Hint: they look like a figure eight.]

10.23 Show that with two virtual channels, arbitrary routing in a bidirectional network can be made deadlock-free.

Qualcomm Ex. 1015

10.24 Pick any flow control scheme in the chapter and compute the fraction of bandwidth used by flow control symbols.

10.25 Revise latency estimates of Figure 10.14 for stacked dimension switches.

10.26 Table 10.1 provides the topologies and an estimate of the basic communication performance characteristics for several important designs. For each, calculate the network latency for 40-byte and 160-byte messages on 16- and 1,024-node configurations. What fraction of this is routing delay and what fraction is occupancy?

Qualcomm Ex. 1015