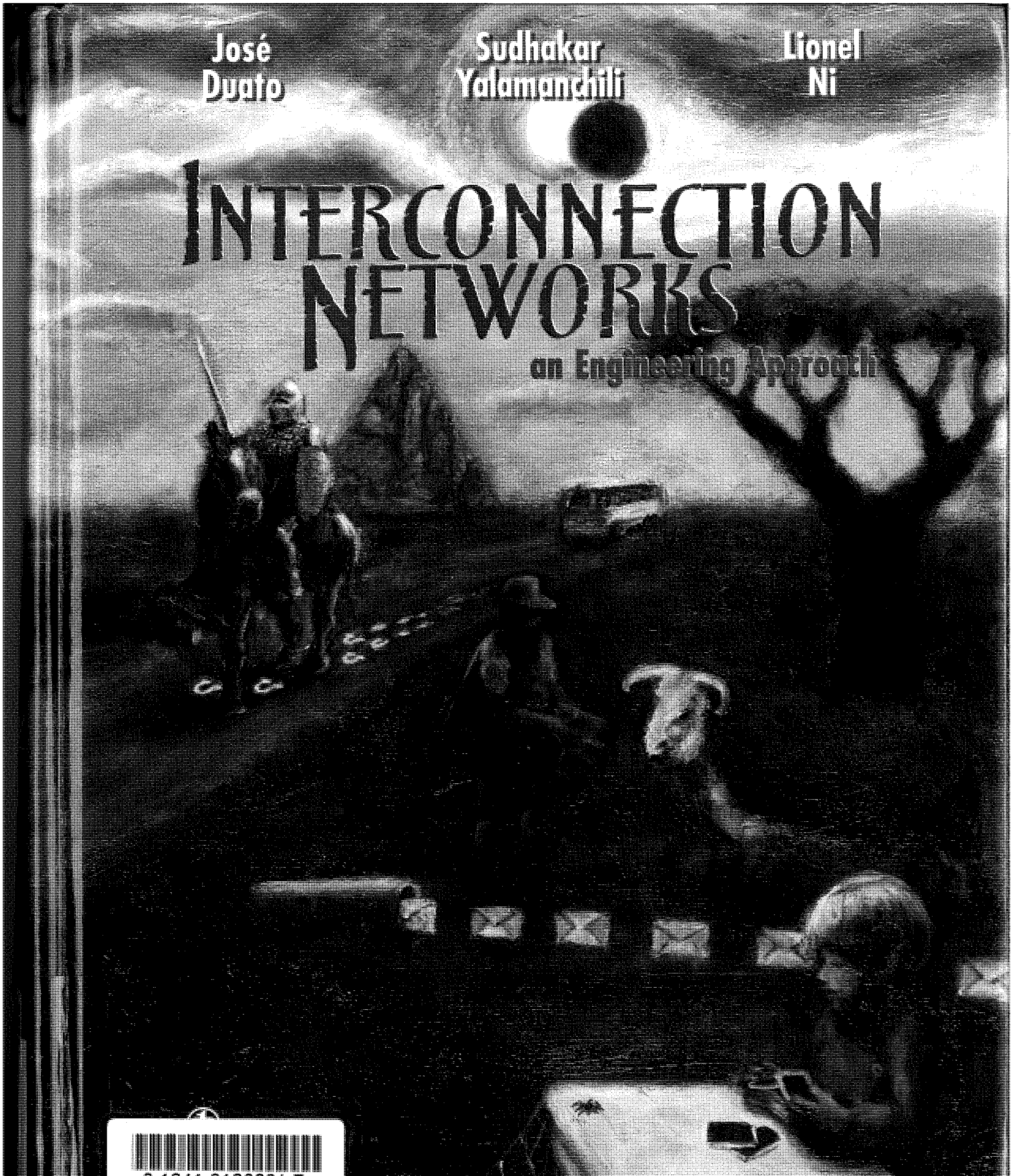
José Duato Sudhakar Yalamanchili Lionel Ni

INTERCONNECTION NETWORKS

an Engineering Approach



0 1341 0136201 7

1

APPLE 1007

Interconnection Networks

An Engineering Approach

José Duato
Sudhakar Yalamanchili
Lionel Ni

130101

IEEE COMPUTER SOCIETY
Los Alamitos, California

Washington • Brussels • Tokyo

Library of Congress Cataloging-in-Publication Data

Duato, José
Interconnection networks: an engineering approach / José Duato, Sudhakar Yalamanchili, Lionel Ni.
p. cm.
Includes bibliographical references.
ISBN 0-8186-7800-3
1. Computer networks. 2. Multiprocessors. I. Yalamanchili. II. Ni, Lionel M. III. Title.
TK5105.5.D88 1997
004.6—dc21 97-20502
CIP

IEEE Computer Society Press Order Number BP07800
Library of Congress Number 97-20502
ISBN 0-8186-7800-3

Additional copies may be ordered from:

IEEE Computer Society Press	IEEE Service Center	IEEE Computer Society	IEEE Computer Society
Customer Service Center	445 Hoes Lane	13, Avenue de l'Aquilon	Ooshima Building
10662 Los Vaqueros Circle	P.O. Box 1331	B-1200 Brussels	2-19-1 Minami-Aoyama
P.O. Box 3014	Piscataway, NJ 08855-1331	BELGIUM	Minato-ku, Tokyo 107
Los Alamitos, CA 90720-1314	Tel: +1-908-981-1393	Tel: +32-2-770-2198	JAPAN
Tel: +1-714-821-8380	Fax: +1-908-981-9667	Fax: +32-2-770-8505	Tel: +81-3-3408-3118
Fax: +1-714-821-4641	mis.custserv@computer.org	euro.ofc@computer.org	Fax: +81-3-3408-3553
Email: cs.books@computer.org			tokyo.ofc@computer.org

Editor-in-Chief: Mohamed Fayad
Publisher: Matt Loeb
Acquisitions Editor: Bill Sanders
Developmental Editor: Cheryl Baltes
Advertising/Promotions: Tom Fink
Production Editor: Lisa O'Conner
Cover Artist: John Marshall

About the cover: *Interconnection Networks* by John Marshall, oil on canvass, 1997, is based on a concept by José Duato. The painting includes symbols representing important elements in the field. To find out more about the symbols in the painting, visit the book's website at http://computer.org/books.

Printed in the United States of America

IEEE
COMPUTER
SOCIETY
IEEE

Chapter 1

Introduction

Interconnection networks are currently being used for many different applications, ranging from internal buses in very large-scale integration (VLSI) circuits to wide area computer networks. Among others, these applications include backplane buses and system area networks, telephone switches, internal networks for asynchronous transfer mode (ATM) switches, processor/memory interconnects for vector supercomputers, interconnection networks for multicomputers and distributed shared-memory multiprocessors, clusters of workstations, local area networks, metropolitan area networks, wide area computer networks, and networks for industrial applications. Additionally, the number of applications requiring interconnection networks is continuously growing. For example, an integral control system for a car requires a network connecting several microprocessors and devices.

The characteristics and cost of these networks considerably depend on the application. There are no general solutions. For some applications, interconnection networks have been studied in depth for decades. This is the case for telephone networks, computer networks, and backplane buses. These networks are covered in many books. However, there are some other applications that have not been fully covered in the existing literature. This is the case for the interconnection networks used in multicomputers and distributed shared-memory multiprocessors.

The lack of standards and the need for very high performance and reliability pushed the development of interconnection networks for multicomputers. This technology was transferred to distributed shared-memory multiprocessors, improving the scalability of those machines. However, distributed shared-memory multiprocessors require an even higher network performance than multicomputers, pushing the development of interconnection networks even more. More recently, this network technology began to be transferred to local area networks (LANs). Also, it has been proposed as a replacement for backplane buses, creating the concept of system area network. Hence, the advances in interconnection networks for multicomputers are the basis for the development of interconnection networks for other architectures and environments. Therefore, there is a need for structuring the concepts and solutions for this kind of interconnection networks. Obviously, when this technology is transferred to another environment, new issues arise that have to be addressed.

Moreover, several of these networks are evolving very quickly, and the solutions proposed for different kinds of networks are overlapping. Thus, there is a need for formally stating the basic concepts, the alternative design choices, and the design trade-offs for most of those networks. In this book, we take that challenge and present in a structured way the basic underlying concepts of most interconnection networks, as well as the most interesting solutions currently implemented or proposed in the literature. As indicated above, the network technology developed for multicomputers has been transferred to other environments. Therefore, in this book we will mainly describe techniques developed for multicomputer networks. Most of these techniques can also be applied to distributed shared-memory multiprocessors, and to local and system area networks. However, we will also describe techniques specifically developed for these environments.

1.1 Parallel Computing and Networks

The demand for even more computing power has never stopped. Although the performance of processors has doubled in approximately every three-year span from 1980 to 1996, the complexity of the software as well as the scale and solution quality of applications have continuously driven the development of even faster processors. A number of important problems have been identified in the areas of defense, aerospace, automotive applications, and science, whose solution requires tremendous amount of computational power. In order to solve these grand challenge problems, the goal has been to obtain computer systems capable of computing at the teraflops (10^{12} floating-point operations per second) level. Even the smallest of these problems requires gigaflops (10^9 floating-point operations per second) of performance for hours at a time. The largest problems require teraflops performance for more than a thousand hours at a time.

Parallel computers with multiple processors are opening the door to teraflops computing performance to meet the increasing demand of computational power. The demand includes more computing power, higher network and input/output (I/O) bandwidths, and more memory and storage capacity. Even for applications requiring a lower computing power, parallel computers can be a cost-effective solution. Processors are becoming very complex. As a consequence, processor design cost is growing so fast that only a few companies all over the world can afford to design a new processor. Moreover, design cost should be amortized by selling a very high number of units. Currently, personal computers and workstations dominate the computing market. Therefore, designing custom processors that boost the performance one order of magnitude is not cost-effective. Similarly, designing and manufacturing high-speed memories and disks is not cost-effective. The alternative choice consists of designing parallel computers from commodity components (processors, memories, disks, interconnects, etc.). In these parallel computers, several processors cooperate to solve a large problem. Memory bandwidth can be scaled with processor computing power by physically distributing memory components among processors. Also, redundant arrays of inexpensive disks (RAID) allow the implementation of high-capacity reliable parallel file systems meeting the performance requirements of parallel computers.

However, a parallel computer requires some kind of communication subsystems to interconnect processors, memories, disks and other peripherals. The specific requirements of these communication subsystems depend on the architecture of the parallel computer. The simplest solution consists of connecting processors to memories and disks as if there were a single processor, using system buses and I/O buses. Then, processors can be interconnected using the interfaces to local area networks. Unfortunately, commodity communication subsystems have been designed to meet a different set of requirements, i.e., those arising in computer networks. Although networks of workstations have been proposed as an inexpensive approach to build parallel computers, the communication subsystem becomes the bottleneck in most applications.

Therefore, designing high-performance interconnection networks becomes a critical issue to exploit the performance of parallel computers. Moreover, as the interconnection network is the only subsystem that cannot be efficiently implemented by using commodity components, its design becomes very critical. This issue motivated the writing of this book. Up to now, most manufacturers designed custom interconnection networks (nCUBE-2, nCUBE-3, Intel Paragon, Cray T3D, Cray T3E, Thinking Machines Corp. CM-5, NEC Cenju-3, IBM SP2). More recently, several high-

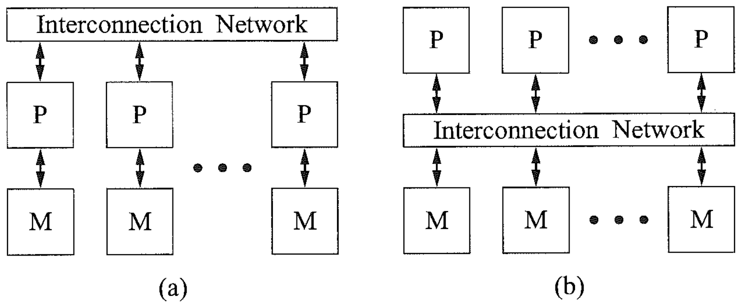


Figure 1.1. Schematic representation of parallel computers: (a) A multicomputer. (b) A UMA shared-memory multiprocessor. (M = memory; P = processor.)

performance switches have been developed (Autonet, Myrinet, ServerNet) and are being marketed. These switches are targeted to workstations and personal computers, offering the customer the possibility of building an inexpensive parallel computer by connecting cost-effective computers through high-performance switches. The main issues arising in the design of networks for both approaches are covered in this book.

1.2 Parallel Computer Architectures

In this section, we briefly introduce the most popular parallel computer architectures. This description will focus on the role of the interconnection network. A more detailed description is beyond the scope of this book.

The idea of using commodity components for the design of parallel computers led to the development of *distributed-memory multiprocessors*, or *multicomputers* in early 1980s. These parallel computers consist of a set of processors, each one connected to its own local memory. Processors communicate between them by passing messages through an interconnection network. Figure 1.1a shows a simple scheme for this architecture. The first commercial multicomputers utilized commodity components, including Ethernet controllers to implement communication between processors. Unfortunately, commodity communication subsystems were too slow, and the interconnection network became the bottleneck of those parallel computers. Several research efforts led to the development of interconnection networks that are several orders of magnitude faster than Ethernet networks. Most of the performance gain is due to architectural rather than technological improvements.

Programming multicomputers is not an easy task. The programmer has to take care of distributing code and data among the processors in an efficient way, invoking message-passing calls whenever some data are needed by other processors. On the other hand, *shared-memory multiprocessors* provide a single memory space to all the processors, simplifying the task of exchanging data among processors. Access to shared memory has been traditionally implemented by using an interconnection network between processors and memory (Figure 1.1b). This architecture is referred to as *uniform memory access* (UMA) architecture. It is not scalable because memory access time includes the latency of the interconnection network, and this latency increases with system size.

More recently, shared-memory multiprocessors followed some trends previously established for multicomputers. In particular, memory has been physically distributed among processors, therefore reducing the memory access time for local accesses and increasing scalability. These parallel computers are referred to as *distributed shared-memory multiprocessors* (DSM). Accesses to remote memory are performed through an interconnection network, very much like in multicomputers. The main difference between DSMs and multicomputers is that messages are initiated by memory accesses rather than by calling a system function. In order to reduce memory latency, each processor has several levels of cache memory, thus matching the speed of processors and memories. This architecture provides *nonuniform memory access* (NUMA) time. Indeed, most of the nonuniformity is due to the different access time between caches and main memories, rather than the different access time between local and remote memories. The main problem arising in DSMs is cache coherence. Several hardware and software cache coherence protocols have been proposed. These protocols produce additional traffic through the interconnection network.

The use of custom interconnects makes multicomputers and DSMs quite expensive. So, *networks of workstations* (NOW) have been proposed as an inexpensive approach to build parallel computers. NOWs take advantage of recent developments in LANs. In particular, the use of ATM switches has been proposed to implement NOWs. However, ATM switches are still expensive, which has motivated the development of high-performance switches, specifically designed to provide a cost-effective interconnect for workstations and personal computers.

Although there are many similarities between interconnection networks for multicomputers and DSMs, it is important to keep in mind that performance requirements may be very different. Messages are usually very short when DSMs are used. Additionally, network latency is important because memory access time depends on that latency. However, messages are typically longer and less frequent when using multicomputers. Usually the programmer is able to adjust the granularity of message communication in a multicomputer. On the other hand, interconnection networks for multicomputers and NOWs are mainly used for message passing. However, the geographical distribution of workstations usually imposes constraints on the way processors are connected. Also, individual processors may be connected to or disconnected from the network at any time, thus imposing additional design constraints.

1.3 Network Design Considerations

Interconnection networks play a major role in the performance of modern parallel computers. There are many factors that may affect the choice of an appropriate interconnection network for the underlying parallel computing platform. These factors include:

1. *Performance requirements.* Processes executing in different processors synchronize and communicate through the interconnection network. These operations are usually performed by explicit message passing or by accessing shared variables. Message *latency* is the time elapsed between the time a message is generated at its source node and the time the message is delivered at its destination node. Message latency directly affects processor idle time and memory access time to remote memory locations. Also, the network may *saturate* — it may be unable to deliver the flow of messages injected by the nodes, limiting the effective computing power of a parallel computer. The maximum amount of information delivered by the network per time unit defines the *throughput* of that network.

2. *Scalability.* A scalable architecture implies that as more processors are added, their memory bandwidth, I/O bandwidth, and network bandwidth should increase proportionally. Otherwise the components whose bandwidth does not scale may become a bottleneck for the rest of the system, decreasing the overall efficiency accordingly.

3. *Incremental expandability.* Customers are unlikely to purchase a parallel computer with a full set of processors and memories. As the budget permits, more processors and memories may be

added until a system's maximum configuration is reached. In some interconnection networks, the number of processors must be a power of 2, which makes them difficult to expand. In other cases, expandability is provided at the cost of wasting resources. For example, a network designed for a maximum size of 1,024 nodes may contain many unused communication links when the network is implemented with a smaller size. Interconnection networks should provide incremental expandability, allowing the addition of a small number of nodes while minimizing resource wasting.

4. *Partitionability.* Parallel computers are usually shared by several users at a time. In this case, it is desirable that the network traffic produced by each user does not affect the performance of other applications. This can be ensured if the network can be partitioned into smaller functional subsystems. Partitionability may also be required for security reasons.

5. *Simplicity.* Simple designs often lead to higher clock frequencies and may achieve higher performance. Additionally, customers appreciate networks that are easy to understand because it is easier to exploit their performance.

6. *Distance span.* This factor may lead to very different implementations. In multicomputers and DSMs, the network is assembled inside a few cabinets. The maximum distance between nodes is small. As a consequence, signals are usually transmitted using copper wires. These wires can be arranged regularly, reducing the computer size and wire length. In NOWs, links have very different lengths and some links may be very long, producing problems such as coupling, electromagnetic noise, and heavy link cables. The use of optical links solves these problems, equalizing the bandwidth of short and long links up to a much greater distance than when copper wire is used. Also, geographical constraints may impose the use of irregular connection patterns between nodes, making distributed control more difficult to implement.

7. *Physical constraints.* An interconnection network connects processors, memories, and/or I/O devices. It is desirable for a network to accommodate a large number of components while maintaining a low communication latency. As the number of components increases, the number of wires needed to interconnect them also increases. Packaging these components together usually requires meeting certain physical constraints, such as operating temperature control, wiring length limitation, and space limitation. Two major implementation problems in large networks are the arrangement of wires in a limited area, and the number of pins per chip (or board) dedicated to communication channels. In other words, the complexity of the connection is limited by the maximum wire density possible, and by the maximum pin count. The speed at which a machine can run is limited by the wire lengths, and the majority of the power consumed by the system is used to drive the wires. This is an important and challenging issue to be considered. Different engineering technologies for packaging, wiring, and maintenance should be considered.

8. *Reliability and repairability.* An interconnection network should be able to deliver information reliably. Interconnection networks can be designed for continuous operation in the presence of a limited number of faults. These networks are able to send messages through alternative paths when some faults are detected. In addition to reliability, interconnection networks should have a modular design, allowing hot upgrades and repairs. Nodes can also fail or be removed from the network. In particular, a node can be powered off in a network of workstations. Thus, NOWs usually require some reconfiguration algorithm for the automatic reconfiguration of the network when a node is powered on or off.

9. *Expected workloads.* Users of a general-purpose machine may have very different requirements. If the kind of applications that will be executed in the parallel computer are known in advance, it may be possible to extract some information on usual communication patterns, message sizes, network load, etc. That information can be used for the optimization of some design parameters. When it is not possible to get information on expected workloads, network design should be robust, i.e., design parameters should be selected in such a way that performance is good over a wide range of traffic conditions.

10. *Cost constraints.* Finally, it is obvious that the "best" network may be too expensive. Design decisions very often are trade-offs between cost and other design factors. Fortunately, cost is not always directly proportional to performance. Using commodity components whenever possible may considerably reduce the overall cost.

1.4 Classification of Interconnection Networks

Among other criteria, interconnection networks have been traditionally classified according to the operating mode (synchronous or asynchronous), and network control (centralized, decentralized, or distributed). Nowadays, multicomputers, multiprocessors, and NOWs dominate the parallel computing market. All of these architectures implement asynchronous networks with distributed control. Therefore, we will focus on other criteria that are currently more significant.

A classification scheme is shown in Figure 1.2 which categorizes the known interconnection networks into four major classes based primarily on network topology: shared-medium networks, direct networks, indirect networks, and hybrid networks. For each class, the figure shows a hierarchy of subclasses, also indicating some real implementations for most of them. This classification scheme is based on the classification proposed in [252], and it mainly focuses on networks that have been implemented. It is by no means complete as other new and innovative interconnection networks may emerge as technology further advances, such as mobile communication and optical interconnections.

In *shared-medium networks*, the transmission medium is shared by all communicating devices. An alternative to this approach consists of having point-to-point links directly connecting each communicating device to a (usually small) subset of other communicating devices in the network. In this case, any communication between nonneighboring devices requires transmitting the information through several intermediate devices. These networks are known as *direct networks.* Instead of directly connecting the communicating devices between them, *indirect networks* connect those devices by means of one or more switches. If several switches exist, they are connected between them using point-to-point links. In this case, any communication between communicating devices requires transmitting the information through one or more switches. Finally, *hybrid* approaches are possible. These network classes and the corresponding subclasses will be described in the following sections.

1.5 Shared-Medium Networks

The least complex interconnect structure is one in which the transmission medium is shared by all communicating devices. In such *shared-medium networks*, only one device is allowed to use the network at a time. Every device attached to the network has requester, driver, and receiver circuits to handle the passing of address and data. The network itself is usually passive, since the network itself does not generate messages.

Interconnection Networks

- Shared-Medium Networks
  - Local Area Networks
    - Contention Bus (Ethernet)
    - Token Bus (Arcnet)
    - Token Ring (FDDI Ring, IBM Token Ring)
  - Backplane Bus (Sun Gigaplane, DEC AlphaServer8X00, SGI PowerPath-2)
- Direct Networks (Router-Based Networks)
  - Strictly Orthogonal Topologies
    - Mesh
      - 2-D Mesh (Intel Paragon)
      - 3-D Mesh (MIT J-Machine)
    - Torus (k-ary n-cube)
      - 1-D Unidirectional Torus or Ring (KSR first-level ring)
      - 2-D Bidirectional Torus (Intel/CMU iWarp)
      - 3-D Bidirectional Torus (Cray T3D, Cray T3E)
    - Hypercube (Intel iPSC, nCUBE)
  - Other Topologies: Trees, Cube-Connected Cycles, de Bruijn Network, Star Graphs, etc.
- Indirect Networks (Switch-Based Networks)
  - Regular Topologies
    - Crossbar (Cray X/Y-MP, DEC GIGAswitch, Myrinet)
    - Multistage Interconnection Networks
      - Blocking Networks
        - Unidirectional MIN (NEC Cenju-3, IBM RP3)
        - Bidirectional MIN (IBM SP, TMC CM-5, Meiko CS-2)
      - Nonblocking Networks: Clos Network
  - Irregular Topologies (DEC Autonet, Myrinet, ServerNet)
- Hybrid Networks
  - Multiple-Backplane Buses (Sun XDBus)
  - Hierarchical Networks (Bridged LANs, KSR)
    - Cluster-Based Networks (Stanford DASH, HP/Convex Exemplar)
  - Other Hypergraph Topologies: Hyperbuses, Hypermeshes, etc.

Figure 1.2. Classification of interconnection networks. (1-D = one-dimensional; 2-D = two-dimensional; 3-D = three-dimensional; CMU = Carnegie Mellon University; DASH = Directory Architecture for Shared-Memory; DEC = Digital Equipment Corp.; FDDI = Fiber Distributed Data Interface; HP = Hewlett-Packard; KSR = Kendall Square Research; MIN = Multistage Interconnection Network; MIT = Massachusetts Institute of Technology; SGI = Silicon Graphics Inc.; TMC = Thinking Machines Corp.)

An important issue here is the *arbitration strategy* that determines the mastership of the shared-medium network to resolve network access conflicts. A unique characteristic of a shared medium is its ability to support atomic *broadcast* in which all devices on the medium can monitor network activities and receive the information transmitted on the shared medium. This property is important to efficiently support many applications requiring one-to-all or one-to-many communication services, such as barrier synchronization and snoopy cache coherence protocols. Due to limited network bandwidth, a single shared medium can only support limited number of devices before the medium becomes a bottleneck.

Shared-medium networks constitute a well established technology. Additionally, their limited bandwidth restricts their use in multiprocessors. So, these networks will not be covered in this book, but we will present a short introduction in the following sections. There are two major classes of shared-medium networks: local area networks, mainly used to construct computer networks that span physical distances no longer than a few kilometers, and backplane buses, mainly used for internal communication in uniprocessors and multiprocessors.

1.5.1 Shared-Medium Local Area Networks

High-speed LANs can be used as a networking backbone to interconnect computers to provide an integrated parallel and distributed computing environment. Physically, a shared-medium LAN uses copper wires or fiber optics in a bit-serial fashion as the transmission medium. The network topology is either a bus or a ring. Depending on the arbitration mechanism used, different LANs have been commercially available. For performance and implementation reasons, it is impractical to have a centralized control or to have some fixed access assignment to determine the bus master who can access the bus. Three major classes of LANs based on distributed control are described below.

Contention Bus

The most popular bus arbitration mechanism is to have all devices to compete for the exclusive access right of the bus. Due to the broadcast nature of the bus, all devices can monitor the state of the bus, such as idle, busy, and collision. Here the term "collision" means that two or more devices are using the bus at the same time and their data collided. When the collision is detected, the competing devices will quit transmission and try later. The most well-known contention-based LAN is Ethernet which adopts carrier-sense multiple access with collision detection (CSMA/CD) protocol. The bandwidth of Ethernet is 10 Mbps and the distance span is 250 meters (coaxial cable). As processors are getting faster, the number of devices that can be connected to Ethernet is limited to avoid the network bottleneck. In order to break the 10 Mbps bandwidth barrier, Fast Ethernet can provide 100 Mbps bandwidth.

Token Bus

One drawback of the contention bus is its nondeterministic nature as there is no guarantee of how much waiting time is required to gain the bus access right. Thus, the contention bus is not suitable to support real-time applications. To remove the nondeterministic behavior, an alternate approach involves passing a token among the network devices. The owner of the token has the right to access the bus. Upon completion of the transmission, the token is passed to the next device based on some scheduling discipline. By restricting the maximum token holding time, the upper bound that a device has to wait for the token can be guaranteed. Arcnet supports token bus with a bandwidth of 2.5 Mbps.

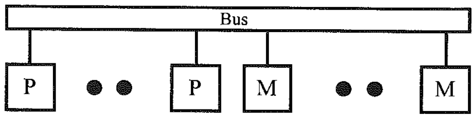


Figure 1.3. A single-bus network. (M = memory; P = processor.)

Token Ring

The idea of token ring is a natural extension of token bus as the passing of the token forms a ring structure. IBM token ring supports bandwidths of both 4 and 16 Mbps based on coaxial cable. Fiber Distributed Data Interface (FDDI) provides a bandwidth of 100 Mbps using fiber optics.

1.5.2 Shared-Medium Backplane Bus

A *backplane bus* is the simplest interconnection structure for bus-based parallel computers. It is commonly used to interconnect processor(s) and memory modules to provide UMA architecture. Figure 1.3 shows a single-bus network. A typical backplane bus usually has 50 – 300 wires and is physically realized by printed lines on a circuit board or by discrete (backplane) wiring. Additional costs are incurred by interface electronics, such as line drivers, receivers, and connectors.

There are three kinds of information in the backplane bus: data, address, and control signals. Control signals include bus request signal and request grant signal, among many others. In addition of the width of data lines, the maximum bus bandwidth that can be provided is dependent on the technology. The number of processors that can be put on a bus depends on many factors, such as processor speed, bus bandwidth, cache architecture, and program behavior.

Methods of Information Transfer

Both data and address information must be carried in the bus. In order to increase the bus bandwidth and provide a large address space, both data width and address bits have to be increased. Such an increase implies another increase in the bus complexity and cost. Some designs try to share address and data lines. For *multiplexed transfer*, address and data are sent alternatively. Hence, they can share the same physical lines and require less power and fewer chips. For *nonmultiplexed transfer*, address and data lines are separated. Thus, data transfer can be done faster.

In *synchronous bus* design, all devices are synchronized with a common clock. It requires less complicated logic and has been used in most existing buses. However, a synchronous bus is not easily upgradable. New faster processors are difficult to fit into a slow bus.

In *asynchronous buses*, all devices connected to the bus may have different speeds and their own clocks. They use a handshaking protocol to synchronize with each other. This provides independence for different technologies and allows slower and faster devices with different clock rates to operate together. This also implies buffering is needed, since slower devices cannot handle messages as fast as faster devices.

Bus Arbitration

In a single-bus network, several processors may attempt to use the bus simultaneously. To deal with this, a policy must be implemented that allocates the bus to the processors making such requests. For performance reasons, bus allocation must be carried out by hardware arbiters. Thus, in order to perform a memory access request, the processor has to exclusively own the bus and become the *bus* *master*. To become the bus master, each processor implements a *bus requester*, which is a collection of logic to request control of the data transfer bus. On gaining control, the requester notifies the requesting master.

Two alternative strategies are used to release the bus:

- Release-when-done: release the bus when data transfer is done
- Release-on-request: hold the bus until another processor requests it

Several different *bus arbitration algorithms* have been proposed, which can be classified into *centralized* or *distributed*. A centralized method has a central *bus arbiter*. When a processor wants to become the bus master, it sends out a bus request to the bus arbiter which then sends out a request grant signal to the requesting processor. A bus arbiter can be an encoder-decoder pair in hardware design. In distributed method, such as daisy chain method, there is no central bus arbiter. The bus request signals form a daisy chain. The mastership is released to the next device when data transfer is done.

Split Transaction Protocol

Most bus transactions involve request and response. This is the case for memory read operations. After a request is issued, it is desirable to have a fast response. If a fast response time is expected, the bus mastership is not released after sending the request, and data can be received soon. However, due to memory latency, the bus bandwidth is wasted while waiting for a response. In order to minimize the waste of bus bandwidth, the *split transaction protocol* has been used in many bus networks.

In this protocol, the bus mastership is released immediately after the request, and the memory has to gain mastership before it can send the data. Split transaction protocol has a better bus utilization but its control unit is much more complicated. *Buffering* is needed in order to save messages before the device can gain the bus mastership.

To support shared-variable communication, some atomic read/modify/write operations to memories are needed. With the split transaction protocol, the atomic read/modify/write can no longer be indivisible. One approach to solve this problem is to disallow bus release for those atomic operations.

Bus Examples

Several examples of buses and the main characteristics are listed below.

- Gigaplane used in Sun Ultra Enterprise X000 Server (ca. 1996): 2.6 Gbyte/s peak, 256 bits data, 42 bits address, split-transaction protocol, 83.8 MHz clock.
- DEC AlphaServer8X00, i.e., 8200 and 8400 (ca. 1995): 2.1 Gbyte/s, 256 bits data, 40 bits address, split-transaction protocol, 100 MHz clock (1 foot length).
- SGI PowerPath-2 (ca. 1993): 1.2 Gbyte/s, 256 bits data, 40 bits address, 6 bits control, split-transaction protocol, 47.5 MHz clock (1 foot length).
- HP9000 Multiprocessor Processor Memory Bus (ca. 1993): one Gbyte/s, 128 bits data, 64 bits address, 13 inches, pipelined-bus, 60 MHz clock.

1.6 Direct Networks

Scalability is an important issue in designing multiprocessor systems. Bus-based systems are not scalable as the bus becomes the bottleneck when more processors are added. The *direct network* or *point-to-point network* is a popular network architecture that scales well to a large number of processors. A direct network consists of a set of *nodes*, each one being directly connected to a (usually small) subset of other nodes in the network. Figures 1.5 through 1.7 show several direct networks. The corresponding interconnection patterns between nodes will be studied below. Each node is a programmable computer with its own processor, local memory, and other supporting devices. These nodes may have different functional capabilities. For example, the set of nodes may contain vector processors, graphics processors, and I/O processors. Figure 1.4 shows the architecture of a generic node. A common component of these nodes is a *router*, which handles message communication among nodes. For this reason, direct networks are also known as router-based networks. Each router has direct connections to the router of its neighbors. Usually, two neighboring nodes are connected by a pair of unidirectional channels in opposite directions. A bidirectional channel may also be used to connect two neighboring nodes. Although the function of a router can be performed by the local processor, dedicated routers have been used in high-performance multicomputers, allowing overlapped computation and communication within each node. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system also increase. Thus, direct networks have been a popular interconnection architecture for constructing large-scale parallel computers.

Each router supports some number of input and output channels. *Internal* channels or *ports* connect the local processor/memory to the router. Although it is common to provide only one pair of internal channels, some systems use more internal channels in order to avoid a communication bottleneck between the local processor/memory and the router [39]. *External* channels are used for communication between routers. By connecting input channels of one node to the output channels of other nodes, the direct network is defined. Unless otherwise specified, the term "channel" will refer to an external channel. Two directly connected nodes are called *neighboring* or *adjacent* nodes. Usually, each node has a fixed number of input and output channels, and every input channel is paired with a corresponding output channel. Through the connections among these channels, there are many ways to interconnect these nodes. Obviously, every node in the network should be able to reach every other node.

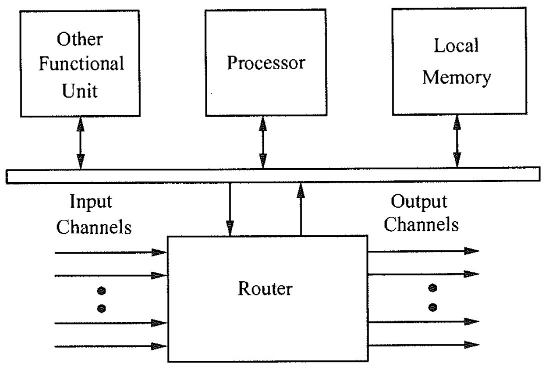


Figure 1.4. A generic node architecture.

1.6.1 Characterization of Direct Networks

Direct networks have been traditionally modeled by a graph $G(N, C)$, where the vertices of the graph N represent the set of processing nodes, and the edges of the graph C represent the set of communication channels. This is a very simple model that does not consider implementation issues. However, it allows the study of many interesting network properties. Depending on the properties under study, a bidirectional channel may be modeled either as an edge or as two arcs in opposite directions (two unidirectional channels). The latter is the case for deadlock avoidance in Chapter 3. Let us assume that a bidirectional channel is modeled as an edge. Some basic network properties can be defined from the graph representation:

- *Node degree:* Number of channels connecting that node to its neighbors.
- *Diameter:* The maximum distance between two nodes in the network.
- *Regularity:* A network is *regular* when all the nodes have the same degree.
- *Symmetry:* A network is *symmetric* when it looks alike from every node.

A direct network is mainly characterized by three factors: topology, routing, and switching. The *topology* defines how the nodes are interconnected by channels, and is usually modeled by a graph as indicated above. For direct networks, the ideal topology would connect every node to every other node. No message would even have to pass through an intermediate node before reaching its destination. This fully connected topology requires a router with N links (including the internal one) at each node for a network with N nodes. Therefore, the cost is prohibitive for networks of moderate to large size. Additionally, the number of physical connections of a node is limited by hardware constraints such as the number of available pins and the available wiring area. These engineering and scaling difficulties preclude the use of such fully connected networks even for small network sizes. As a consequence, many topologies have been proposed, trying to balance performance and some cost parameters. In these topologies, messages may have to traverse some intermediate nodes before reaching the destination node.

From the programmer's perspective, the unit of information exchange is the *message*. The size of messages may vary depending on the application. For efficient and fair use of network resources, a message is often divided into packets prior to transmission. A *packet* is the smallest unit of communication that contains the destination address and sequencing information, which are carried in the packet *header*. For topologies in which packets may have to traverse some intermediate nodes, the *routing* algorithm determines the path selected by a packet to reach its destination. At each intermediate node, the routing algorithm indicates the next channel to be used. That channel may be selected among a set of possible choices. If all the candidate channels are busy, the packet is blocked and cannot advance. Obviously, efficient routing is critical to the performance of interconnection networks.

When a message or packet header reaches an intermediate node, a *switching* mechanism determines how and when the router switch is set, i.e., the input channel is connected to the output channel selected by the routing algorithm. In other words, the switching mechanism determines how network resources are allocated for message transmission. For example, in circuit switching, all the channels required by a message are reserved before starting message transmission. In packet switching, however, a packet is transmitted through a channel as soon as that channel is reserved but the next channel is not reserved (assuming that it is available) until the packet releases the channel it is currently using. Obviously, some buffer space is required to store the packet until the next channel is reserved. That buffer should be allocated before starting packet transmission. So, *buffer allocation* is closely related to the switching mechanism. Flow control is also closely related to the switching and buffer allocation mechanisms. The *flow control* mechanism establishes a dialog between sender and receiver nodes, allowing and stopping the advance of information. If a packet is blocked, it requires some buffer space to be stored. When there is no more available buffer space, the flow control mechanism stops information transmission. When the packet advances and buffer space is available, transmission is started again. If there is no flow control and no more buffer space is available, the packet may be dropped, or derouted through another channel.

The above factors affect the network performance. They are not independent of each other but are closely related. For example, if a switching mechanism reserves resources in an aggressive way (as soon as a packet header is received), packet latency can be reduced. However, each packet may be holding several channels at the same time. So, such a switching mechanism may cause severe network congestion and, consequently, make the design of efficient routing and flow control policies difficult. The network topology also affects performance, as well as how the network traffic can be distributed over available channels. In most cases, the choice of a suitable network topology is restricted by wiring and packaging constraints.

1.6.2 Popular Network Topologies

Many network topologies have been proposed in terms of their graph theoretic properties. However, very few of them have ever been implemented. Most of the implemented networks have an orthogonal topology. A network topology is *orthogonal* if and only if nodes can be arranged in an orthogonal n-dimensional space, and every link can be arranged in such a way that it produces a displacement in a single dimension. Orthogonal topologies can be further classified as strictly orthogonal and weakly orthogonal. In a *strictly orthogonal* topology, every node has at least one link crossing each dimension. In a *weakly orthogonal* topology, some nodes may not have any link in some dimensions. Hence, it is not possible to cross every dimension from every node. Crossing a given dimension from a given node may require moving in another dimension first.

Strictly Orthogonal Topologies

The most interesting property of strictly orthogonal topologies is that routing is very simple. Thus, the routing algorithm can be efficiently implemented in hardware. Effectively, in a strictly orthogonal topology nodes can be numbered by using their coordinates in the n-dimensional space. As each link traverses a single dimension and every node has at least one link crossing each dimension, the distance between two nodes can be computed as the sum of dimension offsets. Also, the displacement along a given link only modifies the offset in the corresponding dimension. Taking into account that it is possible to cross any dimension from any node in the network, routing can be easily implemented by selecting a link that decrements the absolute value of the offset in some dimension. The set of dimension offsets can be stored in the packet header, and updated (by adding or subtracting one unit) every time the packet is successfully routed at some intermediate node. If the topology is not strictly orthogonal, however, routing may become much more complex.

The most popular direct networks are the *n-dimensional mesh*, the *k-ary n-cube* or *torus*, and the *hypercube*. All of them are strictly orthogonal. Formally, an n-dimensional mesh has $k_0 \times k_1 \times \cdots \times k_{n-2} \times k_{n-1}$ nodes, k_i nodes along each dimension i, where $k_i \geq 2$ and $0 \leq i \leq n-1$. Each node X is identified by n coordinates, $(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, where $0 \leq x_i \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes X and Y are neighbors if and only if $y_i = x_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $y_j = x_j \pm 1$. Thus, nodes have from n to $2n$ neighbors, depending on their location in the mesh. Therefore, this topology is not regular.

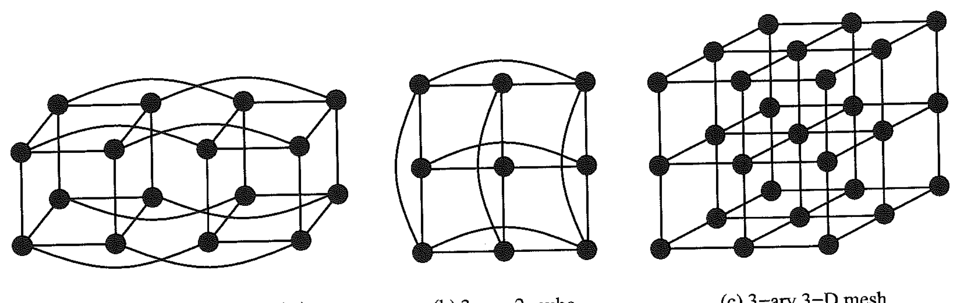


Figure 1.5. Strictly orthogonal direct network topologies.

In a bidirectional k-ary n-cube [71], all nodes have the same number of neighbors. The definition of a k-ary n-cube differs from that of an n-dimensional mesh in that all of the k_i are equal to k and two nodes X and Y are neighbors if and only if $y_i = x_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $y_j = (x_j \pm 1) \bmod k$. The change to modular arithmetic in the definition adds wraparound channels to the k-ary n-cube, giving it regularity and symmetry. Every node has n neighbors if $k = 2$ and $2n$ neighbors if $k > 2$. When $n = 1$, the k-ary n-cube collapses to a bidirectional *ring* with k nodes.

Another topology with regularity and symmetry is the hypercube, which is a special case of both n-dimensional meshes and k-ary n-cubes. A hypercube is an n-dimensional mesh in which $k_i = 2$ for $0 \leq i \leq n-1$, or a 2-ary n-cube, also referred to as a binary n-cube.

Figure 1.5a depicts a binary 4-cube or 16-node hypercube. Figure 1.5b illustrates a 3-ary 2-cube or two-dimensional (2-D) torus. Figure 1.5c shows a 3-ary three-dimensional (3-D) mesh, resulting by removing the wraparound channels from a 3-ary 3-cube.

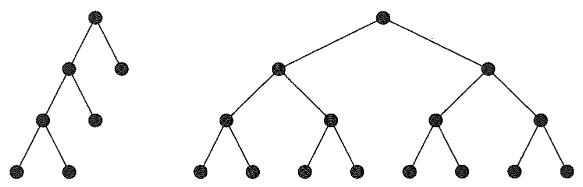
Two conflicting requirements of a direct network are that it must accommodate a large number of nodes while maintaining a low network latency. This issue will be addressed in Chapter 7.

Other Direct Network Topologies

In addition to the topologies defined above, many other topologies have been proposed in the literature. Most of them were proposed with the goal of minimizing the network diameter for a given number of nodes and node degree. As will be seen in Chapter 2, for pipelined switching techniques network latency is almost insensitive to network diameter, especially when messages are long. So it is unlikely that those topologies are implemented. In the following paragraphs, we present an informal description of some relevant direct network topologies.

A popular topology is the *tree*. This topology has a *root* node connected to a certain number of descendant nodes. Each of these nodes is in turn connected to a disjoint set (possibly empty) of descendants. A node with no descendants is a *leaf* node. A characteristic property of trees is that every node but the root has a single parent node. Therefore, trees contain no cycles. A tree in which every node but the leaves has a fixed number k of descendants is a k-ary tree. When the distance from every leaf node to the root is the same, i.e., all the branches of the tree have the same length, the tree is *balanced*. Figures 1.6a and 1.6b show an unbalanced and a balanced binary tree, respectively.

The most important drawback of trees as general-purpose interconnection networks is that the root node and the nodes close to it become a bottleneck. Additionally, there are no alternative paths between any pair of nodes. The bottleneck can be removed by allocating a higher channel



(a) Binary Tree (b) Balanced Binary Tree

Figure 1.6. Some tree topologies.

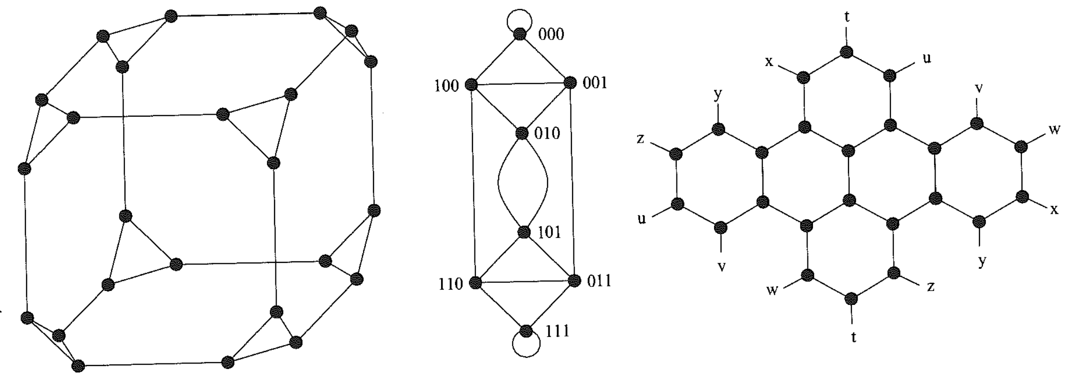
bandwidth to channels located close to the root node. The shorter the distance to the root node, the higher the channel bandwidth. However, using channels with different bandwidth is not practical, specially when message transmission is pipelined. A practical way to implement trees with higher channel bandwidth in the vicinity of the root node (fat trees) will be described in Section 1.7.5.

One of the most interesting properties of trees is that, for any connected graph, it is possible to define a tree that spans the complete graph. As a consequence, for any connected network, it is possible to build an acyclic network connecting all the nodes by removing some links. This property can be used to define a routing algorithm for any irregular topology. However, that routing algorithm may be inefficient due to the concentration of traffic across the root node. A possible way to circumvent that limitation will be presented in Section 4.9.

Some topologies have been proposed with the purpose of reducing node degree while keeping the diameter small. Most of these topologies can be viewed as a hierarchy of topologies. This is the case for the cube-connected cycles [283]. This topology can be considered as an n-dimensional hypercube of virtual nodes, where each virtual node is a ring with n nodes, for a total of $n2^n$ nodes. Each node in the ring is connected to a single dimension of the hypercube. Therefore, node degree is fixed and equal to three: two links connecting to neighbors in the ring, and one link connecting to a node in another ring through one of the dimensions of the hypercube. However, the diameter is of the same order of magnitude as that of a hypercube of similar size. Figure 1.7a shows a 24-node cube-connected cycles network. It is worth to note that cube-connected cycles are weakly orthogonal because the ring is a one-dimensional network, and displacement inside the ring does not change the position in the other dimensions. Similarly, a displacement along a hypercube dimension does not affect the position in the ring. However, it is not possible to cross every dimension from each node.

Many topologies have been proposed with the purpose of minimizing the network diameter for a given number of nodes and node degree. Two well-known topologies proposed with this purpose are the de Bruijn network and the star graphs. In the *de Bruijn* network [300] there are d^n nodes, and each node is represented by a set of n digits in base d. A node $(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, where $0 \leq x_i \leq d-1$ for $0 \leq i \leq n-1$ is connected to nodes $(x_{n-2}, \ldots, x_1, x_0, p)$ and $(p, x_{n-1}, x_{n-2}, \ldots, x_1)$, for all p such that $0 \leq p \leq d-1$. In other words, two nodes are connected if the representation of one node is a right or left shift of the representation of the other. Figure 1.7b shows an eight-node de Bruijn network. When networks are very large, this network topology achieves a very low diameter for a given number of nodes and node degree. However, routing is complex. Additionally, the average distance between nodes is high, close to the network diameter. Finally, some nodes have links connecting to themselves. All of these issues make the practical application of these networks very difficult.

A *star graph* [6] can be informally described as follows. The vertices of the graph are labeled by permutations of n different symbols, usually denoted as 1 to n. A permutation is connected to every other permutation that can be obtained from it by interchanging the first symbol with any



(a) Cube-Connected Cycles (b) De Bruijn Network (c) Star Graph

Figure 1.7. Other direct network topologies.

of the other symbols. A star graph has $n!$ nodes and node degree is equal to $n - 1$. Figure 1.7c shows a star graph obtained by permutations of four symbols. Although a star graph has a lower diameter than a hypercube of similar size, routing is more complex. Star graphs were proposed as a particular case of Cayley graphs [6]. Other topologies like hypercubes and cube-connected cycles are also particular cases of Cayley graphs.

1.6.3 Examples

Several examples of parallel computers with direct networks and the main characteristics are listed below.

- Cray T3E: Bidirectional 3-D torus, 14-bit data in each direction, 375 MHz link speed, 600 Mbytes/s per link.
- Cray T3D: Bidirectional 3-D torus, up to 1,024 nodes ($8 \times 16 \times 8$), 24-bit links (16-bit data, 8-bit control), 150 MHz, 300 Mbytes/s per link.
- Intel Cavallino: Bidirectional 3-D topology, 16-bit-wide channels operating at 200 MHz, 400 Mbytes/s in each direction.
- SGI SPIDER: Router with 20-bit bidirectional channels operating on both edges, 200 MHz clock, aggregate raw data rate of 1 Gbyte/s. Support for regular and irregular topologies.
- MIT M-Machine: 3-D mesh, 800 Mbytes/s for each network channel.
- MIT Reliable Router: 2-D mesh, 23-bit links (16-bit data), 200 MHz, 400 Mbytes/s per link per direction, bidirectional signaling, reliable transmission.
- Chaos Router: 2-D torus topology, bidirectional 8-bit links, 180 MHz, 360 Mbytes/s in each direction.
- Intel iPSC-2 Hypercube: Binary hypercube topology, bit-serial channels at 2.8 Mbytes/s.

1.7 Indirect Networks

Indirect or *switch-based networks* are another major class of interconnection networks. Instead of providing a direct connection among some nodes, the communication between any two nodes has to be carried through some *switches*. Each node has a network adapter that connects to a network switch. Each switch can have a set of *ports*. Each port consists of one input and one output link. A (possibly empty) set of ports in each switch are either connected to processors or left open, whereas the remaining ports are connected to ports of other switches to provide connectivity between the processors. The interconnection of those switches defines various network topologies.

Switch-based networks considerably evolved over time. A wide range of topologies have been proposed, ranging from regular topologies used in array processors and shared-memory UMA multiprocessors to the irregular topologies currently used in NOWs. Both network classes will be covered in this book. Regular topologies have regular connection patterns between switches while irregular topologies do not follow any predefined pattern. Figures 1.19 and 1.21 show several switch-based networks with regular topology. The corresponding connection patterns will be studied below. Figure 1.8 shows a typical switch-based network with irregular topology. Both network classes can be further classified according to the number of switches a message has to traverse before reaching its destination. Although this classification is not important in the case of irregular topologies, it makes a big difference in the case of regular networks because some specific properties can be derived for each network class.

1.7.1 Characterization of Indirect Networks

Indirect networks can also be modeled by a graph $G(N, C)$, where N is the set of switches, and C is the set of unidirectional or bidirectional links between the switches. For the analysis of most properties, it is not necessary to explicitly include processing nodes in the graph. Although a similar model can be used for direct and indirect networks, a few differences exist between them. Each switch in an indirect network may be connected to zero, one, or more processors. Obviously, only the switches connected to some processor can be the source or the destination of a message. Additionally, transmitting a message from a node to another node requires crossing the link between

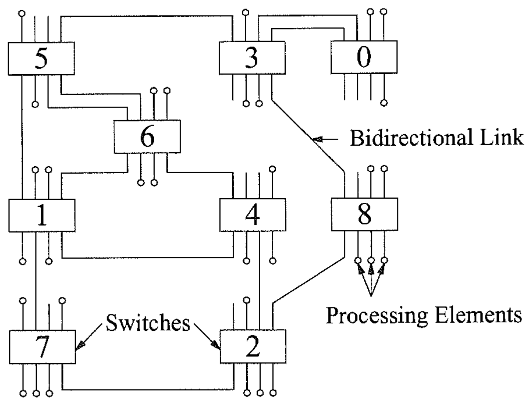


Figure 1.8. A switch-based network with irregular topology.

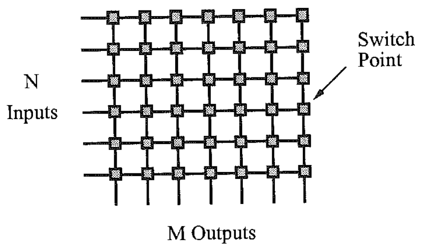


Figure 1.9. An $N \times M$ crossbar.

the source node and the switch connected to it, and the link between the last switch in the path and the destination node. Therefore, the distance between two nodes is the distance between the switches directly connected to those nodes plus two units. Similarly, the diameter is the maximum distance between two switches connected to some node plus two units. It may be argued that it is not necessary to add two units because direct networks also have internal links between routers and processing nodes. However, those links are external in the case of indirect networks. This gives a consistent view of the diameter as the maximum number of external links between two processing nodes. In particular, the distance between two nodes connected through a single switch is two instead of zero.

Similar to direct networks, an indirect network is mainly characterized by three factors: topology, routing, and switching. The topology defines how the switches are interconnected by channels, and can be modeled by a graph as indicated above. For indirect networks with N nodes, the ideal topology would connect those nodes through a single $N \times N$ switch. Such a switch is known as a *crossbar*. Although using a single $N \times N$ crossbar is much cheaper than using a fully connected direct network topology (requiring N routers, each one having an internal $N \times N$ crossbar), the cost is still prohibitive for large networks. Similar to direct networks, the number of physical connections of a switch is limited by hardware constraints such as the number of available pins and the available wiring area. These engineering and scaling difficulties preclude the use of crossbar networks for large network sizes. As a consequence, many alternative topologies have been proposed. In these topologies, messages may have to traverse several switches before reaching the destination node. In regular networks, these switches are usually identical and have been traditionally organized as a set of *stages*. Each stage (but the input/output stages) is only connected to the previous and next stages using regular connection patterns. Input/output stages are connected to the nodes as well as to another stage in the network. These networks are referred to as *multistage* networks, and have different properties depending on the number of stages, and how those stages are arranged.

The remaining issues discussed in Section 1.6.1 (routing, switching, flow control, buffer allocation, and their impact on performance) are also applicable to indirect networks.

1.7.2 Crossbar Networks

Crossbar networks allow any processor in the system to connect to any other processor or memory unit so that many processors can communicate simultaneously without contention. A new connection can be established at any time as long as the requested input and output ports are free. Crossbar networks are used in the design of high-performance small-scale multiprocessors, in the design of routers for direct networks, and as basic components in the design of large-scale indirect networks. A crossbar can be defined as a switching network with N inputs and M outputs, which allows up to

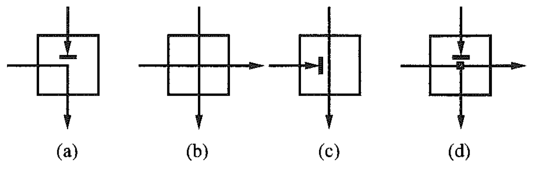


Figure 1.10. States of a switch point in a crossbar network.

min$\{N, M\}$ one-to-one interconnections without contention. Figure 1.9 shows an $N \times M$ crossbar network. Usually, $M = N$ except for crossbars connecting processors and memory modules.

The cost of such a network is $O(NM)$, which is prohibitively high with large N and M. Crossbar networks have been traditionally used in small-scale shared-memory multiprocessors, where all processors are allowed to access memories simultaneously as long as each processor reads from, or writes to, a different memory. When two or more processors contend for the same memory module, arbitration lets one processor proceed while the others wait. The arbiter in a crossbar is distributed among all the switch points connected to the same output. However, the arbitration scheme can be less complex than the one for a bus because conflicts in crossbar are the exception rather than the rule, and therefore easier to resolve.

For a crossbar network with distributed control, each switch point may have four states as shown in Figure 1.10. In Figure 1.10a, the input from the row containing the switch point has been granted access to the corresponding output while inputs from upper rows requesting the same output are blocked. In Figure 1.10b, an input from an upper row has been granted access to the output. The input from the row containing the switch point does not request that output, and can be propagated to other switches. In Figure 1.10c, an input from an upper row has also been granted access to the output. However, the input from the row containing the switch point also requests that output and is blocked. The configuration in Figure 1.10d is only required if the crossbar has to support multicasting (one-to-many communication).

The advent of VLSI permitted the integration of hardware for thousands of switches into a single chip. However, the number of pins on a VLSI chip cannot exceed a few hundreds, which restricts the size of the largest crossbar that can be integrated into a single VLSI chip. Large crossbars can be realized by partitioning them into smaller crossbars, each one implemented using a single chip. Thus, a full crossbar of size $N \times N$ can be implemented with $(N/n)(N/n)$ $n \times n$ crossbars.

1.7.3 Multistage Interconnection Networks

Multistage interconnection networks (MINs) connect input devices to output devices through a number of switch stages, where each switch is a crossbar network. The number of stages and the connection patterns between stages determine the routing capability of the networks.

MINs were initially proposed for telephone networks and later for array processors. In these cases, a central controller establishes the path from input to output. In cases where the number of inputs equals the number of outputs, each input synchronously transmits a message to one output, and each output receives a message from exactly one input. Such unicast communication patterns can be represented as a permutation of the input addresses. For this application, MINs have been popular as alignment networks for storing and accessing arrays in parallel from memory banks. Array storage is typically skewed to permit conflict-free access, and the network is used to unscramble the arrays during access. These networks can also be configured with the number of inputs greater than the number of outputs (concentrators) and vice versa (expanders). On the other hand, in asynchronous

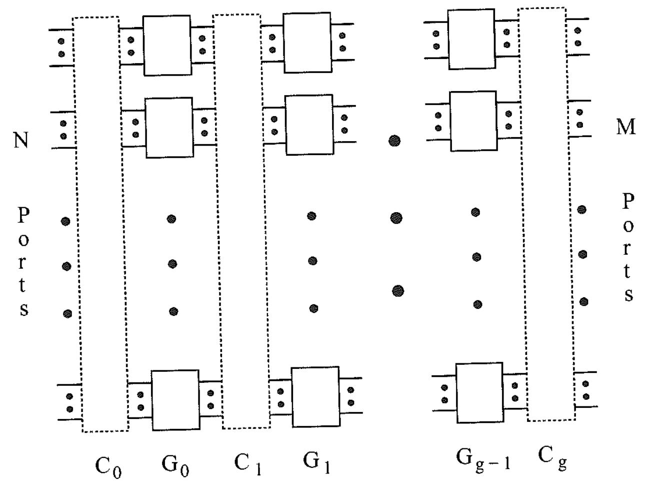


Figure 1.11. A generalized MIN with N inputs, M outputs, and g stages.

multiprocessors, centralized control and permutation routing are infeasible. In this case, a routing algorithm is required to establish the path across the stages of a MIN.

Depending on the interconnection scheme employed between two adjacent stages and the number of stages, various MINs have been proposed. MINs are good for constructing parallel computers with hundreds of processors and have been used in some commercial machines.

1.7.4 A Generalized MIN Model

There are many ways to interconnect adjacent stages. Figure 1.11 shows a generalized multistage interconnection network with N inputs and M outputs. It has g *stages*, G_0 to G_{g-1}. As shown in Figure 1.12, each stage, say G_i, has w_i switches of size $a_{i,j} \times b_{i,j}$, where $1 \leq j \leq w_i$. Thus, stage G_i has p_i inputs and q_i outputs, where

$$p_i = \sum_{j=1}^{w_i} a_{i,j} \quad \text{and} \quad q_i = \sum_{j=1}^{w_i} b_{i,j}$$

The connection between two adjacent stages, G_{i-1} and G_i, denoted C_i, defines the *connection pattern* for $p_i = q_{i-1}$ links, where $p_0 = N$ and $q_{g-1} = M$. A MIN thus can be represented as

$$C_0(N)G_0(w_0)C_1(p_1)G_1(w_1)\ldots G_{g-1}(w_{g-1})C_g(M)$$

A connection pattern $C_i(p_i)$ defines how those p_i links should be connected between the $q_{i-1} = p_i$ outputs from stage G_{i-1} and the p_i inputs to stage G_i. Different connection patterns give different characteristics and topological properties of MINs. The links are labeled from 0 to $p_i - 1$ at C_i.

From a practical point of view, it is interesting that all the switches are identical, thus amortizing the design cost. Banyan networks are a class of MINs with the property that there is a unique path between any pair of source and destination [133]. An N-node ($N = k^n$) Delta network is a subclass of banyan networks, which is constructed from identical $k \times k$ switches in n stages, where each stage contains $\frac{N}{k}$ switches. Many of the known MINs, such as Omega, flip, cube, butterfly, and baseline,

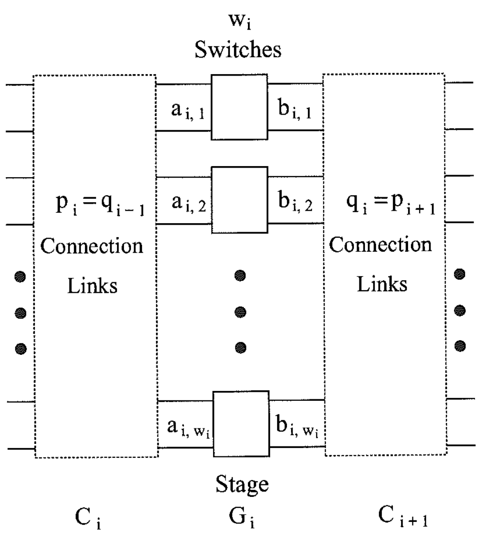


Figure 1.12. A closer view of stage G_i.

belong to the class of Delta networks [272] and have been shown to be topologically and functionally equivalent [351]. A good survey of those MINs can be found in [317]. Some of those MINs are defined below.

When switches have the same number of input and output ports, MINs also have the same number of input and output ports. Since there is a one-to-one correspondence between inputs and outputs, these connections are also called *permutations*. Five basic permutations are defined below. Although these permutations were originally defined for networks with 2×2 switches, for most definitions we assume that the network is built by using $k \times k$ switches, and that there are $N = k^n$ inputs and outputs, where n is an integer. However, some of these permutations are only defined for the case where N is a power of 2. With $N = k^n$ ports, let $X = x_{n-1}x_{n-2}\dots x_0$ be an arbitrary port number, $0 \le X \le N-1$, where $0 \le x_i \le k-1$, $0 \le i \le n-1$.

Perfect Shuffle Connection

The *perfect k-shuffle* connection σ^k is defined by

$$\sigma^k(X) = (kX + \left\lfloor \frac{kX}{N} \right\rfloor) \bmod N$$

A more cogent way to describe the perfect k-shuffle connection σ^k is

$$\sigma^k(x_{n-1}x_{n-2}\dots x_1x_0) = x_{n-2}\dots x_1x_0x_{n-1}$$

The perfect k-shuffle connection performs a cyclic shifting of the digits in X to the left for one position. For $k = 2$, this action corresponds to perfectly shuffling a deck of N cards, as demonstrated in Figure 1.13a for the case of $N = 8$. The perfect shuffle cuts the deck into two halves from the center and intermixes them evenly. The *inverse perfect shuffle* does the opposite as defined below:

$$\sigma^{k^{-1}}(x_{n-1}x_{n-2}\dots x_1x_0) = x_0x_{n-1}\dots x_2x_1$$

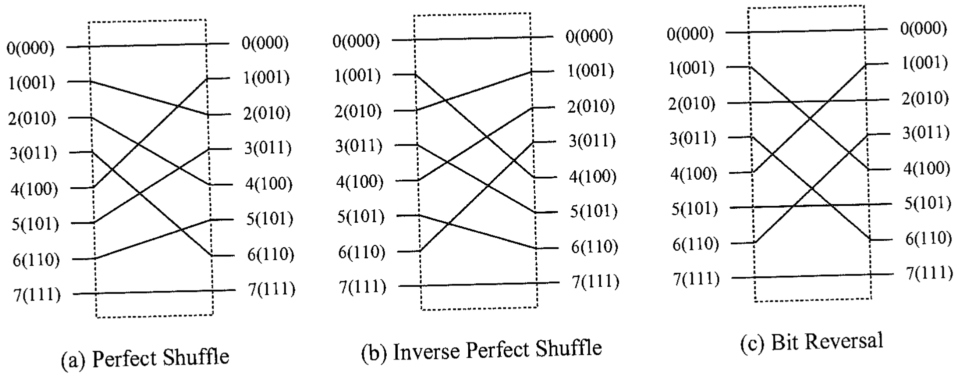


Figure 1.13. The perfect shuffle, inverse perfect shuffle, and bit reversal connections for $N = 8$.

Digit Reversal Connection

The *digit reversal* permutation ρ^k is defined by

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

This permutation is usually referred to as *bit reversal*, clearly indicating that it was proposed for $k = 2$. However, its definition is also valid for $k > 2$. Figure 1.13c demonstrates a bit reversal connection for the case of $k = 2$ and $N = 8$.

Butterfly Connection

The ith k-ary *butterfly* permutation β_i^k, for $0 \le i \le n-1$, is defined by

$$\beta_i^k(x_{n-1}\cdots x_{i+1}x_ix_{i-1}\cdots x_1x_0) = x_{n-1}\cdots x_{i+1}x_0x_{i-1}\cdots x_1x_i$$

The ith butterfly connection interchanges the zeroth and ith digits of the index. Figure 1.14 shows the butterfly connection for $k = 2$, and $i = 0$, 1, and 2 with $N = 8$. Note that β_0^k defines a straight one-to-one connection and is also called *identity connection*, I.

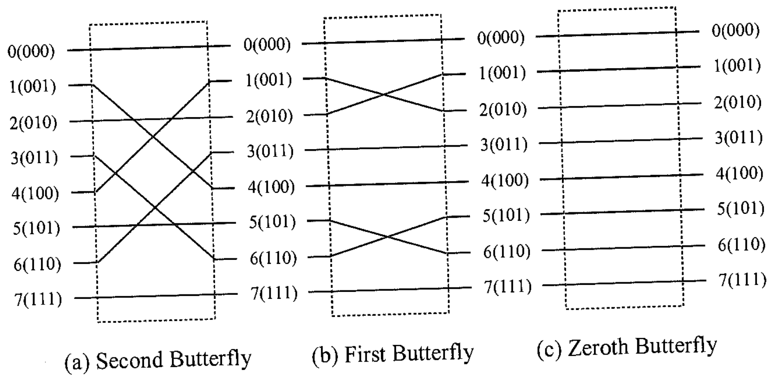


Figure 1.14. The butterfly connection for $N = 8$.

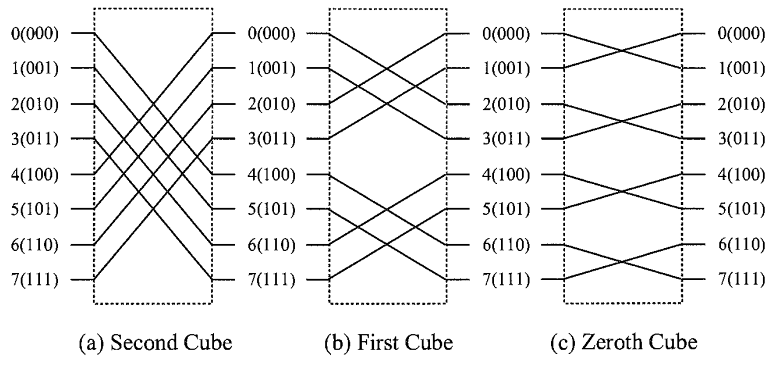


Figure 1.15. The cube connection for $N = 8$.

Cube Connection

The ith *cube* connection E_i, for $0 \leq i \leq n-1$, is defined only for $k = 2$ by

$$E_i(x_{n-1} \ldots x_{i+1}x_i x_{i-1} \ldots x_0) = x_{n-1} \ldots x_{i+1}\overline{x}_i x_{i-1} \ldots x_0$$

The ith cube connection complements the ith bit of the index. Figure 1.15 shows the cube connection for $i = 0$, 1, and 2 with $N = 8$. E_0 is also called the *exchange* connection.

Baseline Connection

The ith k-ary *baseline* permutation δ_i^k, for $0 \leq i \leq n-1$, is defined by

$$\delta_i^k(x_{n-1} \ldots x_{i+1}x_i x_{i-1} \ldots x_1 x_0) = x_{n-1} \ldots x_{i+1}x_0 x_i x_{i-1} \ldots x_1$$

The ith baseline connection performs a cyclic shifting of the $i+1$ least significant digits in the index to the right for one position. Figure 1.16 shows the baseline connection for $k = 2$, and $i = 0$, 1, and 2 with $N = 8$. Note that δ_0^k also defines the identity connection I.

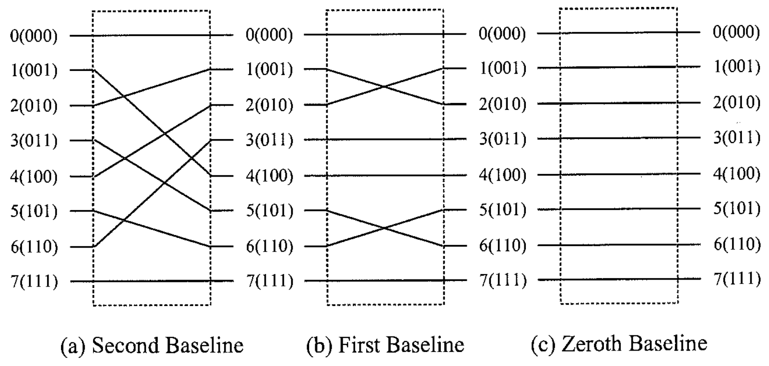


Figure 1.16. The baseline connection for $N = 8$.

1.7.5 Classification of Multistage Interconnection Networks

Depending on the availability of paths to establish new connections, MINs have been traditionally divided into three classes:

1. *Blocking.* A connection between a free input/output pair is not always possible because of conflicts with the existing connections. Typically, there is a unique path between every input/output pair, thus minimizing the number of switches and stages. However, it is also possible to provide multiple paths to reduce conflicts and increase fault tolerance. These blocking networks are also known as *multipath.*

2. *Nonblocking.* Any input port can be connected to any free output port without affecting the existing connections. Nonblocking networks have the same functionality as a crossbar. They require multiple paths between every input and output, which in turn leads to extra stages.

3. *Rearrangeable.* Any input port can be connected to any free output port. However, the existing connections may require rearrangement of paths. These networks also require multiple paths between every input and output but the number of paths and the cost is smaller than in the case of nonblocking networks.

Nonblocking networks are expensive. Although they are cheaper than a crossbar of the same size, their cost is prohibitive for large sizes. The best known example of nonblocking multistage network is the Clos network, initially proposed for telephone networks. Rearrangeable networks require less stages or simpler switches than a nonblocking network. The best known example of rearrangeable network is the Beneš network. Figure 1.17 shows an 8 × 8 Beneš network. For 2^n inputs, this network requires $2n - 1$ stages, and provides 2^{n-1} alternative paths. Rearrangeable networks require a central controller to rearrange connections, and were proposed for array processors. However, connections cannot be easily rearranged on multiprocessors because processors access the network asynchronously. So, rearrangeable networks behave like blocking networks when accesses are asynchronous. Thus, this class has not been included in Figure 1.2. We will mainly focus on blocking networks.

Depending on the kind of channels and switches, MINs can be split into two classes [253]:

1. *Unidirectional MINs.* Channels and switches are unidirectional.

2. *Bidirectional MINs.* Channels and switches are bidirectional. This implies that information can be transmitted simultaneously in opposite directions between neighboring switches.

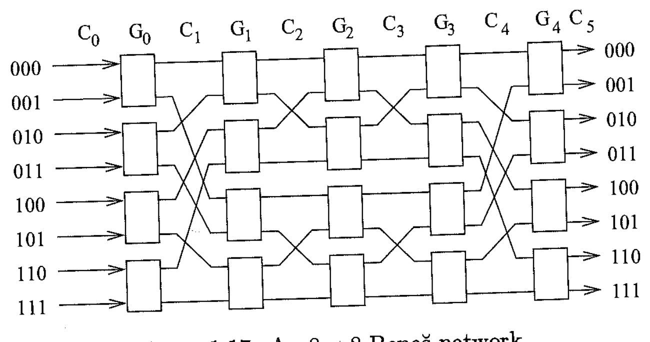


Figure 1.17. An 8 × 8 Beneš network.

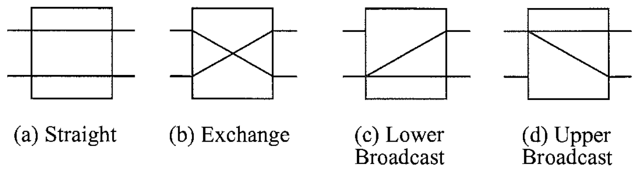


Figure 1.18. Four possible states of a 2×2 switch.

Additionally, each channel may be either multiplexed or replaced by two or more channels. In the latter case, the network is referred to as *dilated MIN*. Obviously, the number of ports of each switch must increase accordingly.

Unidirectional Multistage Interconnection Networks

The basic building blocks of unidirectional MINs are unidirectional switches. An $a \times b$ switch is a crossbar network with a inputs and b outputs. If each input port is allowed to connect to exactly one output port, at most $\min\{a, b\}$ connections can be supported simultaneously. If each input port is allowed to connect to many output ports, a more complicated design is needed to support the so-called *one-to-many* or *multicast* communication. In the *broadcast* mode or *one-to-all* communication, each input port is allowed to connect to all output ports. Figure 1.18 shows four possible states of a 2×2 switch. The last two states are used to support one-to-many and one-to-all communications.

In MINs with $N = M$, it is common to use switches with the same number of input and output ports, i.e., $a = b$. If $N > M$, switches with $a > b$ will be used. Such switches are also called *concentration switches*. In the case of $N < M$, *distribution switches* with $a < b$ will be used.

It can be shown that with N input and output ports, a unidirectional MIN with $k \times k$ switches requires at least $\lceil \log_k N \rceil$ stages to allow a connection path between any input port and any output port. By having additional stages, more connection paths may be used to deliver a message between an input port and an output port at the expense of extra hardware cost. Every path through the MIN crosses all the stages. Therefore, all the paths have the same length.

Four topologically equivalent unidirectional MINs are considered below. These MINs are a class of Delta networks.

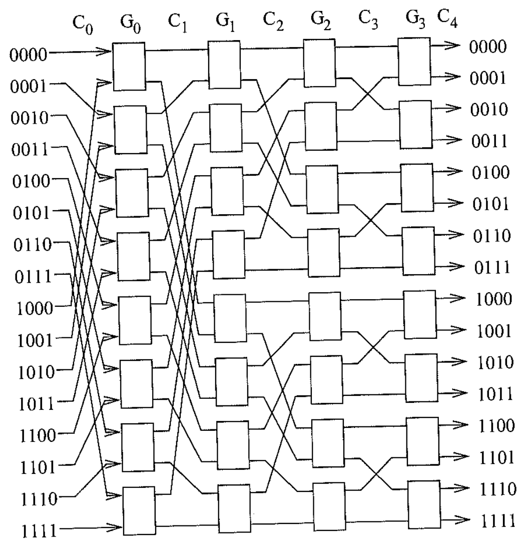
Baseline MINs. In a baseline MIN, connection pattern C_i is described by the $(n-i)$th baseline permutation δ^k_{n-i} for $1 \le i \le n$. Connection pattern C_0 is selected to be σ^k.

Butterfly MINs. In a butterfly MIN, connection pattern C_i is described by the ith butterfly permutation β^k_i for $0 \le i \le n-1$. Connection pattern C_n is selected to be β^k_0.

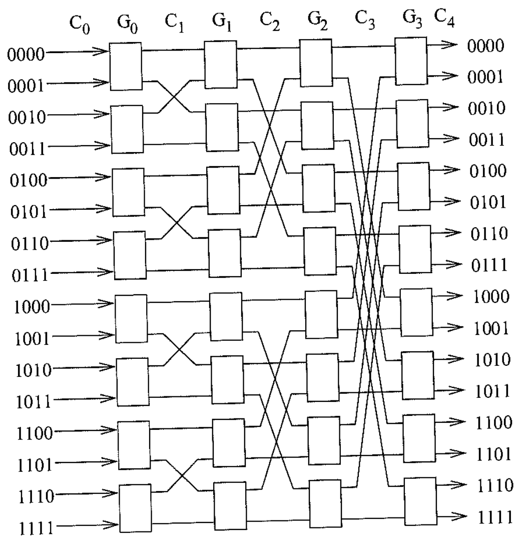
Cube MINs. In a cube MIN (or multistage cube network [317]), connection pattern C_i is described by the $(n-i)$th butterfly permutation β^k_{n-i} for $1 \le i \le n$. Connection pattern C_0 is selected to be σ^k.

Omega network. In an Omega network, connection pattern C_i is described by the perfect k-shuffle permutation σ^k for $0 \le i \le n-1$. Connection pattern C_n is selected to be β^k_0. Thus, all the connection patterns but the last one are identical. The last connection pattern produces no permutation.

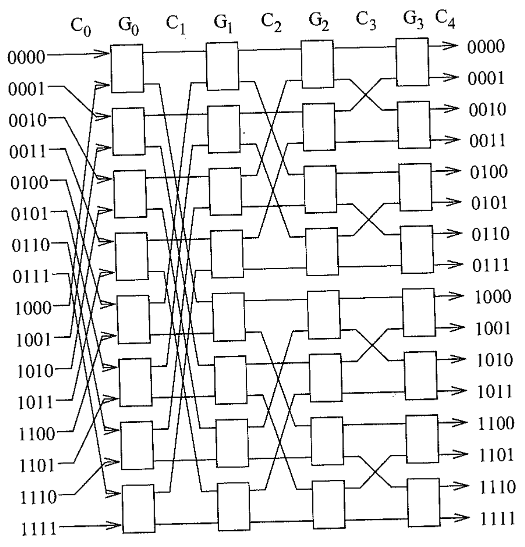
The topological equivalence of these MINs can be viewed as follows: Consider that each input link to the first stage is numbered using a string of n digits $s_{n-1}s_{n-2}\ldots s_1 s_0$, where $0 \le s_i \le k-1$, for $0 \le i \le n-1$. The least significant digit s_0 gives the address of the input port at the corresponding switch and the address of the switch is given by $s_{n-1}s_{n-2}\ldots s_1$. At each stage, a given switch is able to connect any input port with any output port. This can be viewed as changing the value of



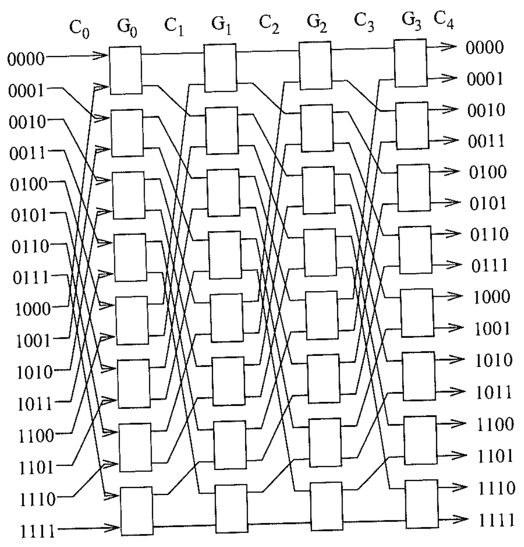
(a) Multistage Baseline Network



(b) Multistage Butterfly Network



(c) Multistage Cube Network



(d) Omega Network

Figure 1.19. Four 16 × 16 unidirectional multistage interconnection networks.

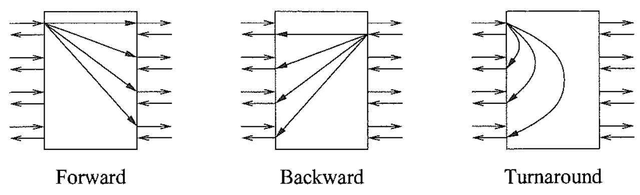


Figure 1.20. Connections in a bidirectional switch.

the least significant digit of the address. In order to be able to connect any input to any output of the network, it should be possible to change the value of all the digits. As each switch is only able to change the value of the least significant digit of the address, connection patterns between stages are defined in such a way that the position of digits is permuted, and after n stages all the digits have occupied the least significant position. Therefore, the above defined MINs differ in the order in which address digits occupy the least significant position.

Figure 1.19 shows the topology of four 16×16 unidirectional multistage interconnection networks: (a) baseline network, (b) butterfly network, (c) cube network, and (d) omega network.

Bidirectional Multistage Interconnection Networks

Figure 1.20 illustrates a bidirectional switch in which each port is associated with a pair of unidirectional channels in opposite directions. This implies that information can be transmitted simultaneously in opposite directions between neighboring switches. For ease of explanation, it is assumed that processor nodes are on the left-hand side of the network, as shown in Figure 1.21. A bidirectional switch supports three types of connections: *forward*, *backward*, and *turnaround* (see Figure 1.20). As turnaround connections between ports at the same side of a switch are possible, paths have different lengths. An eight-node butterfly bidirectional MIN (BMIN) is illustrated in Figure 1.21.

Paths are established in BMINs by crossing stages in forward direction, then establishing a turnaround connection, and finally crossing stages in backward direction. This is usually referred to as *turnaround routing*. Figure 1.22 shows two alternative paths from node S to node D in an eight-node butterfly BMIN. When crossing stages in forward direction, several paths are possible. Each switch can select any of its output ports. However, once the turnaround connection is crossed,

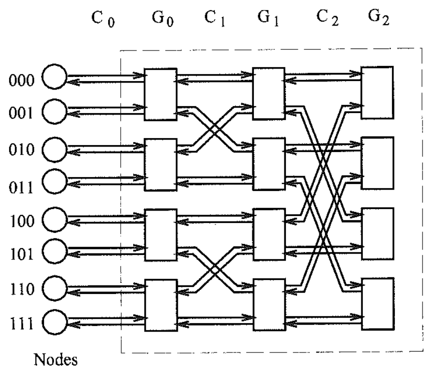


Figure 1.21. An eight-node butterfly bidirectional MIN.

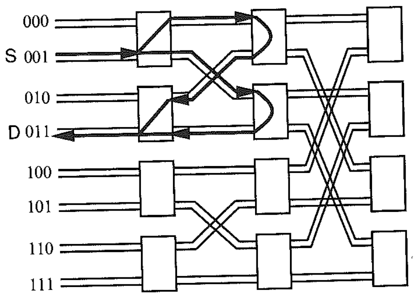


Figure 1.22. Alternative paths in an eight-node butterfly bidirectional MIN.

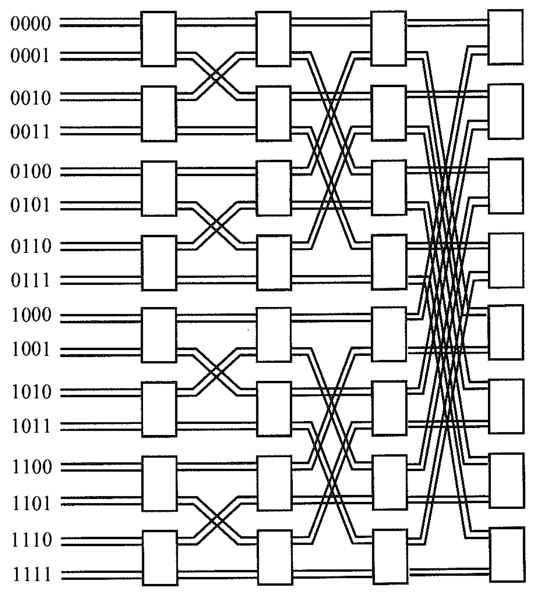
a single path is available up to the destination node. In the worst case, establishing a path in an n-stage BMIN requires crossing $2n - 1$ stages. This behavior closely resembles that of the Beneš network. Indeed, the baseline BMIN can be considered as a folded Beneš network.

As shown in Figure 1.23, a butterfly BMIN with turnaround routing can be viewed as a *fat tree* [201]. In a fat tree, processors are located at leaves, and internal vertices are switches. Transmission bandwidth between switches is increased by adding more links in parallel as switches become closer to the root switch. When a message is routed from one processor to another, it is sent up (in forward direction) the tree to the least common ancestor of the two processors, and then sent down (in backward direction) to the destination. Such a tree routing well explains the turnaround routing mentioned above.

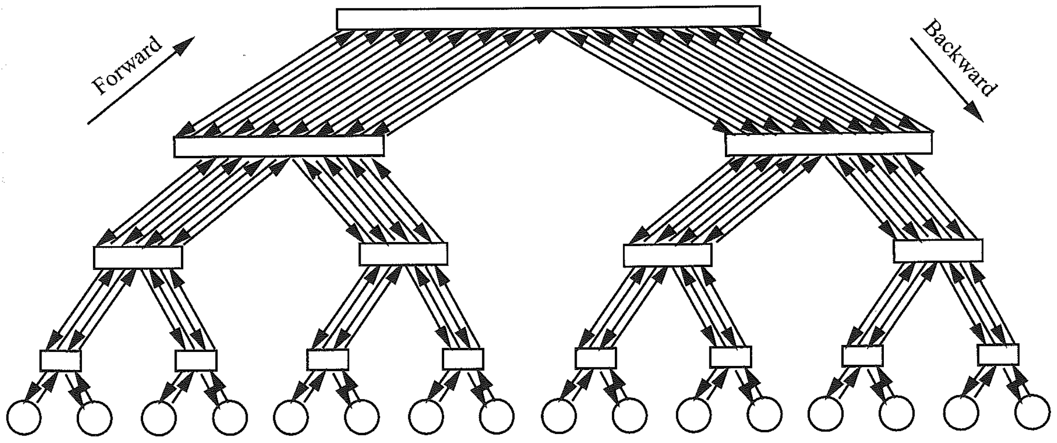
1.7.6 Examples

Several examples of parallel computers with indirect networks and commercial switches to build indirect networks are listed below.

- Myricom Myrinet: Supports regular and irregular topologies, 8 × 8 crossbar switch, 9-bit channels, full-duplex, 640 Mbits/s per link.
- Thinking Machines CM-5: Fat tree topology, 4-bit bidirectional channels at 40 MHz, aggregate bandwidth in each direction of 20 Mbytes/s.
- Inmos C104: Supports regular and irregular topologies, 32 × 32 crossbar switch, serial links, 100 Mbits/s per link.
- IBM SP2: Crossbar switches supporting Omega network topologies with bidirectional, 16-bit channels at 150MHz, 300 Mbytes/s in each direction.
- SGI SPIDER: Router with 20-bit bidirectional channels operating on both edges, 200 MHz clock, aggregate raw data rate of 1 Gbyte/s. Offers support for configurations as nonblocking multistage network topologies as well as irregular topologies.
- Tandem ServerNet: Irregular topologies, 8-bit bidirectional channels at 50 MHz, 50 Mbytes/s per link.



(a) A 16–Node Butterfly BMIN Built with 2 x 2 Switches



(b) A 16–Node Fat Tree

Figure 1.23. Fat tree and butterfly BMIN.

1.8 Hybrid Networks

In this section, we briefly describe some network topologies that do not fit into the classes described above. In general, hybrid networks combine mechanisms from shared-medium networks and direct or indirect networks. Therefore, they increase bandwidth with respect to shared-medium networks, and reduce the distance between nodes with respect to direct and indirect networks. There exist well-established applications of hybrid networks. This is the case for bridged LANs. However, for systems requiring very high performance, direct and indirect networks achieve better scalability than hybrid networks because point-to-point links are simpler and faster than shared-medium buses. Most high-performance parallel computers use direct or indirect networks. Recently hybrid networks have been gaining acceptance again. The use of optical technology enables the implementation of high-performance buses. Currently, some prototype machines are being implemented based on electrical as well as optical interconnects.

Many hybrid networks have been proposed for different purposes. The classification proposed for these networks is mainly dictated by the application fields. In general, hybrid networks can be modeled by a hypergraph [23], where the vertices of the hypergraph represent the set of processing nodes, and the edges represent the set of communication channels and/or buses. Note that an edge in a hypergraph can interconnect an arbitrary number of nodes. When an edge connects exactly two nodes then it represents a point-to-point channel. Otherwise it represents a bus. In some network designs, each bus has a single driving node. No other device is allowed to drive that bus. In this case, there is no need for arbitration. However, it is still possible to have several receivers at a given time, thus retaining the broadcast capability of buses. Obviously, every node in the network must drive at least one bus, therefore requiring a number of buses not lower than the number of nodes. In this case, the network topology can be modeled by a directed hypergraph.

1.8.1 Multiple Backplane Buses

Due to limited shared-medium network bandwidth, a shared-medium network can only support a small number of devices, has limited distance, and is not scalable to support a large system. Some approaches have been used or studied to remove such bottlenecks. One approach to increase network bandwidth is to have multiple buses as shown in Figure 1.24. However, concern about wiring and interface costs is a major reason why multiple buses have seen little use so far in multiprocessor design [246]. Due to the limitations of electrical packaging technology, it is unlikely to have a multiple-bus network with more than four buses. However, many more buses are likely feasible with other packaging technologies such as wavelength division multiplexing on fiber optics [279].

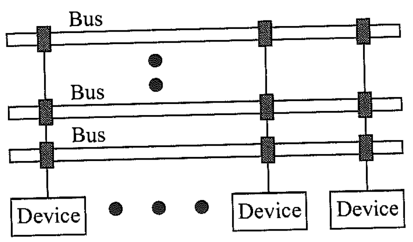


Figure 1.24. A multiple-bus network.

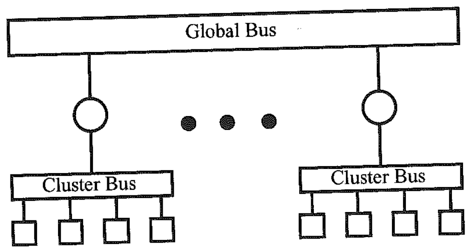


Figure 1.25. Two-level hierarchical buses.

1.8.2 Hierarchical Networks

Another approach to increase the network bandwidth is to have a hierarchical structure as shown in Figure 1.25. Different buses are interconnected by routers or bridges to transfer information from one side of the network to the other side of the network. These routers or bridges can filter the network traffic by examining the destination address of each passing message. The hierarchical network can expand the network area and handle more devices, but it is no longer a simple shared-medium network. This approach is used in bridged LANs. Usually, a higher bandwidth is available at the global bus. Otherwise, it may become a bottleneck. This can be achieved by using a faster technology. This is the case for the backbone LAN in some bridged LANs. Hierarchical networks have also been proposed as the interconnection scheme for shared-memory multiprocessors. Again, the global bus may become a bottleneck. For example, the Encore Gigamax addressed this problem by having an optical fiber bus to increase the bandwidth of the global bus.

1.8.3 Cluster-Based Networks

Cluster-based networks also have a hierarchical structure. Indeed, they can be considered as a subclass of hierarchical networks. Cluster-based networks combine the advantages of two or more kinds of networks at different levels in the hierarchy. For example, it is possible to combine the advantages of buses and point-to-point links by using buses at the lower level in the hierarchy to form clusters, and a direct network topology connecting clusters at the higher level. This is the case for the Stanford Directory Architecture for Shared-Memory (DASH) [203]. Figure 1.26 shows the basic architecture of this parallel computer. At the lower level, each cluster consists of four processors connected by a bus. At the higher level, a 2-D mesh connects the clusters. The broadcast capability of the bus is used at the cluster level to implement a snoopy protocol for cache coherence. The direct network at the higher level overcomes the bandwidth constraints of the bus, considerably increasing the scalability of the machine.

Other combinations are possible. Instead of combining buses and direct networks, the HP/Convex

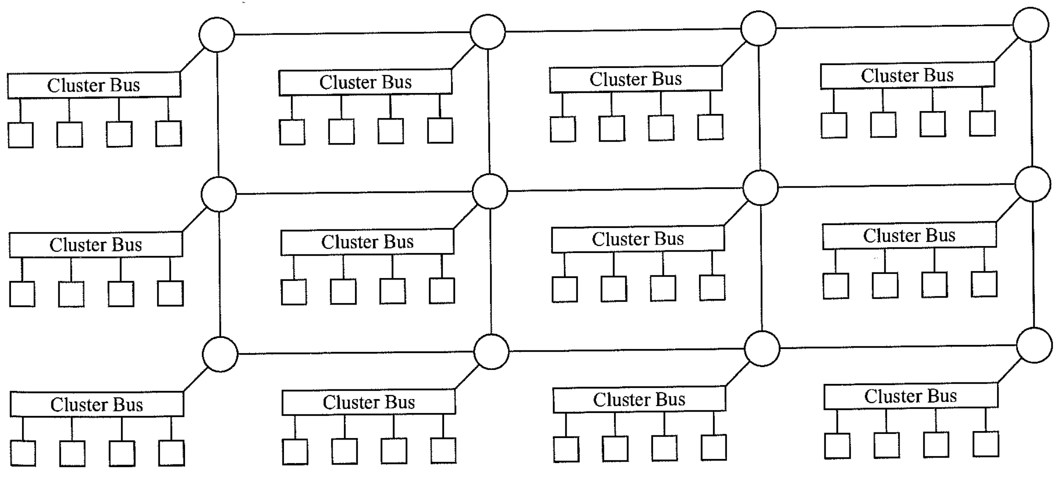


Figure 1.26. Cluster-based 2-D mesh.

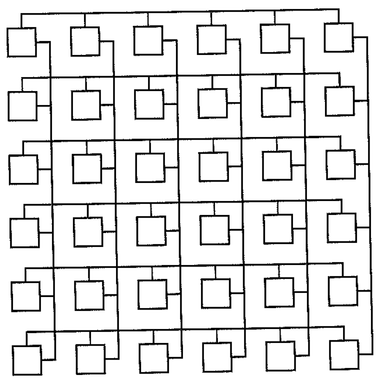


Figure 1.27. A two-dimensional hypermesh.

Exemplar multiprocessor combines indirect and direct networks. This multiprocessor has 5×5 nonblocking crossbars at the lower level in the hierarchy, connecting four functional blocks and one I/O interface to form clusters or *hypernodes*. Each functional block consists of two processors, two memory banks and interfaces. These hypernodes are connected by a second level *coherent toroidal interconnect* made out of multiple rings using Scalable Coherent Interface (SCI). Each ring connects one functional block from all the hypernodes. At the lower level of the hierarchy, the crossbars allow all the processors within a hypernode to access the interleaved memory modules in that hypernode. At the higher level, the rings implement a cache coherence protocol.

1.8.4 Other Hypergraph Topologies

Many other hybrid topologies have been proposed [25, 336, 348]. Among them, a particularly interesting class is the *hypermesh* [335]. A hypermesh is a regular topology consisting of a set of nodes arranged into several dimensions. Instead of having direct connections to the neighbors in each dimension, each node is connected to all the nodes in each dimension through a bus. There are several ways to implement a hypermesh. The most straightforward way consists of connecting all the nodes in each dimension through a shared bus. Figure 1.27 shows a 2-D hypermesh. In this network, multiple buses are arranged in two dimensions. Each node is connected to one bus in each dimension. This topology was proposed by Wittie [348], and was referred to as *spanning-bus hypercube*. This topology has a very low diameter, and average distance between nodes scales very well with network size. However, the overall network bandwidth does not scale well. Additionally, the frequent changes of bus mastership incur significant overheads.

An alternative implementation that removes the above mentioned constraints consists of replacing the single shared bus connecting the nodes along a given dimension by a set of as many buses as nodes in that dimension. This is the approach proposed in the Distributed Crossbar Switch Hypermesh (DCSH) [222, 223]. Figure 1.28 shows one dimension of the network. Each bus is driven by a single node. Therefore, there are no changes in mastership. Also, bandwidth scales with the number of nodes. Two major concerns, however, are the high number of buses required in the system and the very high number of input ports required at each node. Although the authors propose an electrical implementation, this topology is also suitable for optical interconnects.

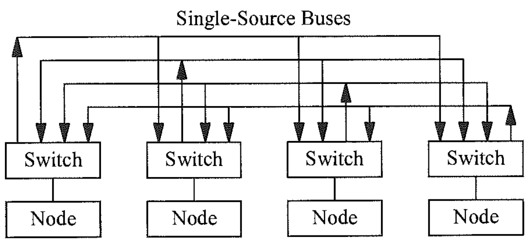


Figure 1.28. A one-dimensional Distributed Crossbar Switch Hypermesh.

1.9 A Unified View of Direct and Indirect Networks

Up to this point, most researchers and manufacturers considered direct and indirect networks as two completely different approaches for the interconnection of a set of nodes. However, the last few years have seen developments in router architecture, algorithms, and switch design evolve to a point that is blurring the distinction between these two classes of networks. For performance driven environments, as networks converge on the use of pipelined message transmission and source based routing algorithms, the major differences between the switch architectures for direct and indirect networks is becoming more a question of topology. Direct networks have typically used routers with routing algorithms implemented in the network. Switch-based designs have had routing performed at the source. Modern routers are being designed to support simple, fast, reprogrammable routing decisions within the network with substantial control of routing at the message source. This expands the base of applications for the (sizeable) investment in router/switch designs. This flexibility is also evidenced by the fact that the traditional high-performance multiprocessor router/switch designs are moving up to implementations involving workstation and personal computer (PC) clusters.

In particular, during the last few years several manufacturers have introduced switches to interconnect a set of processors. These switch-based networks allow the user to define the topology. Let us consider the particular case in which each switch has a single processor directly attached to it, and the remaining ports are used to connect with other switches. This network does not differ substantially from a direct network because each switch can be considered as a router associated with the corresponding node. Indeed, the functionality of current routers and switches is practically identical. Moreover, some manufacturers proposed the use of the same switch to implement either direct or indirect networks. This is the case of the Inmos C104 switch [228] and the SGI SPIDER [118]. For interconnection networks using the former switch, Inmos proposed the use of bidirectional multistage interconnection networks as well as direct topologies like meshes. In the latter case, each switch is connected to a single node through a set of independent ports. Also, two adjacent switches may use several ports in parallel to communicate between them. The SGI SPIDER router implementation was designed to support configurations as nonblocking multistage networks, as irregular topologies, and as routers for conventional multiprocessor topologies. Such flexibility is typically achieved using topology-independent routing algorithms and flexible switching techniques.

Therefore, we can view networks using point-to-point links as a set of interconnected switches, each one being connected to zero, one, or more nodes. Direct networks correspond to the case where every switch is connected to a single node. Crossbar networks correspond to the case where there is a single switch connected to all the nodes. Multistage interconnection networks correspond to the case where switches are arranged into several stages and the switches in intermediate stages are not connected to any processor. However, other choices are possible under this unified view of direct and indirect interconnection networks. Effectively, nothing prevents the designer from interconnecting switches using a typical direct network topology, and connecting several processors to each switch. This is the case for the Cray T3D, which implements a 3-D torus topology, connecting two processors to each router.

The unified view allows the development of generic routing algorithms. In these algorithms, the destination address specifies the destination switch as well as the port number in that switch that is connected to the destination processor. Note that routing algorithms for direct networks are a particular case of these generic algorithms. In this case, there is a single processor connected to each switch. So, the output port in the last switch is not required, assuming that processors are connected to the same port in every switch. This unified view also allows the application of many results developed for direct networks to indirect networks, and vice versa. For example, the theory of deadlock avoidance presented in Chapter 3 was initially developed for direct networks but can also be applied to indirect networks with minor changes, as will be shown in that chapter. Obviously, not all the results can be directly applied to both network classes. Additionally, applying some results to other network classes may require a considerable effort. In the remaining chapters of the book, most techniques will be described for the class of networks for which they were initially proposed. However, it is important to keep in mind that most of those techniques can also be applied to other network topologies by taking into account the unified view proposed in this section.

Chapter 2

Message Switching Layer

Interprocessor communication can be viewed as a hierarchy of services starting from the physical layer that synchronizes the transfer of bit streams to higher-level protocols layers that perform functions such as packetization, data encryption, data compression, etc. Such a layering of communication services is common in the local and wide area network communities. While there currently may not be a consensus on a standard set of layers for multiprocessor systems, we find it useful to distinguish between three layers in the operation of the interconnection network: the *routing layer*, the *switching layer*, and the *physical layer*. The physical layer refers to link-level protocols for transferring messages and otherwise managing the physical channels between adjacent routers. The switching layer utilizes these physical layer protocols to implement mechanisms for forwarding messages through the network. Finally, the routing layer makes routing decisions to determine candidate output channels at intermediate router nodes and thereby establish the path through the network. The design of routing protocols and their properties, e.g., deadlock and livelock freedom, are largely determined by the services provided by the switching layer.

This chapter focuses on the techniques that are implemented within the network routers to realize the switching layer. These techniques differ in several respects. The *switching techniques* determine when and how internal switches are set to connect router inputs to outputs and the time at which message components may be transferred along these paths. These techniques are coupled with *flow control* mechanisms for the synchronized transfer of units of information between routers and through routers in forwarding messages through the network. Flow control is tightly coupled with *buffer management* algorithms that determine how message buffers are requested and released, and as a result determine how messages are handled when blocked in the network. Implementations of the switching layer differ in decisions made in each of these areas, and in their relative timing, i.e., when one operation can be initiated relative to the occurrence of the other. The specific choices interact with the architecture of the routers and traffic patterns imposed by parallel programs in determining the latency and throughput characteristics of the interconnection network.

As we might expect, the switching techniques employed in multiprocessor networks initially followed those techniques employed in local and wide area communication networks, e.g., circuit switching and packet switching. However, as the application of multiprocessor systems spread into increasingly compute-intensive domains, the traditional layered communication designs borrowed from LANs became a limiting performance bottleneck. New switching techniques and implementations evolved that were better suited to the low latency demands of parallel programs. This chapter reviews these switching techniques and their accompanying flow control and buffer management algorithms.

2.1 Network and Router Model

In comparing and contrasting alternative implementations of the switching layer we are interested in evaluating their impact on the router implementations. The implementations in turn determine the cycle time of router operation and therefore the resulting message latency and network bandwidth. The architecture of a generic router is shown in Figure 2.1 and is comprised of the following major components.

- *Buffers.* These are first-in first-out (FIFO) buffers for storing messages in transit. In the above model, a buffer is associated with each input physical channel and each output physical channel. In alternative designs, buffers may be associated only with inputs (input buffering) or outputs (output buffering). The buffer size is an integral number of flow control units.

- *Switch.* This component is responsible for connecting router input buffers to router output buffers. High-speed routers will utilize crossbar networks with full connectivity, while lower-speed implementations may utilize networks that do not provide full connectivity between input buffers and output buffers.

- *Routing and arbitration unit.* This component implements the routing algorithms, selects the output link for an incoming message, and accordingly sets the switch. If multiple messages simultaneously request the same output link this component must provide for arbitration between them. If the requested link is busy, the incoming message remains in the input buffer. It will be routed again after the link is freed and if it successfully arbitrates for the link.

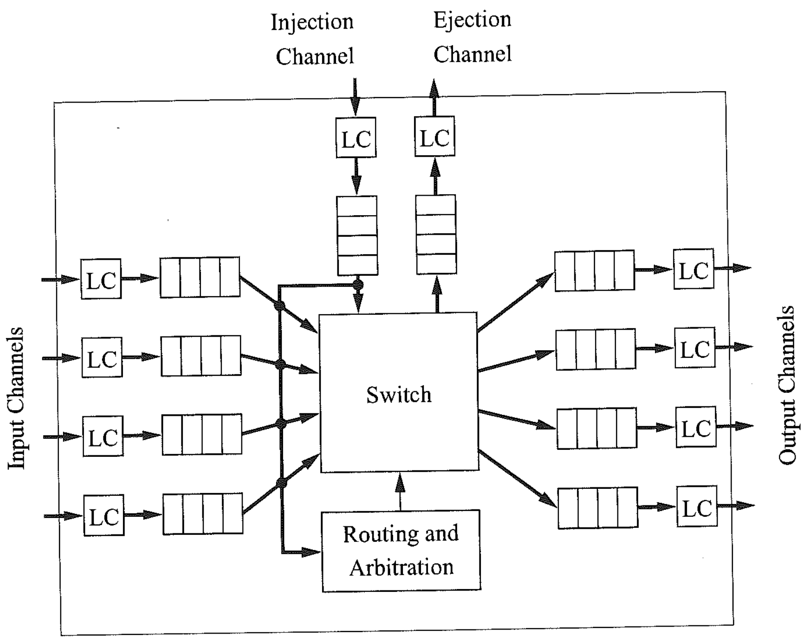


Figure 2.1. Generic router model. (LC = Link controller.)

- *Link controllers (LC).* The flow of messages across the physical channel between adjacent routers is implemented by the link controller. The link controllers on either side of a channel coordinate to transfer units of flow control.
- *Processor interface.* This component simply implements a physical channel interface to the processor rather than to an adjacent router. It consists of one or more injection channels from the processor and one or more ejection channels to the processor. Ejection channels are also referred to as delivery channels or consumption channels.

From the point of view of router performance we are interested in two parameters [57]. When a message first arrives at a router, it must be examined to determine the output channel over which the message is to be forwarded. This is referred to as the *routing delay*, and typically includes the time to set the switch. Once a path has been established through a router by the switch, we are interested in the rate at which messages can be forwarded through the switch. This rate is determined by the propagation delay through the switch (intrarouter delay), and the signaling rate for synchronizing the transfer of data between the input and output buffers. This delay has been characterized to as the the *internal flow control* latency [57]. Similarly, the delay across the physical links (interrouter delay) is referred to as the *external flow control* latency. The routing delay and flow control delays collectively determine achievable message latency through the switch, and along with contention by messages for links, determines the network throughput.

The following section addresses some basic concepts in the implementation of the switching layer, assuming the generic router model shown in Figure 2.1. The remainder of the chapter focuses on alternative implementations of the switching layer.

2.2 Basic Concepts

Switching layers can be distinguished by the implementation and relative timing of flow control operations and switching techniques. In addition, these operations may be overlapped with the time to make routing decisions.

Flow control is a synchronization protocol for transmitting and receiving a unit of information. The *unit of flow control* refers to that portion of the message whose transfer must be synchronized. This unit is defined as the smallest unit of information whose transfer is requested by the sender and acknowledged by the receiver. The request/acknowledgment signaling is used to ensure successful transfer and the availability of buffer space at the receiver. Note that there is no restriction on when requests or acknowledgments are actually sent or received. Implementation efficiency governs the actual exchange of these control signals, e.g., the use of block acknowledgments. For example, it is easy to think of messages in terms of fixed-length packets. A packet is forwarded across a physical channel or from the input buffers of a router to the output buffers. Note that these transfers are atomic in the sense that sufficient buffering must be provided so that a packet is either transferred in its entirety, or transmission is delayed until sufficient buffer space becomes available. In this example, the *flow* of information is managed and *controlled* at the level of an entire packet.

Flow control occurs at two levels. In the preceding example, *message flow control* occurs at the level of a packet. However, the transfer of a packet across a physical channel between two routers make take several steps or cycles, e.g., the transfer of a 128-byte packet across a 16-bit data channel. The resulting multicycle transfers use *physical channel flow control* to forward a message flow control unit across the physical link connecting routers.

Switching techniques differ in the relationship between the sizes of the physical and message flow control units. In general, each message may be partitioned into fixed-length *packets*. Packets in

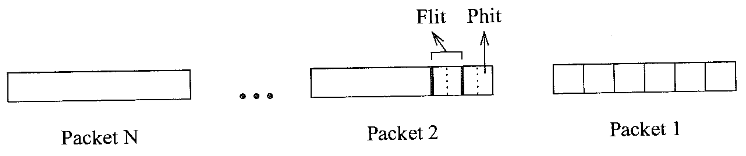


Figure 2.2. Alternative flow control units in a message.

turn may be broken into message flow control units or *flits* [78]. Due to channel width constraints, multiple physical channel cycles may be used to transfer a single flit. A *phit* is the unit of information that can be transferred across a physical channel in a single step or cycle. Flits represent logical units of information as opposed to phits which correspond to physical quantities, i.e., the number of bits that can be transferred in parallel in a single cycle. An example of a message comprised of N packets, 6 flits/packet and 2 phits/flit is shown in Figure 2.2.

The relationships between the sizes of phits, flits, and packets differs across machines. Many machines have the phit size equivalent to the flit size. In the IBM SP2 switch [327], a flit is 1 byte and is equivalent to a phit. Alternatively, the Cray T3D [311] utilizes flit-level message flow control where each flit is comprised of eight 16-bit phits. The specific choices reflect trade-offs in performance, reliability, and implementation complexity.

Example 2.1

There are many candidate synchronization protocols for coordinating phit transfers across a channel, and Figure 2.3 illustrates an example of a simple four-phase asynchronous handshaking protocol. Only one direction of transfer is shown. Router $R1$ asserts the RQ signal when information is to be transferred. Router $R2$ responds by reading the data and asserting the ACK signal. This leads to deasserting RQ by $R1$ which in turn causes $R2$ to deassert ACK. This represents one cycle of operation wherein 1 phit is transferred across the channel.

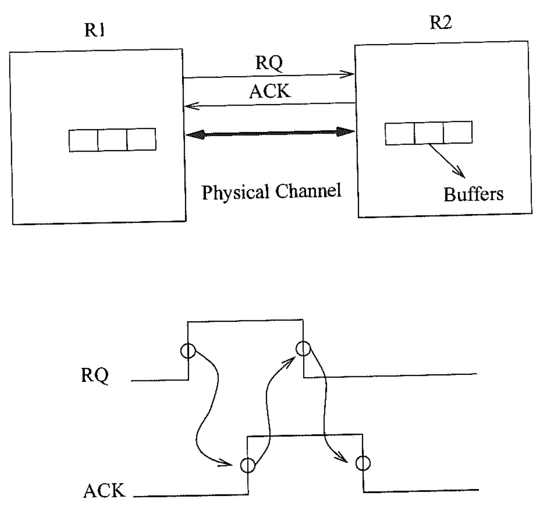


Figure 2.3. An example of asynchronous physical channel flow control.

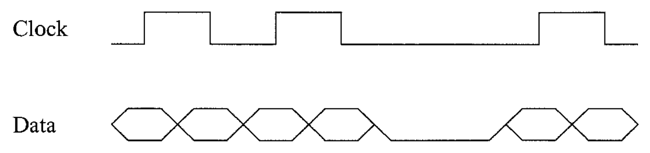


Figure 2.4. An example of synchronous physical channel flow control.

Another transfer may now be initiated. The ACK signal can serve to both acknowledge reception (rising edge) as well as the availability of buffer space (falling edge) for the transmission of the next unit of information. Thus, the flow of phits across the channel is synchronized to prevent buffer overflow in the receiving end of the channel. Note that higher-level message flow control mechanisms can ensure the availability of sufficient buffer space for each flit.

Example 2.2

Physical channel flow control can also be synchronous as shown in Figure 2.4. The clock signal is transmitted across the channel and both the rising and falling edges of the clock line validate the data on the data lines for the receiver. Such physical channel flow control mechanism can be found within the routers of the Intel iPSC/2 and iPSC 860 machines [257]. Figure 2.4 does not show the acknowledgment signals to indicate the availability of buffer space on the receiving node. Acknowledgment signals may be provided for the transfer of each data item, or the channel may utilize block acknowledgments, i.e., each acknowledgment signal indicates the availability of buffer space for some fixed number of data items. Such an approach both reduces the acknowledgment traffic as well as the signaling rate of acknowledgments. It also enables other optimizations for high-speed channel operation that are discussed in Chapter 7.

While interrouter transfers are necessarily constructed in terms of phits, the switching technique deals with flits (which could be defined to be the complete message packet!). The switching techniques set the internal switch to connect input buffers to output buffers, and forward flits along this path. These techniques are distinguished by the time at which they occur relative to the message flow control operation and the routing operation. For example, switching may take place after a flit has been received in its entirety. Alternatively, the transfer of a flit through the switch may begin as soon as the routing operation has been completed, but before the remainder of the flit has been received from the preceding router. In this case switching is overlapped with message-level flow control. In at least one proposed switching technique, switching begins after the first phit is received and even before the routing operation is complete! In general, high-performance switching techniques seek to overlap switching and message flow control as far as possible. While such an approach provides low-latency communication, it does complicate link-level diagnosis and error recovery.

This chapter describes the prevalent switching techniques that have been developed to date for use in current-generation multiprocessors. Switching layers can share the same physical channel flow control mechanism, but differ in the choice of message flow control. Unless otherwise stated, flow control will refer to message flow control.

2.3 Basic Switching Techniques

For the purposes of comparison, for each switching technique we will consider the computation of the base latency of an L-bit message in the absence of any traffic. The phit size and flit size are assumed to be equivalent and equal to the physical data channel width of W bits. The routing header is assumed to be 1 flit, thus the message size is $L + W$ bits. A router can make a routing decision in t_r seconds. The physical channel between two routers operates at B Hz, i.e., the physical channel bandwidth is BW bits per second. The propagation delay across this channel is denoted by $t_w = \frac{1}{B}$. Once a path has been set up through the router, the intrarouter delay or switching delay is denoted by t_s. The router internal data paths are assumed to be matched to the channel width of W bits. Thus, in t_s seconds a W-bit flit can be transferred from the input of the router to the output. The source and destination processors are assumed to be D links apart. The relationship between these components as they are used to compute the no-load message latency is shown in Figure 2.5.

2.3.1 Circuit Switching

In circuit switching, a physical path from the source to the destination is reserved prior to the transmission of the data. This is realized by injecting the routing header flit into the network. This *routing probe* contains the destination address and some additional control information. This routing probe progresses towards the destination reserving physical links as it is transmitted through intermediate routers. When the probe reaches the destination, a complete path has been set up and an acknowledgment is transmitted back to the source. The message contents may now be transmitted at the full bandwidth of the hardware path. The circuit may be released by the destination or by the last few bits of the message. In the Intel iPSC/2 routers [257], the acknowledgments are multiplexed in the reverse direction on the same physical line as the message. Alternatively, implementations may provide separate signal lines to transmit acknowledgment signals. A time-space diagram of the transmission of a message over three links is shown in Figure 2.6. The header probe is forwarded across three links followed by the return of the acknowledgment. The shaded boxes represent the times during which a link is busy. The space between these boxes represents the time to process the routing header, and the intrarouter propagation delays. The clear box represents the duration the links are busy transmitting data through the circuit. Note that the routing and intrarouter delays at the source router are not included and would precede the box corresponding to the first busy link.

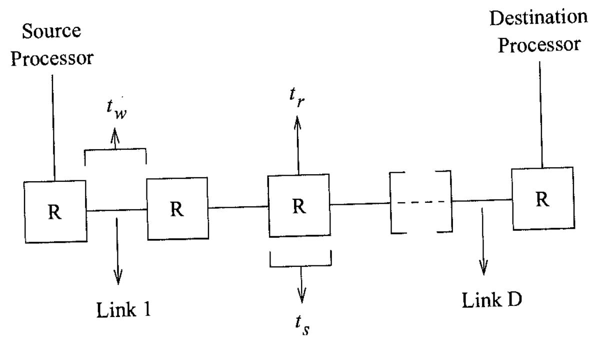


Figure 2.5. View of the network path for computing the no-load latency. (R = Router.)

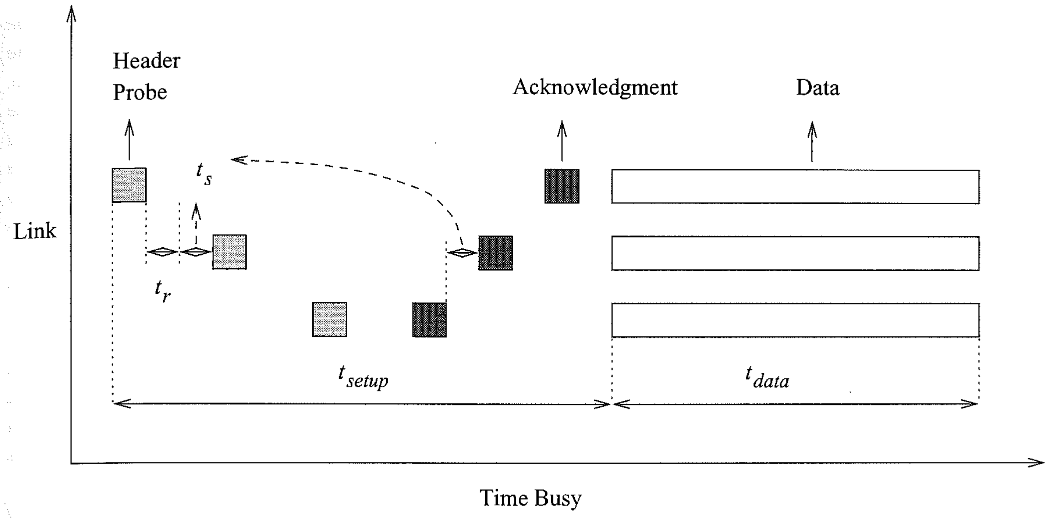


Figure 2.6. Time-space diagram of a circuit-switched message.

An example of a routing probe used in the JPL Mark III binary hypercube is shown in Figure 2.7. The network of the Mark III was quite flexible, supporting several distinct switching mechanisms in configurations up to 2,048 nodes. Bits 0 and 16 of the header define the switching technique being employed. The values shown in Figure 2.7 are for circuit switching. Bits 17–19 are unused while the destination address is provided in bits 1–11. The remaining 4-bit fields are used to address 1 of 11 output links at each individual router. There are 11 such fields supporting a 11-dimensional hypercube and requiring a two-word, 64-bit header. The path is computed at the source node. An alternative could have been to compute the value of the output port at each node rather than storing the addresses of all intermediate ports in the header. This would significantly reduce the size of the routing header probe. However, this scheme would require routing time and buffering logic within the router. In contrast, the format shown in Figure 2.7 enables a fast lookup using the header and simple processing within the router.

31 28		23 20		16	15 12	11 1	0
CHN	• • • •	CHN	XXX	1	0000	DEST	0

31 28		3 0
CHN	• • • • • • • •	CHN

Figure 2.7. An example of the format of a circuit probe. (CHN = Channel number; DEST = Destination address; XXX = Not defined.)

Circuit switching is generally advantageous when messages are infrequent and long, i.e., the message transmission time is long compared to the path setup time. The disadvantage is that the physical path is reserved for the duration of the message and may block other messages. For example, consider the case where the probe is blocked waiting for a physical link to become free. All of the links reserved by the probe up to that point remain reserved, cannot be used by other circuits, and may be blocking other circuits preventing them from being set up. Thus, if the size of the message is not that much greater than the size of the probe, it would be advantageous to transmit the message along with the header and buffer the message within the routers while waiting for a free link. This alternative technique is referred to as *packet switching*, and will be studied in Section 2.3.2.

The base latency of a circuit-switched message is determined by the time to set up a path, and the subsequent time the path is busy transmitting data. The router operation differs a bit from that shown in Figure 2.1. While the routing probe is buffered at each router, data bits are not. There are no intervening data buffers in the circuit which operates effectively as a single wire from source to destination. This physical circuit may use asynchronous or synchronous flow control as shown in Figures 2.3 or 2.4. In this case the time for the transfer of each flit from source to destination is determined by the clock speed of the synchronous circuit or signaling speed of the asynchronous handshake lines. The signaling period or clock period must be greater than the propagation delay through this circuit. This places a practical limit on the speed of circuit switching as a function of system size. More recent techniques have begun to investigate the use of this delay as a form of storage. At very high signal speeds, multiple bits may be present on a wire concurrently, proceeding as *waves* of data. Such techniques have been referred to as *wave pipelining* [112]. Using such techniques the technological limits of router and network designs have been reexamined [102, 310] and it has been found that substantial improvements in wire bandwidth is possible. The challenges to widespread use remain the design of circuits that can employ wave pipelining with stable and predictable delays, while in large designs the signal skew remains particularly challenging.

Without wave pipelining, from Figure 2.6 we can write an expression for the base latency of a message as follows:

$$\begin{aligned} t_{circuit} &= t_{setup} + t_{data} \\ t_{setup} &= D[t_r + 2(t_s + t_w)] \\ t_{data} &= \frac{1}{B}\left\lceil \frac{L}{W} \right\rceil \end{aligned} \tag{2.1}$$

Actual latencies clearly depend on a myriad of implementation details. Figure 2.6 represents some simplifying assumptions about the time necessary for various events such as processing an acknowledgment, or initiating the transmission of the first data flit. The factor of 2 in the setup cost represents the time for the forward progress of the header and the return of the acknowledgment. The use of B Hz as the channel speed represents the transmission across hardwired path from source to destination.

2.3.2 Packet Switching

In circuit switching, the complete message is transmitted after the circuit has been set up. Alternatively, the message can be partitioned and transmitted as fixed-length packets, e.g., 128 bytes. The first few bytes of a packet contain routing and control information and are referred to as the *packet header*. Each packet is individually routed from source to destination. A packet is completely buffered at each intermediate node before it is forwarded to the next node. This is the reason why this switching technique is also referred to as *store-and-forward* (SAF) switching. The header information is extracted by the intermediate router and used to determine the output link over which

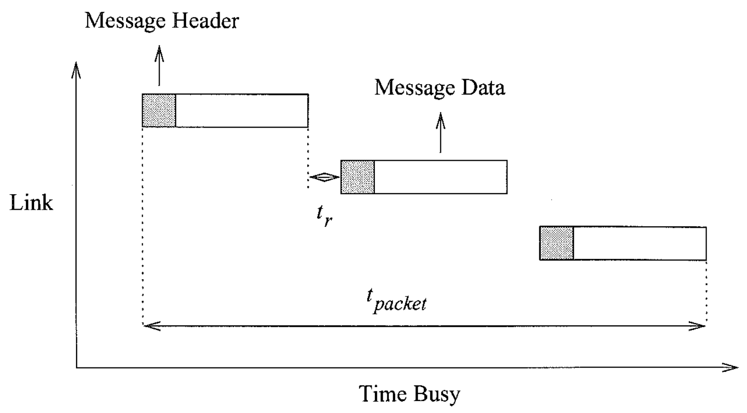


Figure 2.8. Time-space diagram of a packet-switched message.

the header is to be forwarded. A time-space diagram of the progress of a packet across three links is shown in Figure 2.8. From the figure we can see that the latency experienced by a packet is proportional to the distance between the source and destination nodes. Note that the figure has omitted the packet latency through the router.

Packet switching is advantageous when messages are short and frequent. Unlike circuit switching, where a segment of a reserved path may be idle for a significant period of time, a communication link is fully utilized when there are data to be transmitted. Many packets belonging to a message can be in the network simultaneously even if the first packet has not yet arrived at the destination. However, splitting a message into packets produces some overhead. In addition to the time required at source and destination nodes, every packet must be routed at each intermediate node. An example of the format of a data packet header is shown in Figure 2.9. This is the header format used in the JPL Hyperswitch. Since the hyperswitch can operate in one of many modes, bit field 12–16 and bit 0 collectively identify the switching technique being used: in this case it is packet switching using a fixed-path routing algorithm. Bits 1–11 identify the destination address, limiting the format to systems of 2,048 processors or less. The LEN field identifies the packet size in units of 192 bytes. For the current implementation packet size is limited to 384 bytes. If packets are routed adaptively through the network, packets from the same message may arrive at the destination out of order. In this case the packet headers must also contain sequencing information so that the messages can be reconstructed at the destination.

In multidimensional, point-to-point networks it is evident that the storage requirements at the individual router nodes can become extensive if packets can become large and multiple packets must be buffered at a node. In the JPL implementation, packets are not stored in the router,

31		16	15 12	11 1	0
LEN	XXX	0	0001	DEST	0

Figure 2.9. An example packet header format. (DEST = Destination address; LEN = Packet length in units of 192 bytes; XXX = Not defined.)

but are rather stored in the memory of the local node and a special-purpose message coprocessor is used to process the message, i.e., compute the address of an output channel and forward the message. Other multicomputers using packet switching also buffer packets in the memory of the local node (Cosmic Cube [313], Intel iPSC/1 [164]). This implementation is no doubt a carryover from implementations in local and wide area networks where packets are buffered in memory and special-purpose coprocessors and network interfaces have been dedicated to processing messages. In modern multiprocessors, the overhead and impact on message latency render such message processing impractical. To be viable, messages must be buffered and processed within the routers. Storage requirements can be reduced by using central queues in the router that are shared by all input channels rather than providing buffering at each input channel, output channel, or both. In this case, internal and external flow control delays will typically take many cycles.

The base latency of a packet-switched message can be computed as follows:

$$t_{packet} = D\left\{t_r + (t_s + t_w)\left\lceil\frac{L+W}{W}\right\rceil\right\} \tag{2.2}$$

This expression follows the router model in Figure 2.1, and as a result includes factors to represent the time for the transfer of packet of length $L + W$ bits across the channel (t_w) as well as from the input buffer of the router to the output buffer (t_s). However, in practice, the router could be only input-buffered, output-buffered, or use central queues. The above expression would be modified accordingly. The important point to note is that the latency is directly proportional to the distance between the source and destination nodes.

2.3.3 Virtual Cut-Through (VCT) Switching

Packet switching is based on the assumption that a packet must be received in its entirety before any routing decision can be made and the packet forwarded to the destination. This is not generally true. Consider a 128-byte packet and the router model shown in Figure 2.1. In the absence of 128-byte-wide physical channels, the transfer of the packet across the physical channel will take multiple cycles. However, the first few bytes will contain routing information that is typically available after the first few cycles. Rather than waiting for the entire packet to be received, the packet header can be examined as soon as it is received. The router can start forwarding the header and following data bytes as soon as routing decisions have been made and the output buffer is free. In fact, the message does not even have to be buffered at the output and can *cut through* to the input of the next router before the complete packet has been received at the current router. This switching technique is referred to as *virtual cut-through* (VCT) switching. In the absence of blocking, the latency experienced by the header at each node is the routing latency and propagation delay through the router and along the physical channels. The message is effectively pipelined through successive switches. If the header is blocked on a busy output channel, the complete message is buffered at the node. Thus, at high network loads, VCT switching behaves like packet switching.

Figure 2.10 illustrates a time-space diagram of a message transferred using VCT switching where the message is blocked after the first link waiting for an output channel to become free. In this case we see that the complete packet has to be transferred to the first router where it remains blocked waiting for a free output port. However, from the figure we can see that the message is successful in cutting through the second router and across the third link.

The base latency of a message that successfully cuts through each intermediate router can be computed as follows:

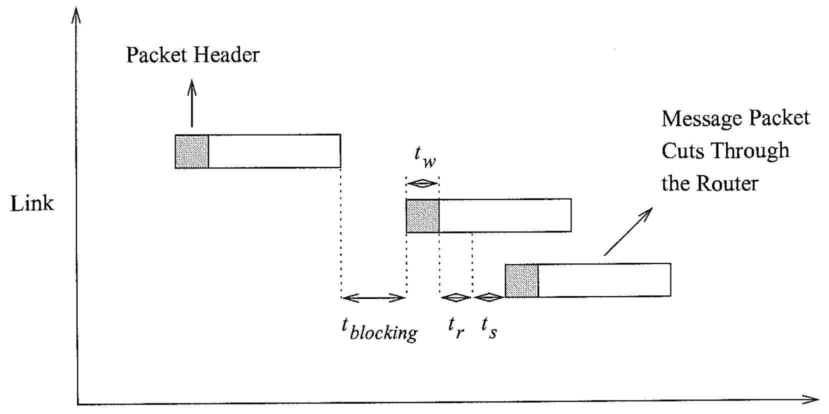


Figure 2.10. Time-space diagram of a virtual cut-through switched message. ($t_{blocking}$ = Waiting time for a free output link.)

$$t_{vct} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \tag{2.3}$$

Cut-through routing is assumed to occur at the flit level with the routing information contained in 1 flit. This model assumes that there is no time penalty for cutting through a router if the output buffer and output channel are free. Depending on the speed of operation of the routers this may not be realistic. Note that only the header experiences routing delay, as well as the switching delay and wire delay at each router. This is because the transmission is pipelined and the switch is buffered at the input and output. Once the header flit reaches the destination, the cycle time of this message pipeline is determined by the maximum of the switch delay and wire delay between routers. If the switch had been buffered only at the input, then in one cycle of operation, a flit traverses the switch and channel between the routers. In this case the coefficient of the second term and the pipeline cycle time would be $(t_s + t_w)$. Note that the unit of message flow control is a packet. Therefore even though the message may cut through the router, sufficient buffer space must be allocated for a complete packet in case the header is blocked.

2.3.4 Wormhole Switching

The need to buffer complete packets within a router can make it difficult to construct small, compact, and fast routers. In wormhole switching, message packets are also pipelined through the network. However the buffer requirements within the routers are substantially reduced over the requirements for VCT switching. A message packet is broken up into flits. The flit is the unit of message flow control, and input and output buffers at a router are typically large enough to store a few flits. For example, the message buffers in the Cray T3D are 1 flit deep and each flit is comprised of eight 16-bit phits. The message is pipelined through the network at the flit level and is typically too large to be completely buffered within a router. Thus, at any instant in time a blocked message occupies buffers in several routers. The time-space diagram of a wormhole-switched message is shown in Figure 2.11. The clear rectangles illustrate the propagation of flits across the physical channel. The

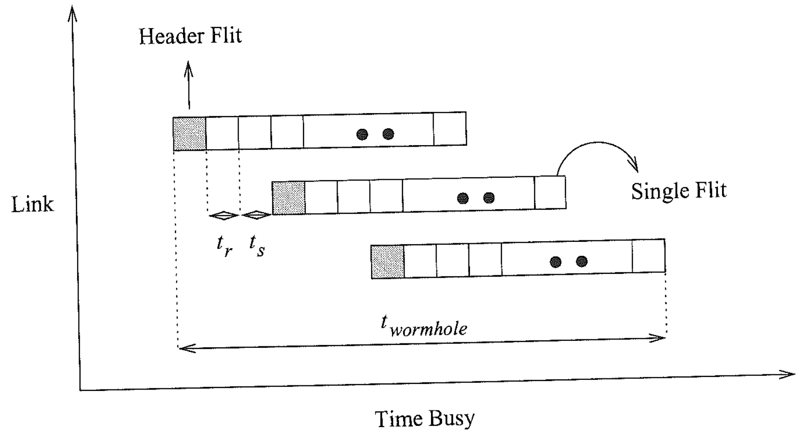


Figure 2.11. Time-space diagram of a wormhole-switched message.

shaded rectangles illustrate the propagation of header flits across the physical channels. Routing delays and intrarouter propagation of the header flits are also captured in this figure. The primary difference between wormhole switching and VCT switching is that the unit of message flow control is a single flit and, as a consequence, the use of small buffers. An entire message cannot be buffered at a router.

In the absence of blocking the message packet is pipelined through the network. However, the blocking characteristics are very different from that of VCT. If the required output channel is busy, the message is blocked "in place." For example, Figure 2.12 illustrates a snapshot of a message being transmitted through routers $R1$, $R2$, and $R3$. Input and output buffers are 2 flits deep and the routing header is 2 flits. At router $R3$, message A requires an output channel that is being used by message B. Therefore message A blocks in place. The small buffer sizes at each node ($<$ message size) causes the message to occupy buffers in multiple routers, similarly blocking other messages. In effect dependencies between buffers span multiple routers. This property complicates the issue of deadlock freedom. However, it is no longer necessary to use the local processor memory to buffer messages, significantly reducing average message latency. The small buffer requirements and message pipelining enable the construction of routers that are small, compact, and fast.

Examples of the format of wormhole-switched packets in the Cray T3D are shown in Figure 2.13. In this machine, a phit is 16 bits wide — the width of a T3D physical channel — and a flit is comprised of 8 phits. A word is 64 bits and thus 4 phits. A message is comprised of header phits and possibly data phits. The header phits contain the routing tag, destination node address, and control information. The routing tag identifies a fixed path through the network. The control information is interpreted by the receiving node to determine any local operations that may have to be performed, e.g., read and return a local datum. Depending on the type of packet, additional header information may include the source node address, and memory address at the receiving node. For example, in the figure, a read-request packet is comprised of only header phits while the read response packet contains four 64-bit words. Each word has an additional phit that contains 14 check bits for error correction and detection.

From the example in Figure 2.13 we note that routing information is associated *only* with the header phits (flits) and not with the data flits. As a result, each incoming data flit of a message packet is simply forwarded along the same output channel as the preceding data flit. As a result, the

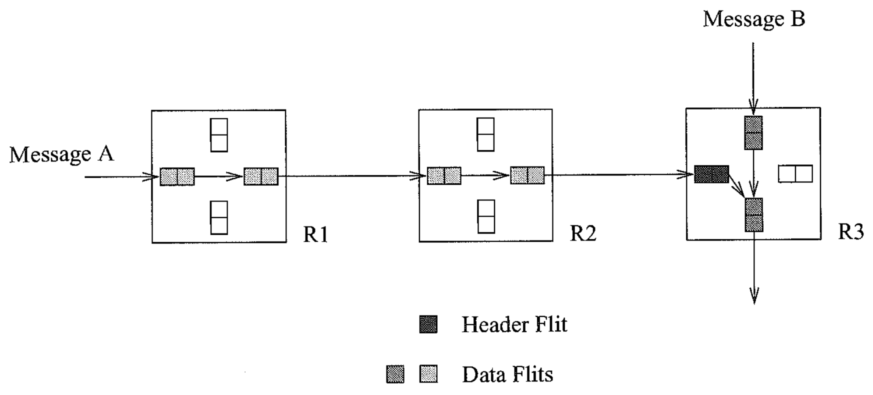


Figure 2.12. An example of a blocked wormhole-switched message.

transmission of distinct messages cannot be interleaved or multiplexed over a physical channel. The message must cross the channel in its entirety before the channel can be used by another message. This is why messages A and B in Figure 2.12 cannot be multiplexed over the physical channel without some additional architectural support.

The base latency of a wormhole-switched message can be computed as follows:

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil \qquad (2.4)$$

This expression assumes flit buffers at the router inputs and outputs. Note that in the absence of contention, VCT and wormhole switching have the same latency. Once the header flit arrives at the destination, the message pipeline cycle time is determined by the maximum of the switch delay and wire delay. For an input-only, or output-only buffered switch this cycle time would be given by the sum of the switch and wire delays.

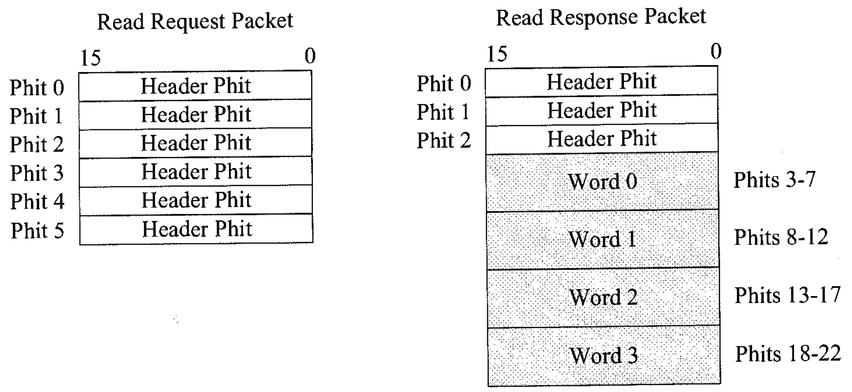


Figure 2.13. Format of wormhole-switched packets in the Cray T3D.

2.3.5 Mad Postman Switching

VCT switching improved the performance of packet switching by enabling pipelined message flow while retaining the ability to buffer complete message packets. Wormhole switching provided further reductions in latency by permitting small buffer VCT so that routing could be completely handled within single-chip routers, therefore providing low latency necessary for tightly coupled parallel processing. This trend toward increased message pipelining is continued with the development of the *mad postman switching* mechanism in an attempt to realize the minimal possible routing latency per node.

The technique is best understood in the context of bit-serial physical channels. Consider a 2-D mesh network with message packets that have a 2-flit header. Routing is dimension-order: messages are first routed completely along dimension 0 and then along dimension 1. The leading header flit contains the destination address of a node in dimension 0. When the message reaches this node, the message is forwarded along dimension 1. The second header flit contains the destination in dimension 1. In VCT and wormhole switching flits cannot be forwarded until the header flits have been received in their entirety at the router. If we had 8-bit flits, transmission of the header flits across a bit-serial physical channel will take 16 cycles. Assuming a 1-cycle delay to select the output channel at each intermediate router, the minimum latency for the header to reach a destination router three links away is 51 cycles. The mad postman attempts to reduce the per-node latency further by pipelining at the bit level. When a header flit starts arriving at a router, it is assumed that the message will be continuing along the same dimension. Therefore header bits are forwarded to the output link in the same dimension as soon as they are received (assuming that the output channel is free). Each bit of the header is also buffered locally. Once the last bit of the first flit of the header has been received, the router can examine this flit and determine if the message should indeed proceed further along this dimension. If it is to proceed along the second dimension, the remainder of the message starting with the second flit of the header is transmitted to the output along the second dimension. If the message has arrived at its destination, it is delivered to the local processor. In essence, the message is first delivered to an output channel and the address is checked later, hence the name of this switching technique. This strategy can work very well in 2-D networks

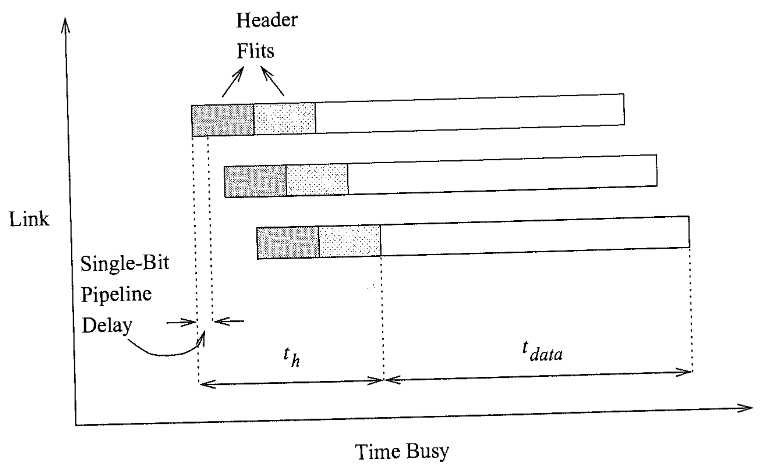


Figure 2.14. Time-space diagram for message transmission using mad postman switching.

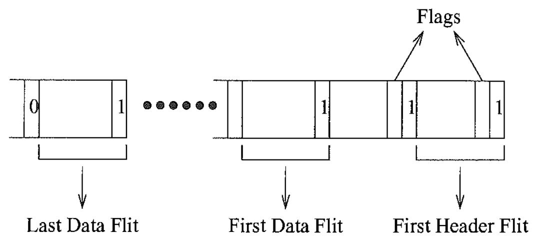


Figure 2.15. An example message format for the mad postman switching technique.

since a message will make at most one turn from one dimension to another and we can encode each dimension offset in 1 header flit. The common case of messages continuing along the same dimension is made very fast. A time-space diagram of a message transmitted over three links using the mad postman switching technique is illustrated in Figure 2.14.

Some constraints must be placed on the organization of the header. An example is shown in Figure 2.15, wherein each dimension offset is encoded in a header flit, and these flits are ordered according to the order of traversal. For example, when the message packet has completely traversed the first dimension the router can start transmitting in the second dimension with the start of the first bit of the second header flit. The first flit has effectively been stripped off the message, but continues to traverse the first dimension. Such a flit is referred to as a *dead address flit.* In a multidimensional network, each time a message changes to a new dimension, a dead flit is generated and the message becomes smaller. At any point if a dead flit is buffered, i.e., blocked by another message packet, it can be detected in the local router and removed.

Let us consider an example of routing in a 4×4, 2-D mesh. In this example the routing header is comprised of 2 flits. Each flit is 3 bits long: a special start bit and 2 bits to identify the destination node in each dimension. The message is pipelined through the network at the bit level. Each input and output buffer is 2 bits deep. Consider the case where a message is being transmitted from node 20 to node 32. Figure 2.16 illustrates the progress and location of the header flits. The message

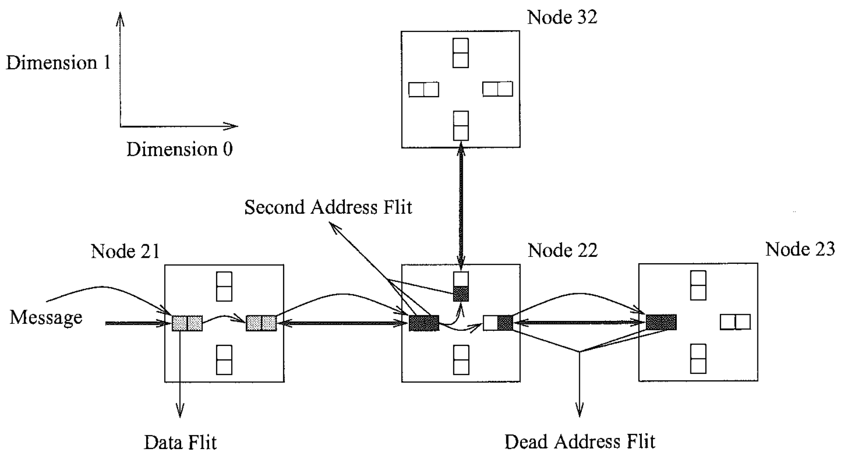


Figure 2.16. Example of routing with mad postman switching and generation of dead address flits.

is transmitted along dimension 0 to node 22 where it is transmitted along dimension 1 to node 32. At node 22, the first flit is pipelined through to the output as it is received. After receiving the third bit, it is determined the message must continue along dimension 1. The first bit of the second header flit is forwarded to the output in dimension 1 as shown in the figure. Note that header flits are stripped off as the message changes dimensions and the message becomes smaller. The dead address flit proceeds along dimension 0 until it can be detected and removed.

For a given number of processors the size of the dead address flit is determined by the number of processors in a dimension. Therefore it follows that for a given number of processors, low-dimension networks will introduce a smaller number of larger dead address flits while higher-dimension networks will introduce a larger number of smaller dead address flits. Initially it would appear that the dead address flits would adversely affect performance until they are removed from the network since they consume physical channel bandwidth. Since message packets will generally be larger than a dead address flit, the probability of a packet being blocked by a dead address flit is very small. It is more likely that a dead address flit will be blocked by a message packet. In this case the local router has an opportunity to detect the dead address flit and remove it from the network. At high loads, we are concerned with dead address flits consuming precious network bandwidth. It is interesting to note that increased blocking in the network will provide more opportunities for routers to remove dead address flits. The greater the congestion, the less likely that a packet will encounter a dead address flit!

By optimistically forwarding the message bit stream to an output channel the routing latency at a node is minimized and full bit-level pipelining can be achieved. Considering again a 2-D mesh with bit-serial physical channels and packets that have two 8-bit header flits, traversing three links, the minimum latency for the header to reach the destination is 18 rather than 51 cycles. In general, the mad postman strategy is useful when it takes multiple cycles for a header to cross a physical channel. In this case latency can be reduced by optimistically forwarding portions of the header onward before the correct output link is actually determined. However, the pin-out constraints of modern routers permit wider flits to be transmitted across a channel in a single cycle. If the header can be transmitted in one cycle, there is little if any advantage to be gained.

The base latency of a message routed using the mad postman switching technique can be computed as follows:

$$\begin{aligned} t_{madpostman} &= t_h + t_{data} \\ t_h &= (t_s + t_w)D + \max(t_s, t_w)W \\ t_{data} &= \max(t_s, t_w)L \end{aligned} \tag{2.5}$$

The above expression makes several assumptions. The first is the use of bit-serial channels which is the most favorable for the mad postman strategy. The routing time t_r is assumed to be equivalent to the switch delay and occurs concurrently with bit transmission, and therefore does not appear in the expression. The term t_h corresponds to the time taken to completely deliver the header.

Let us consider the general case where we do not have bit-serial channels, but rather C bit channels, where $1 < C < W$. Multiple cycles would be required to transfer the header flit across the physical channel. In this case the mad postman switching strategy would realize a base latency of

$$t_{madpostman} = D(t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{W}{C} \right\rceil + \max(t_s, t_w)\left\lceil \frac{L}{C} \right\rceil \tag{2.6}$$

For comparison purposes, in this case the expression for wormhole switching would have been

$$t_{wormhole} = D\left\{t_r + (t_s + t_w)\left\lceil\frac{W}{C}\right\rceil\right\} + \max(t_s, t_w)\left\lceil\frac{L}{C}\right\rceil \tag{2.7}$$

Assuming that the internal and external channel widths are C bits, a header flit (of width W bits) requires $\lceil\frac{W}{C}\rceil$ cycles to cross the channel and the router. This cost is incurred at each intermediate router. When $C = W$, the above expression reduces to the previous expression for wormhole switching with single-flit headers. As larger physical channel widths become feasible in practice, the advantage of the mad postman switching over wormhole switching will diminish.

2.4 Virtual Channels

The preceding switching techniques were described assuming that messages or parts of messages were buffered at the input and output of each physical channel. Buffers are commonly operated as FIFO queues. Therefore once a message occupies a buffer for a channel, no other message can access the physical channel, even if the message is blocked. Alternatively, a physical channel may support several *logical* or *virtual channels* multiplexed across the physical channel. Each unidirectional virtual channel is realized by an independently managed pair of message buffers as illustrated in Figure 2.17. This figure shows two unidirectional virtual channels in each direction across the physical channel. Consider wormhole switching with a message in each virtual channel. Each message can share the physical channel on a flit-by-flit basis. The physical channel protocol must be able to distinguish between the virtual channels using the physical channel. Logically, each virtual channel operates as if each were using a distinct physical channel operating at half the speed. Virtual channels were originally introduced to solve the problem of deadlock in wormhole-switched networks. Deadlock is a network state where no messages can advance because each message requires a channel occupied by another message. This issue is discussed in detail in Chapter 3.

Virtual channels can also be used to improve message latency and network throughput. By allowing messages to share a physical channel, messages can make progress rather than remain blocked. For example, Figure 2.18 shows two messages crossing the physical channel between routers $R1$ and $R2$. With no virtual channels message A will prevent message B from advancing until the transmission of message A has been completed. However, in the figure, there are two single-flit virtual channels multiplexed over each physical channel. By multiplexing the two messages on a flit-by-flit basis, both messages continue to make progress. The rate at which each message is forwarded

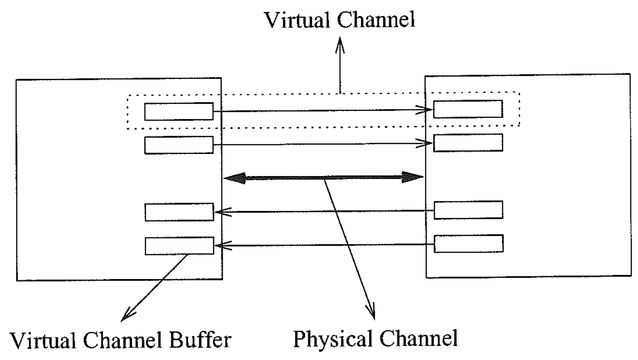


Figure 2.17. Virtual channels.

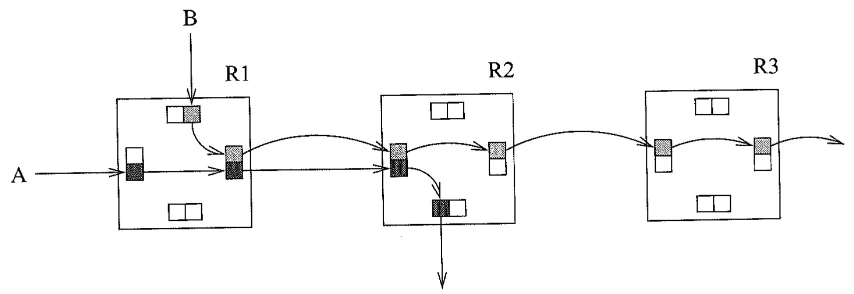


Figure 2.18. An example of the reduction in header blocking delay by using two virtual channels for each physical channel.

is nominally one half the rate achievable when the channel is not shared. In effect, the use of virtual channels decouples the physical channels from message buffers allowing multiple messages to share a physical channel in the same manner that multiple programs may share a central processing unit (CPU). The overall time a message spends blocked at a router waiting for a free channel is reduced leading to an overall reduction in individual message latency. There are two specific cases where such sharing of the physical link bandwidth is particularly beneficial. Consider the case where message A is temporarily blocked downstream from the current node. With an appropriate physical channel flow control protocol, message B can be make use of the full bandwidth of physical channel between the routers. Without virtual channels, both messages would be blocked. Alternatively, consider the case where message A is a very long message relative to message B. Message B can still make progress at half the link speed, and then message A can resume transmission at the full link speed. Studies have shown that message traffic in parallel programs is often bimodal comprised of short (cache lines, control messages) and long messages (data structures) [176].

The approach described in the preceding paragraph does not place any restrictions on the use of the virtual channels. Therefore, when used in this manner these buffers are referred to as *virtual lanes*. Virtual channels were originally introduced as a mechanism for deadlock avoidance in networks with physical cycles, and as such routing restrictions are placed on their use. For example, packets may be prohibited from being transferred between certain classes of virtual channels to prevent cyclic waiting dependencies for buffer space. Thus, in general we have virtual channels that may in turn be comprised of multiple lanes. While the choice of virtual channels at a router may be restricted, it does not matter which lane within a virtual channel is used by a message, although all of the flits within a message will use the same lane within a channel.

We have seen from Section 2.2 that acknowledgment traffic is necessary to regulate the flow of data and to ensure the availability of buffer space on the receiver. Acknowledgments are necessary for each virtual channel or lane, increasing the volume of such traffic across the physical channel. Furthermore, for a fixed amount of buffer space within a router, the size of each virtual channel or lane buffer is now smaller. Therefore the effect of optimizations such as the use of acknowledgments for a block of flits or phits is limited. If physical channel bandwidth is allocated in a demand-driven fashion, the operation of the physical channel now includes the transmission of the virtual channel address to correctly identify the receiving virtual channel, or to indicate which virtual channel has available message buffers.

We can envision continuing to add virtual channels to further reduce the blocking experienced by each message. The result is increased network throughput measured in flits/s, due to increased physical channel utilization. However, each additional virtual channel improves performance by a smaller amount, and the increased channel multiplexing reduces the data rate of individual messages, increasing the message latency. This increase in latency due to data multiplexing will eventually overshadow the reduction in latency due to blocking leading to overall increasing average message latency. An analysis of this phenomena can be found in Chapter 9 which provides detailed performance data to quantify various aspects of network performance.

Increasing the number of virtual channels has a direct impact on router performance through their effect on the achievable hardware cycle time of the router. The link controllers now become more complex since they must support arbitration between multiple virtual channels/lanes for the physical channel, and this arbitration function can be on the critical path for internode delay. The number of inputs and outputs that must be switched at each node is increased, substantially increasing the switch complexity. For a fixed amount of buffer space in a node, how is this buffer space to be allocated among channels, and lanes within a channel? Further, the flow of messages through the router must be coordinated with the allocation of physical channel bandwidth. The increasing complexity of these functions can lead to net increases in internal and external flow control latencies. This increase affects all messages through the routers. Such trade-offs and related issues affecting the design of routers are discussed in detail in Chapter 7 and evaluated in Chapter 9.

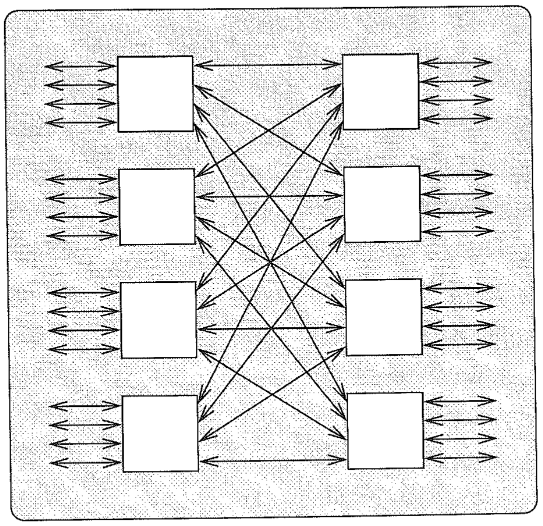
2.5 Hybrid Switching Techniques

The availability and flexibility of virtual channels have led to the development of several hybrid switching techniques. These techniques have been motivated by a desire to combine the advantages of several basic approaches, or have been motivated by the need to optimize performance metrics other than traditional latency and throughput, e.g., fault tolerance and reliability. Some common hybrid switching techniques are presented in this section.

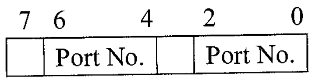
2.5.1 Buffered Wormhole Switching

A switching technique that combines aspects of wormhole switching and packet switching is *buffered wormhole switching* (BWS), proposed and utilized in IBM's Power Parallel SP systems. The switching technique and message formats have been motivated by the interconnection network utilized in the SP systems. This network is a multistage, generalized Omega network using 4×4 crossbar switches with bidirectional links. The system building block is a 16-processor system configured around the two-stage switch with eight crossbar switches as shown in Figure 2.19a. This module is referred to as a frame. Each 4×4 switch uses bidirectional links, and therefore can be viewed as an 8×8 implementation of the router organization shown in Figure 2.1 with the functionality described below.

The basic switching mechanism is wormhole switching. Message packets can be of variable length and up to 255 flits in length. Each flit is 1 byte and is equal to the physical channel width. The first flit of a message contains the length of the message while the following flits contain routing information. Routing is source-based where each routing flit contains the address of the output ports in intermediate switches. There is 1 routing flit for each frame, i.e., for each group of 16 processors. The format of the routing flit is shown in Figure 2.19b. Note that these 4×4 crossbar switches have bidirectional links, and therefore eight input ports and eight output ports. Bits 4–6 are used to select the output port of the first switch in the frame. Bits 0–2 are used to select the output port of the second switch in the frame. Bit 7 is used to determine which field is to be used. It is initially cleared and set by the first switch in the frame. Larger systems are built up as groups of frames. Every frame requires 1 routing flit.



(a)



(b)

Figure 2.19. (a) Organization of the switch used in the IBM Power Series parallel machines. (b) Routing flit format.

As the message is routed through the switches, the corresponding routing flit is discarded, shortening the message. This is similar to the mad postman switching strategy. As long as the path is conflict-free, the message progresses as in wormhole switching with interswitch flow control operating at the flit level. When output channels are available, data flow through the switch is through byte-wide paths through the internal crossbar and to the output port. When messages block, flow control within the switch is organized into 8-flit units referred to as *chunks*. When messages block, chunks are constructed at the input port of a switch, transmitted through 64-bit-wide paths to the a local memory. Subsequently, buffered chunks are transferred to an output port where they are converted to a flit stream for transmission across the physical channel.

When there is a conflict at the output of a routing node, flits are buffered within the switch as chunks. These chunks are buffered in a dynamically allocated central storage. The storage is organized as a linked list for each output, where each element of the list is a message to be transmitted on that output port. Since only the first chunk of a message contains routing information, each message is in turn organized as a list of chunks. Thus, ordering of flits within a message packet is preserved. This type of organization is depicted in Figure 2.20 where two messages are shown queued at an output port. The first message is comprised of 24 flits and the second message is 16 flits long. Messages waiting on each of the other output ports are similarly organized. When an

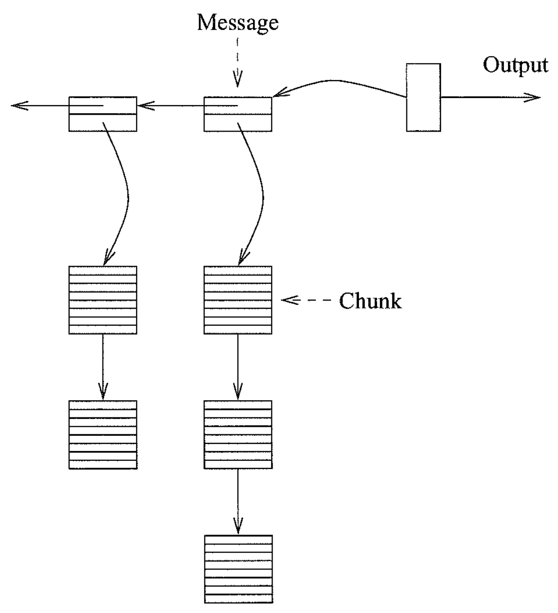


Figure 2.20. Logical organization of the message flit buffers.

output port becomes free, messages are transmitted to the output channel as 64-bit chunks in a single cycle: since the internal datapath and flow control to/from central memory is based on 8-flit chunks. The central storage is dual-ported and can support 128 chunks. A minimum of one chunk is made available for each output port. The remaining chunks are dynamically allocated as necessary. In a single cycle, one input port or one output port can be serviced from the central storage. Thus we see that short messages can be completely buffered.

BWS differs from wormhole switching in that flits are not buffered in place. Rather flits are aggregated and buffered in a local memory within the switch. If the message is small and space is available in the central queue, the input port is released for use by another message even though this message packet remains blocked. In this respect, BWS appears similar to packet switching. BWS differs from packet switching and VCT switching in that flow control is largely at the flit level and when messages are blocked, flow control (within the switch) is at the level of 8-flit chunks. If the central queue were made large enough to ensure that complete messages could always be buffered, the behavior of BWS would approach that of VCT switching.

The base latency of a message routed using BWS is identical to that of wormhole-switched messages.

2.5.2 Pipelined Circuit Switching

In many environments rather than minimizing message latency or maximizing network throughput, the overriding issue is the ability to tolerate the failure of network components such as routers and links. In wormhole switching, header flits containing routing information establish a path through the network from source to destination. Data flits are pipelined through the path immediately following the header flits. If the header cannot progress due to a faulty component, the message is

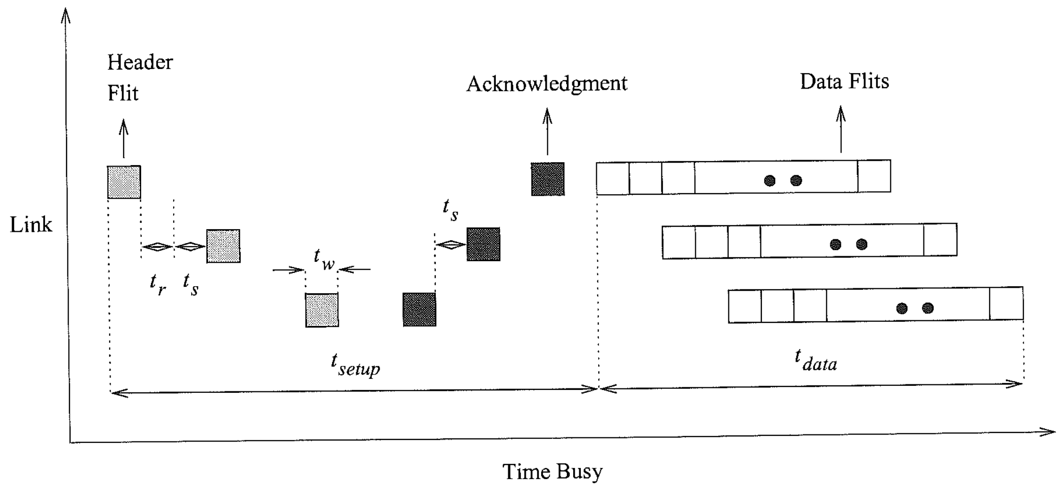
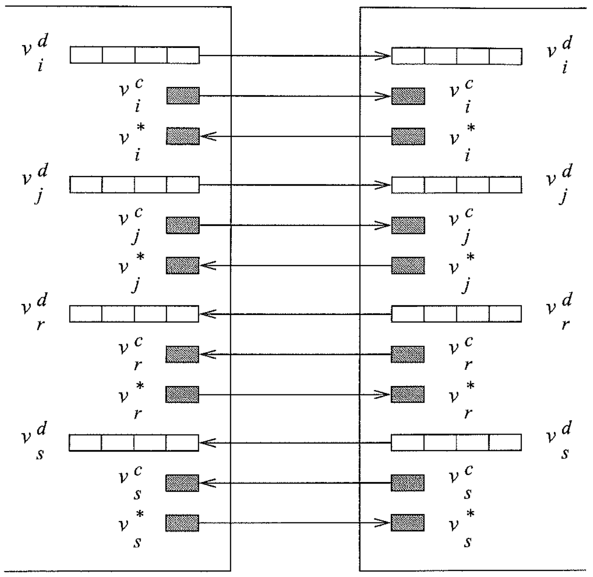


Figure 2.21. Time-space diagram of a message transmitted using PCS.

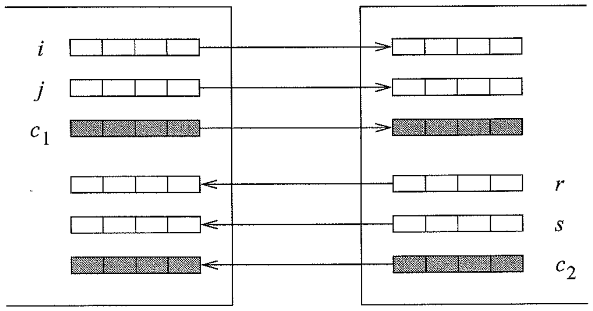
blocked in place indefinitely, holding buffer resources and blocking other messages. This situation can eventually result in a deadlocked configuration of messages. While techniques such as adaptive routing can alleviate the problem, it cannot by itself solve the problem. This has motivated the development of different switching techniques.

Pipelined circuit switching (PCS) combines aspects of circuit switching and wormhole switching. PCS sets up a path before starting data transmission as in circuit switching. Basically, PCS differs from circuit switching in that paths are formed by virtual channels instead of physical channels. In pipelined circuit switching, data flits do not immediately follow the header flits into the network as in wormhole switching. Consequently, increased flexibility is available in routing the header flit. For example, rather than blocking on a faulty output channel at an intermediate router, the header may backtrack to the preceding router and release the previously reserved channel. A new output channel may now be attempted at the preceding router in finding an alternative path to the destination. When the header finally reaches the destination node, an *acknowledgment flit* is transmitted back to the source node. Now data flits can be pipelined over the path just as in wormhole switching. The resilience to component failures is obtained at the expense of larger path setup times. This approach is flexible in that headers can perform a backtracking search of the network, reserving and releasing virtual channels in an attempt to establish a fault-free path to the destination. This technique combines message pipelining from wormhole switching with a more conservative path setup algorithm based on circuit switching techniques. A time-space diagram of a PCS message transmission over three links in the absence of any traffic or failures is shown in Figure 2.21.

Since headers do not block holding channel or buffer resources, routing restrictions are not necessary to avoid deadlock. This increases the probability of finding a path while still avoiding deadlocked configurations of messages. Moreover, reservation of virtual channels by the header does not by itself lead to use of physical channel bandwidth. Therefore, unlike circuit switching, path setup does not lead to excessive blocking of other messages. As a result, multipath networks in conjunction with the flexibility of PCS are good candidates for providing low-latency fault-tolerant performance. For purely performance-driven applications where fault tolerance is not a primary concern, the added overhead of PCS makes wormhole switching the mechanism of choice.



(a)



(b)

Figure 2.22. Virtual channel model for PCS.

In PCS, we distinguish between flits that carry control information, e.g., header flits and acknowledgment flits, and those that carry data. This distinction is supported in the virtual channel model that separates control flit traffic and data flit traffic. A unidirectional virtual channel v_i is composed of a *data channel*, a *corresponding channel*, and a *complementary channel* (v_i^d, v_i^c, v_i^*) and is referred to as a *virtual channel trio*. The router header will traverse v_i^c while subsequent data flits will traverse v_i^d. The complementary channel v_i^* is reserved for use by acknowledgment flits and backtracking header flits. The complementary channel of a trio traverses the physical channel in the direction opposite to that of its associated data channel. The channel model is illustrated in Figure 2.22. There are two virtual channels $v_i(v_r)$ and $v_j(v_s)$ from $R1$ ($R2$) to $R2$ ($R1$). Only

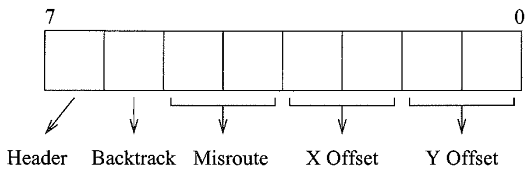


Figure 2.23. Example format of a PCS header.

one message can be in progress over a given data channel. Therefore, compared to existing channel models, this model requires exactly two extra flit buffers for each data channel — one each for the corresponding channel and the complementary channel, respectively. Since control flit traffic is a small percentage of the overall flit traffic, in practice all control channels across a physical link are multiplexed through a single virtual control channel as shown in Figure 2.22a. Thus, compared to the more common use of virtual channels, this model requires one extra virtual channel in each direction between a pair of adjacent routers. For example, channel c_1 in Figure 2.22b corresponds to flit buffers $v_i^c, v_j^c, v_r^*, v_s^*$. The implementation of PCS in the Ariadne router [7] utilized two data channels and one virtual control channel over each physical link.

This separation of control traffic and data traffic is useful in developing fault tolerant routing and distributed fault recovery mechanisms. Such mechanisms are discussed in greater detail in Chapter 6. The Ariadne router [7] is a single-chip PCS router with two virtual channel trios per physical channel. The prototype router had byte-wide physical channels and 8-bit flits. The format of the header flit is shown in Figure 2.23. In this design a single bit distinguished a control flit from a data flit (this only left 7-bit data flits!). A single bit distinguishes between backtracking flits and flits making forward progress. The misroute field keeps track of the number of misrouting steps the header has taken. The maximum number of misroutes that the header can take in this design is 3. Finally, two fields provide X and Y offsets for routing in a 2-D mesh.

The base latency of a pipelined circuit switched message can be computed as follows:

$$\begin{aligned} t_{pcs} &= t_{setup} + t_{data} \\ t_{setup} &= D(t_r + t_s + t_w) + D(t_s + t_w) \\ t_{data} &= D(t_s + t_w) + \max(t_s, t_w)\left(\left\lceil \frac{L}{W} \right\rceil - 1\right) \end{aligned} \tag{2.8}$$

The first term in t_{setup} is the time taken for the header flit to reach the destination. The second term is the time taken for the acknowledgment flit to reach the source node. We then have t_{data} as the time for pipelining the data flits into the destination network interface. The first term is the time for the first data flit to reach the destination. The second term is the time required to receive the remaining flits. The message pipeline cycle time is determined by the maximum of the switch delay and wire delay.

2.5.3 Scouting Switching

Scouting switching is a hybrid message flow control mechanism that can be dynamically configured to provide specific trade-offs between fault tolerance and performance. In PCS the first data flit is injected into the network only after the complete path has been set up. In an attempt to reduce PCS path setup time overhead, in scouting switching the first data flit is constrained to remain at least K links behind the routing header. When $K = 0$, the flow control is equivalent to wormhole switching, while large values can ensure path setup prior to data transmission (if a path exists).

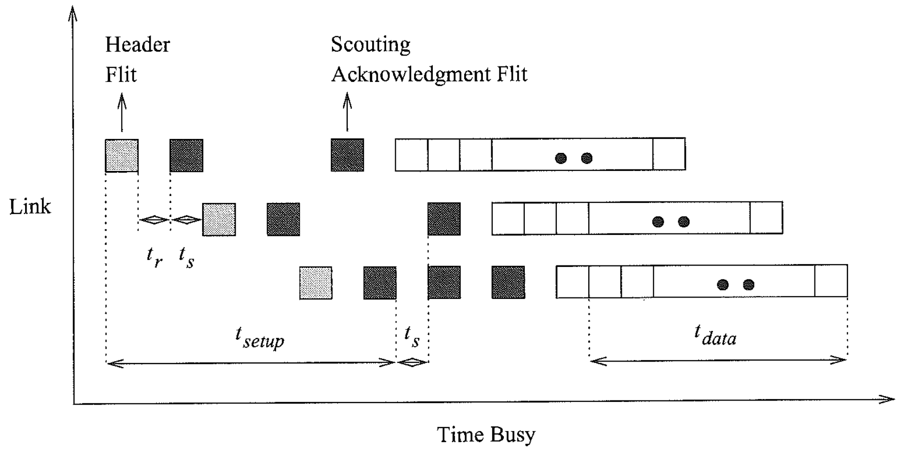


Figure 2.24. Time-space diagram of a message transmitted using scouting switching.

Intermediate values of K permit the data flits to follow the header at distance, while still allowing the header to backtrack if the need arises. Therefore when the header reaches the destination, the first data flit arrives shortly thereafter rather than immediately (as in wormhole switching). Figure 2.24 illustrates a time-space diagram for messages being pipelined over three links using scouting switching ($K = 2$). The parameter, K, is referred to as the *scouting distance* or *probe lead.* Every time a channel is successfully reserved by the routing header, a positive acknowledgment is returned in the opposite direction. As a particular case, positive acknowledgments are continuously transmitted when the routing header has reached the destination node. Associated with each virtual channel is a programmable counter. The counter associated with the virtual channel reserved by a header is incremented when a positive acknowledgment is received, and is decremented when a negative acknowledgment is received. When the value of the counter is equal to K, data flits are allowed to advance. As acknowledgments flow in the direction opposite to the routing header, the gap between the header and the first data flit can grow up to a maximum of $2K - 1$ links while the header is advancing. If the routing header backtracks, a negative acknowledgment is transmitted. For performance reasons, when $K = 0$ no acknowledgments are sent across the channels. In this case, data flits immediately follow the header flit and flow control is equivalent to wormhole switching.

For example, in Figure 2.25 a message is being transmitted between nodes A and G and $K = 2$. The initial path attempted by the header is row first. Data flits remain at least two links behind the header. On encountering faulty output link at node B, the header can backtrack over the previous link. Encountering another faulty link the header can still backtrack one more link to node C. During this time the first data flit remains blocked at node C. From node C it is possible to make progress towards the destination via node D. When the header reaches node F, it is $2K - 1 = 3$ links from the first data flit at node C, and data flits can begin to flow again.

By statically fixing the value of K, we fix the trade-off between network performance (overhead of positive and negative acknowledgment) and fault tolerance (the number of faults that must be tolerated). By dynamically modifying K, we can gain further improvement via run-time trade-offs between fault tolerance and performance. Such configurable flow control protocols are discussed in the context of fault tolerant and reliable routing in Chapter 6.

The base latency of scouting switching can be computed as follows:

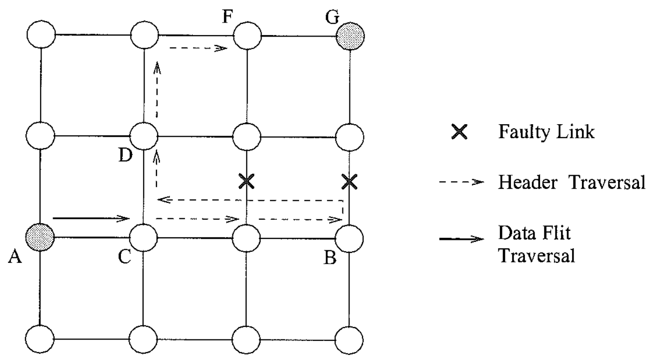


Figure 2.25. An example of fault-tolerant routing enabled by scouting switching.

$$\begin{aligned} t_{scouting} &= t_{setup} + (t_s + t_w)(2K-1) + t_{data} \\ t_{setup} &= D(t_r + t_s + t_w) \\ t_{data} &= \max(t_s, t_w)\left(\lceil \tfrac{L}{W} \rceil - 1\right) \end{aligned} \qquad (2.9)$$

The first term is the time taken for the header flit to reach the destination. The first data flit can be at a maximum of $(2K-1)$ links behind the header. The second term is the time taken for the first data flit to reach the destination. The last term is the time for pipelining the remaining data flits into the destination network interface.

2.6 Optimizing Switching Techniques

The switching techniques described in this chapter are subject to application-specific optimizations to further improve performance and/or reliability. Such optimizations do not fundamentally alter the nature of these techniques but can lead to considerable improvements in performance in specific application domains. For example, consider the overheads experienced in transmitting and receiving a message. The programming interface may be a message-passing library comprised of various send and receive procedures. The concern at this level has been to provide consistent semantics for sending and receiving messages. The source program builds a message in local buffers and transfers control to the operating system via system calls. Data are copied into operating system space where protected device drivers construct message headers, packetize the data, and interact with special-purpose network interfaces to inject data into the network. This overhead is experienced with every message. When message sizes are small, the overhead on a per-message basis can be substantial. There have been several successful efforts to reduce this overhead. Often, hardware support is provided for packetization and network interfaces are becoming tightly coupled with memory and in some cases even the processor registers through memory-mapped techniques and existence on the processor memory bus rather than on the slower I/O buses. Copying to and from system buffers is also being eliminated through the use of message handlers that operate within the user address spaces. Modern machines are now beginning to focus on the design of the interface between the network and memory. These techniques are beneficial regardless of the switching techniques.

Similarly, optimizations within the low-level physical channel flow control protocols benefit most switching techniques. High-speed signaling mechanisms for physical channel flow control affect the interrouter latency. As higher-dimensional networks and wider channels are employed the number of inputs and outputs at a router can become very large. Current packaging technology provides chip carriers with 300–400 pins. These pins can begin to become a scarce resource. One innovative technique to addressing this problem is the use of bidirectional signaling [75, 193]. This technique allows simultaneous signaling between two routers across a single signal line. Thus, full-duplex, bidirectional communication of a single bit between two routers can be realized with one pin (signal) rather than two signals. A logic 1 (0) is transmitted as positive (negative) current. The transmitted signal is the superposition of these two signals. Each transmitter generates a reference signal which is subtracted from the superimposed signal to generate the received signal. The result is a considerable savings over the number of input/output pins required, and consequent reduction in the packaging cost. Such optimizations at the physical level are also clearly independent of and benefit all switching techniques.

Application environments that exhibit locality in interprocessor communication are particularly good candidates for application-specific optimizations. For example, systolic computation makes use of fine-grained, pipelined, parallel transmission of multiple data streams through a fixed communication topology such as multidimensional mesh or hexagonal interconnection topologies. In such cases, it is beneficial to set up interprocessor communication paths once to be shared by many successive data elements (i.e., messages). The Intel iWarp chip was designed to support such systolic communication through *message pathways*: long-lived communication paths [39]. Rather than set up and remove network paths each time data are to be communicated, paths through the network persist for long periods of time. Special messages called *pathway begin markers* are used to reserve virtual channels (referred to as *logical channels* in iWarp) and set up interprocessor communication paths. Messages are periodically injected into the network to use these existing paths utilizing wormhole switching. On completion of the computation, the paths are explicitly removed by other control messages. Unlike conventional wormhole switching, the last flit of the message does not cause the routers to tear down the path. The overhead of preparing a node for message transmission is incurred once, and amortized over all messages that use the path. Such optimizations are possible due to the regular, predictable nature of systolic communication. The basic switching technique is wormhole, but it is applied in a manner to optimize the characteristics of the applications.

Not only fine-grained parallel algorithms exhibit communication locality that can be exploited by switching techniques. Studies of VLSI CAD programs and programs from linear algebra have shown that coarse-grained message-passing applications can also exhibit sufficient communication locality to benefit from long-lived communication paths and justify the design of an enhanced wormhole router [159]. As in the iWarp chip, paths are set up and shared by multiple messages until they are explicitly removed. The basic wormhole switching technique is modified to prevent the path from being removed when the first message has been successfully received at the destination. In this case interprocessor paths can even be shared between applications in a multiprogrammed parallel architecture. Message flow control must avoid deadlock as well as ensuring that reserved paths do not preclude new messages from being injected into the network.

The need for reliable message transmission has also led to proposals for enhancing switching techniques. For example, one way to ensure message delivery is to buffer the message at the source until it can be asserted that the message has been received at the destination. Receipt at the destination can be determined through the use of message acknowledgments. In this case, paths are removed by explicit control signals/messages generated by the source/destination rather than by a part of the message. Alternatively, consider the use of wormhole switching when the number of flits in the message exceeds the number of links, D, between the source and destination nodes. If each router only buffers a single flit, receipt of the header at the destination can be asserted at the source when flit $D+1$ is injected into the network. If messages are short, they can be padded with empty flits so that the number of flits in the message exceeds the distance between the source and destination (padding must also account for buffer space within each router). By keeping track of the number of flits that have been injected into the network, the source router can determine if the header has been received at the destination. Moreover, the source node can determine if the whole message has been received at the destination by injecting $D + 1$ padding flits after the last data flit of the message. Such reliable switching techniques modify the basic wormhole switching mechanisms to include additional flits or control signals, e.g., acknowledgments or padding flits. This particular technique was proposed as *compressionless routing* by its developers [179].

The need to support distinct traffic types also leads to new optimizations of switching techniques [293]. Real-time communication traffic places distinct demands on the performance and behavior of network routers. Such traffic requires guaranteed bounds on latency and throughput. The manner in which messages are buffered and scheduled across physical channels must be able to provide such guarantees on a per-router basis. Such guarantees would be used by higher-level, real-time scheduling algorithms. Packet switching is attractive from this point of view since predictable demands are made on buffer requirements and channel bandwidth at each intermediate router. In contrast, the demands that will be placed on router resources by a message using VCT will vary depending on the load and communication pattern (i.e., is the output link free). Buffering of packets permits the application of priority-based scheduling algorithms and thus provide some control over packet latencies. Wormhole-switched messages use demand-driven scheduling disciplines for accessing physical channel bandwidth and may be blocked across multiple nodes. Demand-driven scheduling and very low buffer requirements work to provide low average latency but high variability and thus poor predictability. Priority-based scheduling of virtual channels is infeasible since a channel may have messages of multiple priorities, and messages may be blocked over multiple links. These properties make it difficult to utilize wormhole switching to support real-time traffic. Rexford and Shin [293] observed that packet switching and wormhole switching made demands on distinct router resources while sharing physical channel bandwidth. Thus, the authors proposed a scheme where virtual channels were partitioned to realize two distinct virtual networks: one packet-switched, and the other wormhole-switched. The two networks share the physical link bandwidth in a controlled manner, thus enabling the network to provide latency and throughput guarantees for real-time traffic, while standard traffic realized low average latency. The switching technique experienced by a message is determined at the source node, based on the traffic type.

We can envision other optimizations that deal with issues such as allocation/deallocation of buffer space within routers, allocation/deallocation of virtual channels, scheduling of virtual channels (equivalently messages) over the physical channel, etc. Some of these optimizations are examined in greater detail in the Exercises section at the end of this chapter.

2.7 A Comparison of Switching Techniques

The evolution of switching techniques was naturally influenced by the need for better performance. VCT switching introduced pipelined message transmission, and wormhole switching further contributed reduced buffer requirements in conjunction with fine-grained pipelining. The mad postman switching technique carried pipelining to the bit level to maximize performance. In packet switching and VCT messages are completely buffered at a node. As a result, the messages consume network bandwidth proportional to the network load. On the other hand, wormhole-switched messages may block occupying buffers and channels across multiple routers, precluding access to the network bandwidth by other messages. Thus, while average message latency can be low, the network saturates at a fraction of the maximum available bandwidth and the variance of message latency can be high.

The use of virtual channels decouples the physical channel from blocked messages, thus reducing the blocking delays experienced by messages and enabling a larger fraction of the available bandwidth to be utilized. However, the increasing multiplexing of multiple messages increases the delay experienced by data flits. Furthermore, multiple virtual channels can increase the flow control latency through the router and across the physical channel, producing upward pressure on average message latency.

The effects of wormhole switching on individual messages can be highly unpredictable. Since buffer requirements are low, contention in the network can substantially increase the latency of a message in parts of the network. Packet switching tends to have more predictable latency characteristics, particularly at low loads since messages are buffered at each node. VCT will operate like wormhole switching at low loads and approximate packet switching at high loads where link contention will force packets to be buffered at each node. Thus, at low loads we expect to see wormhole switching techniques providing superior latency/throughput relative to packet-switched networks, while at high loads we expect to see packet-switched schemes perform better. As expected, the performance of VCT approaches that of wormhole switching at low loads and that of packet switching at high loads. More detailed performance comparisons can be found in Chapter 9.

These switching techniques can be characterized as *optimistic* in the sense that buffer resources and links are allocated as soon as they become available, regardless of the state of progress of the remainder of the message. In contrast, pipelined circuit switching and scouting switching may be characterized as *conservative*. Data flits are transmitted only after it is clear that flits can make forward progress. These flow control protocols are motivated by fault tolerance concerns. BWS seeks to improve the fraction of available bandwidth that can be exploited by wormhole switching by buffering groups of flits.

In packet switching, error detection and retransmission can be performed on a link-by-link basis. Packets may be adaptively routed around faulty regions of the network. When messages are pipelined over several links, error recovery and control becomes complicated. Error detection and retransmission (if feasible) must be performed by higher-level protocols operating between the source and destination, rather than at the level of the physical link. If network routers or links have failed, message progress can be indefinitely halted, with messages occupying buffer and channel resources. This can lead to deadlocked configurations of messages and eventually failure of the network.

2.8 Engineering Issues

Switching techniques have a very strong impact on the performance and behavior of the interconnection network. Performance is more heavily influenced by the switching technique than by the topology or the routing algorithm. Furthermore, true tolerance to faulty network components can only be obtained by using a suitable switching technique. The use of topologies with multiple alternative paths between every source destination pair, and the availability of adaptive routing protocols simply reduces the probability of a message encountering a faulty component. The switching technique determines how messages may recover from faulty components.

Switching techniques also have a considerable influence on the architecture of the router, and as a result, the network performance. For example, consider the magnitude of the improvement in performance that wormhole switching provides over packet switching. First-generation multicomputers such as the Intel iPSC/1 utilized packet switching. The iPSC/1 network had routing times on the order of several milliseconds. In addition, message latency was proportional to the distance traveled by the message. In contrast, modern multicomputers such as the Intel Paragon and the Cray T3D have routing and switch delays on the order of several nanoseconds. Message latency has decreased from a few tens of milliseconds to a few hundreds of nanoseconds. In one decade, latency improved by five orders of magnitude! Obviously, this improvement benefits from advances in VLSI technology. However, VLSI technology only improved performance by one order of magnitude. Network devices were clocked at 10 MHz in the Intel iPSC/1 while the Cray T3D is clocked at 150 MHz.

Pipelined message transfer alone cannot be responsible for this magnitude of performance improvement. What then is the source? An important difference between first-generation multicomputers and current multicomputers is that the routing algorithm is computed in hardware. However, packet switching would still be much slower than wormhole switching even if the routing algorithm were computed in hardware in both cases. Wormhole switching performs flow control at the flit level. This apparently unimportant change considerably reduces the need for buffering space. Small hardware buffers are enough to handle flit propagation across intermediate routers. As a consequence, wormhole routers are small, compact, and fast. Moreover, wormhole routers are able to handle messages of any length. However, packet-switched routers must provide buffering space for full packets, either limiting packet size or necessitating the use of local processor memory for storing packets. Access to local node memory is very expensive in terms of time. Storing packets in node memory not only consumes memory bandwidth, but also the network bandwidth of a node is reduced to a fraction of the memory bandwidth. However, wormhole routers do not store packets in memory. Memory is only accessed for injecting and delivering packets. As a consequence, channel bandwidth can be much higher than in packet-switched routers. Depending on the design of the network interface, channel bandwidth may even exceed memory bandwidth. This is the case for the iWarp chip, in which the processor directly accesses the network through special registers.

Even if the router is able to buffer full packets, the larger packet buffers are slower than flit buffers, increasing the flow control latency through the router and slowing down clock frequency. Furthermore, the use of hardware packet buffers implies a fixed packet size. Variable-sized messages must be partitioned into fixed-size packets. This increases message latency and percentage of network bandwidth devoted to overhead, e.g., processing and transmitting packet headers. The unit of flow control in VCT switching is also a packet. Thus, many of these design considerations are applicable to VCT switching. However, wormhole switching does not require messages to be split into packets. This is one of the reasons why VCT routers have not replaced wormhole routers.

We might expect that the mad postman switching technique may considerably increase performance over wormhole switching. However, mad postman switching can only improve performance if the default output channel selected at an intermediate router has a high probability of being the correct output channel. The highest probabilities occur in low-dimensional networks, e.g., 2-D meshes because messages turn only once. However, for a fixed pin-out on the router chips, low-dimensional networks allow the use of wider data channels. Consequently, a header can be transmitted across a physical channel in a single clock cycle, rendering finer-grained pipelining unnecessary and nullifying any advantage of using the mad postman switching.

In summary, we observe that wormhole switching owes its popularity in part to the fact that it performs flow control at the flit level, requiring only small flit buffers. Messages are not stored in memory when they block, but rather span multiple routers. However, the small buffers produce a short delay, and wormhole routers can be clocked at a very high frequency. The result is very high channel bandwidth, potentially higher than the bandwidth to local memory. Given the current state of the technology, we believe that the most promising approach to increase performance considerably with respect to current interconnection networks consists of defining new switching techniques that take advantage of communication locality, and optimize performance for groups of messages rather than individual messages. Similarly, we believe that the most effective way to offer an architectural support for collective communication, and for fault-tolerant communication, is by designing specific switching techniques. These issues will be explored in Chapters 5 and 6.

2.9 Commented references

Circuit switching has its origin in telephone networks. Packet switching has its origin in data networks for intercomputer communication. The first parallel machines were generally packet- or circuit-switched. The Intel iPSC/1 was packet-switched with message latencies on the order of milliseconds. The Direct Connect Module (DCM) introduced in the later-generation Intel iPSC/2 and iPSC/860 machines [257] employed circuit-switched communication with short messages being transmitted in a manner akin to wormhole switching. The GP 1000 from BBN employed a circuit-switched multistage network. The original Denelcor HEP [190], the MIT Tagged Token Dataflow Machine [12], and the Manchester Dynamic Dataflow Machine [144, 145], were all early machines that utilized a packet-switched interprocessor communication network.

Wormhole switching was introduced in the Torus Routing Chip [77, 78] and the performance for wormhole-switched multidimensional tori was examined in [71]. The latest in the line of machines from Intel, the Paragon [165], utilizes wormhole-switched communication. Other machines that utilize wormhole switching include the Cray T3D [258], the IBM Power Parallel SP series [326, 327], and the Meiko Computing Surface [22]. At the time of this writing, interconnection networks in current generation machines appear to be adopting wormhole switching as the mechanism of choice. While the introduction of VCT switching [172] predates wormhole switching, it is not yet in use in commercially available parallel architectures. The best known implementation of VCT is the Chaos router [188]. The use of virtual channel flow control was introduced in [73]. The Cray T3D [258] and the Cray T3E [312] utilize multiple virtual channels per physical channel.

Mad postman switching was introduced in [167] and has found its way into the implementation of low-cost asynchronous routers. However, with the increasing pin-out and channel width in modern routers, wormhole switching still appears to hold an advantage over the mad postman switching technique. More recently, pipelined circuit switching was proposed as a robust switching mechanism [127] and was subsequently realized in the Ariadne router [7]. Scouting switching [100] and the use of dynamically configurable switching techniques [80] was designed to improve the performance of pipelined circuit switching on message traffic that did not encounter any faulty channels.

A thorough study of router architectures and the development of a cost/performance model for router architectures can be found in [11, 57]. These models provide definitions of critical measures of router performance and enable assessment of the impact of virtual channels, channel signaling speed, message formats, etc. on the flow control latency and router delays.

EXERCISES

2.1 Modify the router model shown in Figure 2.1 to use input buffering only and no virtual channels. Rewrite the expressions for the base latency of wormhole switching and packet switching for this router model.

Solution Figure 2.26 illustrates an input-buffered version of Figure 2.1. In this case, in a single cycle a flit is routed through the switch across a physical channel, and into the input buffer of the next router. When a message arbitrates for the output of the router switch, it simultaneously acquires the output physical channel. In general, the duration of a cycle in an input-buffered switch will be longer than that of a switch that has both buffered inputs and buffered outputs. Similar observations can be made about output-buffered switches.

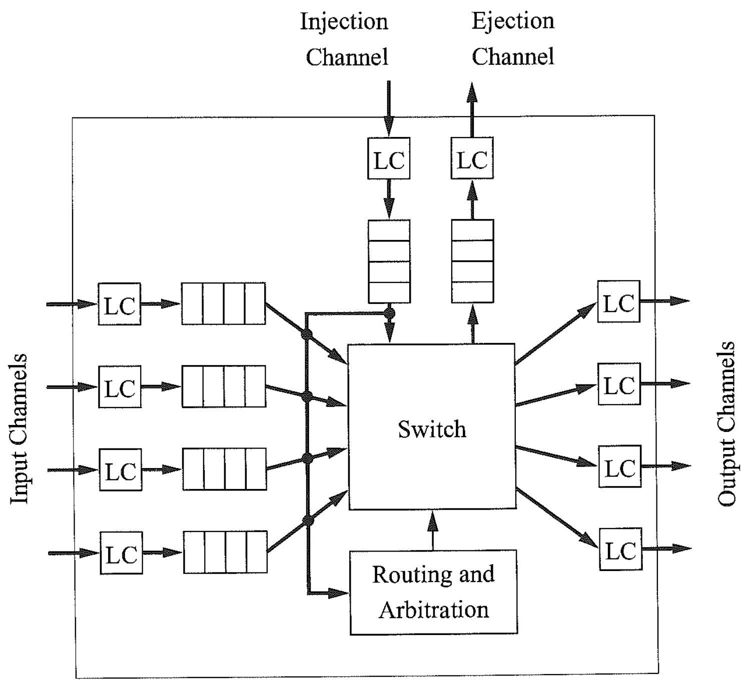


Figure 2.26. Architecture of an input-buffered router with no virtual channels. (LC = Link controller.)

The no-load latency for wormhole switching and packet switching can be rewritten as follows:

$$\begin{aligned} t_{wormhole} &= D(t_r + t_s + t_w) + (t_s + t_w)\left\lceil \frac{L}{W} \right\rceil \\ t_{packet} &= D\left\{t_r + (t_s + t_w)\left\lceil \frac{L+W}{W} \right\rceil\right\} \end{aligned} \quad (2.10)$$

2.2 Assume that the physical channel flow control protocol assigns bandwidth to virtual channels on a strict time-sliced basis rather than a demand-driven basis. Derive a expression for the base latency of a wormhole-switched message in the worst case as a function of the number of virtual channels. Assume that the routers are input-buffered.

Solution Assume that we have V virtual channels. Regardless of network traffic, each message will receive only $\frac{1}{V}$ of the physical channel bandwidth over every link. Therefore the ideal link speed seen by a message is Vt_w seconds. After routing a header, it takes a random number of cycles until the time slice is assigned to the selected virtual channel. In the worst case, it will take V cycles. Therefore the no-load latency becomes

$$t_{wormhole} = D[t_r + V(t_s + t_w)] + V(t_s + t_w)\left\lceil \frac{L}{W} \right\rceil \quad (2.11)$$

PROBLEMS

2.1 In wormhole switching, the last flit or tail flit causes the intermediate routers to release resources such as buffers or links. The time-space diagram used to describe wormhole switching adopts this view. Consider a modification to wormhole switching where the message path is not removed by the tail flit. Rather an acknowledgment is received from the destination, following which a tail flit is transmitted. Draw a time-space diagram of the transmission of a message over three links and write an expression for the number of cycles a link remains reserved for the transmission of this message. What percentage of this time is the link busy assuming no contention for any physical channel along the path? Assume that the routers are input-buffered only.

2.2 One of the advantages of virtual channels is the reduction of header blocking delay. However, this assumes that the physical channel bandwidth is allocated on a fair basis. Suppose we were to schedule virtual channels over the physical channel on a priority basis. Consider a design with two virtual channels. Show how priority scheduling of a low-priority message and a high-priority message over the same physical channel can result in deadlock.

2.3 Physical channel flow control synchronizes the transmission of phits across a channel. Assuming a synchronous channel, show how phits can be streamed across a channel, i.e., a sequence of K phits are transmitted before any acknowledgment is received from the receiver, so that we need only synchronize the transmission and receipt of K phits at a time. What are the requirements for buffer space on the receiver?

2.4 Consider a wormhole-switched network where virtual circuits persist until they are explicitly torn down by control signals or special messages. Draw a time-space diagram of the transmission of three messages over a path three links in length before it is removed by a special control flit injected into the path at the source node.

2.5 The IBM Power Parallel SP-2 represents a balanced design where the rate at which chunks can be transferred to switch memory is eight times the rate at which flits cross a physical channel. A similar observation can be made about the interface between switch memory and the output channels. The maximum message size is 8 chunks. Write an expression for the size of switch memory in flits in terms of the internal data path width (equal to one chunk), number of flits/chunk, and the number of input/output ports. Assume that one flit can be transmitted across the physical channel in one cycle. Validate this expression by instantiating with parameters from the SP-2.

2.6 The main advantage of packet switching over message switching is that several packets can be simultaneously in transit along the path from source to destination. Assuming that all the packets follow the same path, there is no need to add a sequence number to each packet. Draw a time-space diagram of the transmission of a message consisting of four packets over a path three links in length. Assuming that the routers are input-buffered only, compute the optimal number of packets that minimizes the base latency for the transmission of a message of L bits along D channels.

2.7 In the previous exercise, as all the packets follow the same path, assume that packet headers are stripped from all the packets but the first one. Assuming that the routers are input-buffered only, compute the optimal number of packets and the optimal packet size that minimizes the base latency for the transmission of a message of L bits along D channels. Make the analogy between packets without headers and flits in wormhole switching, and determine the optimal flit size.

2.8 Consider a network using wormhole switching, and a node architecture that is able to send and receive up to four packets simultaneously without interference. The start-up latency to initiate a new packet transmission after initiating the previous one is t_s. Assuming that there are four minimal paths between two nodes that are D links apart, compute the base latency to send a message of L bits when the message is split into four sequences of packets, each sequence following a different path. Assume that routers are input-buffered only.

2.9 A hierarchical network topology has channels of two different widths, W and $\frac{W}{2}$, all of them being clocked at the same frequency B. The network uses wormhole switching, and routers have input buffers and output buffers deep enough to avoid filling the buffers in the absence of contention. Internal data paths are W bits wide. Compute the base latency to send a message of L bits across two channels in two cases: (a) the width of the first and second channels are W and $\frac{W}{2}$, respectively; (b) the width of the first and second channels are $\frac{W}{2}$ and W, respectively. Assume that messages are not split into packets.

Chapter 4

Routing Algorithms

In this chapter we study routing algorithms. Routing algorithms establish the path followed by each message or packet. The list of routing algorithms proposed in the literature is almost endless. We clearly cannot do justice to all of these algorithms developed to meet many distinct requirements. We will focus on a representative set of approaches, being biased toward those being used or proposed in modern and future multiprocessor interconnects. Thus, we hope to equip the reader with an understanding of the basic principles that can be used to study the spectrum of existing algorithms. Routing algorithms for wormhole switching are also valid for other switching techniques. Thus, unless explicitly stated, the routing algorithms presented in this chapter are valid for all the switching techniques. Specific proposals for some switching techniques will also be presented. Special emphasis is given to design methodologies, because they provide a simple and structured way to design a wide variety of routing algorithms for different topologies.

Many properties of the interconnection network are a direct consequence of the routing algorithm used. Among these properties we can cite the following:

- *Connectivity.* Ability to route packets from any source node to any destination node. This property was introduced in Chapter 3.

- *Adaptivity.* Ability to route packets through alternative paths in presence of contention or faulty components.

- *Deadlock and livelock freedom.* Ability to guarantee that packets will not block or wander across the network forever. This issue was discussed in depth in Chapter 3.

- *Fault tolerance.* Ability to route packets in presence of faulty components. Although it seems that fault tolerance implies adaptivity, this is not necessarily true. Fault tolerance can be achieved without adaptivity by routing a packet in two or more phases, storing it in some intermediate nodes. Fault tolerance also requires some additional hardware mechanisms, as will be detailed in Chapter 6.

The next section presents a taxonomy of routing algorithms. The most interesting classes in the taxonomy are studied in the remaining sections of this chapter.

4.1 Taxonomy of Routing Algorithms

Figure 4.1 presents a taxonomy of routing algorithms that extends an earlier classification scheme [126]. Routing algorithms can be classified according to several criteria. Those criteria are indicated in the left column in italics. Each row contains the alternative approaches that can be followed for each criterion. Arrows indicate the relations between different approaches. An overview of the taxonomy is presented first, developing it in greater detail later. Routing algorithms can be first classified according to the number of destinations. Packets may have a single destination (*unicast routing*) or multiple destinations (*multicast routing*). Multicast routing will be studied in depth in Chapter 5 and is included here for completeness.

Routing algorithms can also be classified according to the place where routing decisions are taken. Basically, the path can be either established by a centralized controller (*centralized routing*) at the source node prior to packet injection (*source routing*) or determined in a distributed manner while the packet travels across the network (*distributed routing*). Hybrid schemes are also possible. We call these hybrid schemes *multiphase routing*. In multiphase routing, the source node computes some destination nodes. The path between them is established in a distributed manner. The packet may be delivered to all the computed destination nodes (*multicast routing*) or only to the last destination node (*unicast routing*). In this case, intermediate nodes are used to avoid congestion or faults.

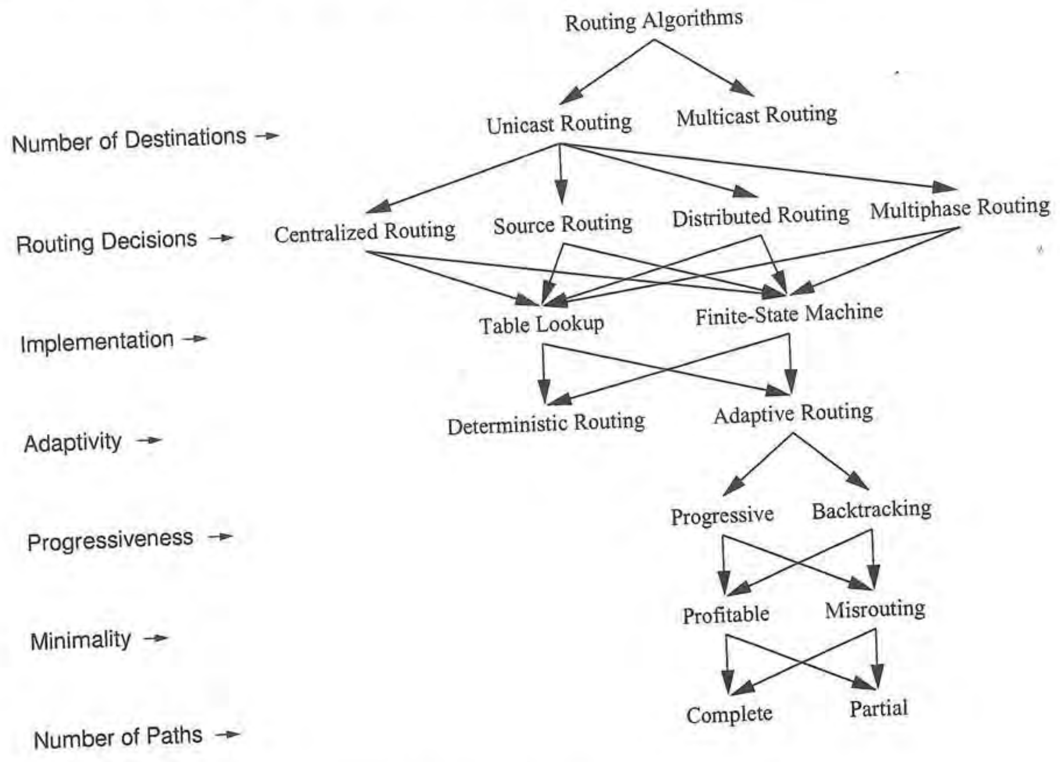


Figure 4.1. A taxonomy for routing protocols.

Routing algorithms can be implemented in different ways. The most interesting ways proposed up to now consist of either looking at a routing table (*table-lookup*) or executing a routing algorithm in software or hardware according to a *finite-state machine*. In both cases, the routing algorithm can be either *deterministic* or *adaptive*. Deterministic routing algorithms always supply the same path between a given source/destination pair. Adaptive routing algorithms use information about network traffic and/or channel status to avoid congested or faulty regions of the network. Routing algorithms designed to tolerate faults will be studied in depth in Chapter 6.

Adaptive routing algorithms can be classified according to their progressiveness as *progressive* or *backtracking*. Progressive routing algorithms move the header forward, reserving a new channel at each routing operation. Backtracking algorithms also allow the header to backtrack, releasing previously reserved channels. Backtracking algorithms are mainly used for fault-tolerant routing.

At a lower level, routing algorithms can be classified according to their minimality as *profitable* or *misrouting*. Profitable routing algorithms only supply channels that bring the packet closer to its destination. They are also referred to as *minimal*. Misrouting algorithms may also supply channels that send the packet away from its destination. They are also referred to as *nonminimal*. At the lowest level, routing algorithms can be classified according to the number of alternative paths as completely adaptive (also known as *fully adaptive*) or *partially adaptive*.

In this chapter we focus on unicast routing algorithms for multiprocessors and multicomputers. Centralized routing requires a central control unit. This is the kind of routing algorithms used in single-instruction multiple-data (SIMD) machines [163]. In source routing, the source node specifies the routing path on the basis of a deadlock-free routing algorithm (either using table-lookup or not). The computed path is stored in the packet header, being used at intermediate nodes to reserve channels. The routing algorithm may use only the addresses of current and destination nodes to compute the path (deterministic routing) or may also use information collected from other nodes about traffic conditions in the network (adaptive routing). Note that collecting information from other nodes may produce a considerable overhead. Additionally, that information may be obsolete. Thus, adaptive source routing is only interesting if traffic conditions change very slowly. Source routing has been mainly used in computer networks with irregular topologies [337]. Myrinet [30], a high-performance LAN supporting irregular topologies, also uses source routing. The first few flits of the packet header contain the address of the switch ports on intermediate switches. See Section 7.2.8 for a description of Myrinet.

Source routing has also been proposed for multicomputer interconnection networks. As traffic conditions may change very quickly in these networks, adaptive source routing is not interesting. Since the header itself must be transmitted through the network, thereby consuming network bandwidth, it is important to minimize header length. One source routing method that achieves this goal is called street-sign routing. The header is analogous to a set of directions given to a driver in a city. Only the names of the streets that the driver must turn on, along with the direction of the turn, are needed. More precisely, packets arriving at an intermediate node have a default output channel in the same dimension and direction as the current channel. For each turn, the header must contain the node address at which the turn will take place and the direction of the turn. Furthermore, this information must be stored in the header according to the order in which nodes are reached. Upon receiving a header flit, the router compares the node address in the flit to the local node address. If they match, the packet either turns or has reached its destination, as specified in the header flit. Otherwise, the packet will be forwarded through the default output channel. Street-sign routing was proposed in iWarp [39]. See Exercise 4.1 for an example of street-sign routing.

For efficiency reasons most hardware routers use distributed routing. Pipelined switching techniques route the header of the packet as soon as it is received at an intermediate node. With distributed routing, the header is very compact. It only requires the destination address and a few

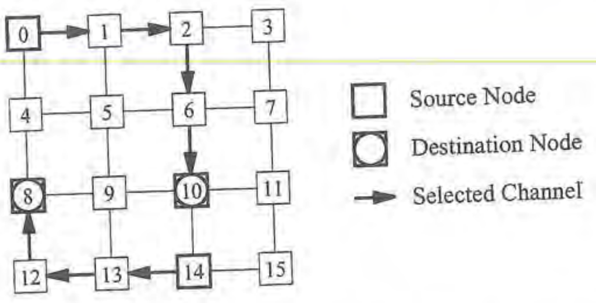


Figure 4.2. Routing example for dimension-order routing on a 2-D mesh.

implementation-dependent control bits. In distributed routing, each intermediate node has to make a routing decision based on the local knowledge of the network. By repeating this process at each intermediate node, the packet should be able to reach its destination. Note that distributed routing algorithms route packets at intermediate nodes without requiring a global knowledge of the network. This can be done because the designer knows the topology of the whole network. Distributed routing algorithms are mainly used in regular topologies so that the same routing algorithm can be used in all the nodes. Almost all commercial topologies can be seen as consisting of several orthogonal dimensions. In these topologies, it is easy to compute the distance between current and destination nodes as the sum of the offsets in all the dimensions. As a consequence, routing decisions are much simpler than in other topologies.

It is also possible to route a packet in two or more phases. In multiphase unicast routing the source node computes an intermediate node. The packet is routed toward this node using distributed routing. On reception of the packet, the intermediate node reinjects the packet into the network, using either another intermediate node or the final destination as the next destination for the packet. An example of multiphase routing algorithm is the random routing algorithm [344]. In this algorithm, the source node computes a random intermediate destination. On reception, the intermediate node forwards the packet toward its final destination. Random routing was proposed to reduce contention by randomizing the paths followed by a set packets transferred between every source-destination pair. However, random routing destroys all the communication locality existing in network traffic. Multiphase routing has also been proposed for fault-tolerant routing [332]. In this case, intermediate destinations are used to break dependencies between channels and avoid deadlocks in the presence of faults. This algorithm will be analyzed in Chapter 6.

Regardless of the place where the routing algorithm is computed, it should be able to deliver the packet to its destination node. In most parallel computers, the designer chooses the topology of the interconnection network. As indicated above, most machines use topologies that can be decomposed into orthogonal dimensions. This is the case for hypercubes, meshes, and tori. In these topologies, it is possible to use simple routing algorithms based on finite-state machines like e-cube (dimension-order routing) [333]. This routing algorithm routes packets by crossing dimensions in increasing order, nullifying the offset in one dimension before routing in the next one. A routing example is shown in Figure 4.2. Note that dimension-order routing can be executed at the source node, storing information about turns (changes of dimension) in the header. This is the street-sign routing algorithm described above. Dimension-order routing can also be executed in a distributed manner. At each intermediate node, the routing algorithm supplies an output channel crossing the lowest dimension for which the offset is not null.

Some manufacturers supply building blocks for parallel computers, so that different topologies can be built by using the same chips [227]. Also, some parallel computers feature reconfigurable topologies, so that the user can change the topology even dynamically at run-time [113]. Finally, some manufacturers allow the final users to build the topology that best fits their needs, allowing the use of irregular topologies [256]. In all these cases, specially for irregular topologies, it is very difficult to derive a routing algorithm based on a finite-state machine. An alternative implementation approach consists of using table-lookup [337]. This is a traditional approach used in computer networks.

An obvious implementation of table-lookup routing is to place a routing table at each node, with the number of entries in the table equal to the number of nodes in the network. Once again, routing can be performed either at the source node or at each intermediate node. In the first case, given a destination node address, the corresponding entry in the table indicates the whole path to reach that node. In the second case, each table entry indicates which outgoing channel should be used to forward the packet toward its destination. However, such an implementation is only practical for very small systems because table size increases linearly with network size. One way to reduce the table size for distributed routing is to define a range of addresses to be associated with each output channel. In this case, each routing table requires only one entry per output channel. Each entry contains an interval of destination addresses, specified by indicating its bounds. This routing technique is called interval routing [199] and has been implemented in the Inmos T9000 transputer and its associated router C104 [227, 228]. An important issue in interval routing is how to assign appropriate labels to nodes so that a single interval per output channel is enough to route all the packets and the resulting routing algorithm is minimal and deadlock-free. See Exercise 4.2 for an example of interval routing.

While establishing a path between source and destination nodes, the routing algorithm may supply a single path (*deterministic routing*). When source routing is used, this path is computed at the source node without considering network traffic. When deterministic routing is implemented in a distributed way, the routing algorithm supplies a single routing choice at each intermediate node, based on current and destination node addresses. In this case, channel status is not considered while computing the output channel to be used. Deterministic routing algorithms for regular topologies are simple. When implemented in hardware using a finite-state machine, routing logic is compact and fast. Most commercially available multicomputers use distributed deterministic routing. Deterministic routing algorithms usually perform well under uniform traffic. However, performance is poor when traffic is not uniform, especially when some pairs of nonneighbor nodes exchange information very frequently.

Alternatively, *adaptive routing* algorithms consider network state while making a decision. As indicated above, it is not interesting to combine source routing and adaptive routing. Thus, in what follows, we only consider distributed routing. Although some authors considered nonlocal information [288], most proposals only use local information for efficiency reasons. As seen in Chapter 3, adaptive routing algorithms can be decomposed into two functions: routing and selection. The *routing function* supplies a set of output channels based on the current node or buffer and the destination node. A selection from this set is made by the *selection function* based on the status of output channels at the current node. This selection is performed in such a way that a free channel (if any) is supplied. As a consequence, adaptive routing algorithms are able to follow alternative paths instead of waiting on busy channels. Thus, these algorithms increase routing flexibility at the expense of a more complex and slower hardware. Several experimental parallel computers use adaptive routing [75]. Also, commercial machines with adaptive routing are being developed [256] or are even available in the market [312]. Example 3.8 describes a distributed adaptive routing algorithm.

Adaptive routing algorithms can be classified as *progressive* or *backtracking*. Progressive routing algorithms move the header forward, reserving a new channel at each routing operation. Backtracking algorithms allow the header to backtrack, releasing previously reserved channels. Backtracking algorithms systematically search the network, backtracking as needed, using history information to ensure that no path is searched more than once. Note that in most switching techniques, data immediately follow the packet header. In these switching techniques, backtracking is not possible without a very complex hardware support. However, limited backtracking (one channel) is possible with some hardware support [181]. On the other hand, pipelined circuit switching is very well suited for backtracking because data flits do not follow the header immediately, giving more freedom to the header to search the network. Backtracking algorithms are mainly used for fault-tolerant routing, as will be seen in Chapter 6.

At a lower level, routing algorithms can be classified as *profitable* or *misrouting*. Profitable routing algorithms only supply channels that bring the packet closer to its destination. Misrouting algorithms may also supply channels that send the packet away from its destination. Misrouting algorithms are based on an optimistic view of the network: taking an unprofitable channel is likely to bring the header to another set of profitable channels that will allow further progress to the destination. Although misrouting algorithms are more flexible, they usually consume more network resources. As a consequence, misrouting algorithms usually exhibit a lower performance when combined with pipelined switching techniques. Also, misrouting algorithms may suffer from livelock, as seen in Chapter 3. Misrouting algorithms are usually proposed for fault-tolerant routing because they are able to find alternative paths when all the minimal paths are faulty. These algorithms will also be studied in Chapter 6.

At the lowest level, routing algorithms can be completely adaptive (also known as *fully adaptive*) or *partially adaptive*. A fully adaptive algorithm can use all the physical paths in its class. For example, a profitable algorithm that is fully adaptive is able to choose among all the minimal paths available in the network. These algorithms are also called *fully adaptive minimal* routing algorithms. It should be noted that although all the physical paths are available, a given routing algorithm may restrict the use of virtual channels in order to avoid deadlock. A routing algorithm that maximizes the number of routing options while avoiding deadlock is referred to as *maximally adaptive*. An even higher flexibility in the use of virtual channels can be achieved by using deadlock recovery techniques. In this case, there is no restriction on the use of virtual channels, and the corresponding routing algorithm is referred to as *true fully adaptive*. A completely adaptive backtracking algorithm is also called *exhaustive*. Partially adaptive algorithms are only able to use a subset of the paths in their class.

Note that deterministic routing algorithms should be progressive and profitable. Backtracking makes no sense because the same path will be reserved again. Also, misrouting is not interesting because some bandwidth is wasted without any benefit.

This chapter is organized as follows. Section 4.2 studies some deterministic routing algorithms as well as a basic design methodology. Section 4.3 presents some partially adaptive routing algorithms and a design methodology. Section 4.4 analyzes fully adaptive routing algorithms and their evolution, also presenting design methodologies. Section 4.5 describes some routing algorithms that maximize adaptivity or minimize the routing resources required for fully adaptive routing. Section 4.6 presents some nonminimal routing algorithms. Section 4.7 describes some backtracking algorithms. As backtracking algorithms have interesting properties for fault-tolerant routing, these algorithms will also be analyzed in Chapter 6. Sections 4.8 and 4.9 study some routing algorithms for switch-based networks, focusing on multistage interconnection networks and irregular topologies, respectively. Finally, Section 4.10 presents several selection functions as well as some resource allocation policies. The chapter ends with some engineering issues and commented references.

4.2 Deterministic Routing Algorithms

Deterministic routing algorithms establish the path as a function of the destination address, always supplying the same path between every pair of nodes. Deterministic routing is distinguished from *oblivious* routing. Although both concepts are sometimes considered to be identical, in the latter the routing decision is independent of (i.e., oblivious to) the state of the network. However, the choice is not necessarily deterministic. For example, a routing table may include several options for an output channel based on the destination address. A specific option may be selected randomly, cyclically or in some other manner that is independent of the state of the network. A deterministic routing algorithm will always provide the same output channel for the same destination. While deterministic algorithms are oblivious, the converse is not necessarily true.

Deterministic routing became very popular when wormhole switching was invented [78]. Wormhole switching requires very small buffers. Wormhole routers are compact and fast. However, pipelining does not work efficiently if one of the stages is much slower than the remaining stages. Thus, wormhole routers have the routing algorithm implemented in hardware. It is not surprising that designers chose the simplest routing algorithms in order to keep routing hardware as compact and fast as possible. Most commercial multicomputers (Intel Paragon [165], Cray T3D [174], nCUBE-2/3 [247]) and experimental multiprocessors (Stanford DASH [203], MIT J-Machine [255]) use deterministic routing.

In this section, we present the most popular deterministic routing algorithms as well as a design methodology. Obviously, the most popular routing algorithms are the simplest ones. Some topologies can be decomposed into several orthogonal dimensions. This is the case for hypercubes, meshes, and tori. In these topologies, it is easy to compute the distance between current and destination nodes as the sum of the offsets in all the dimensions. Progressive routing algorithms will reduce one of those offsets in each routing step. The simplest progressive routing algorithm consists of reducing an offset to zero before considering the offset in the next dimension. This routing algorithm is known as dimension-order routing. This routing algorithm routes packets by crossing dimensions in strictly increasing (or decreasing) order, reducing to zero the offset in one dimension before routing in the next one.

For n-dimensional meshes and hypercubes, dimension-order routing produces deadlock-free routing algorithms. These algorithms are very popular and receive several names, like XY routing (for 2-D mesh) or e-cube (for hypercubes) [333]. These algorithms are described in Figures 4.3 and 4.4, respectively, where $FirstOne()$ is a function that returns the position of the first bit set to one, and $Internal$ is the channel connecting to the local node. Although these algorithms assume that the packet header carries the absolute address of the destination node, the first few sentences in each algorithm compute the offset from the current node to the destination node. This offset is the value carried by the header when relative addressing is used. So, the remaining sentences in each algorithm describe the operations for routing using relative addressing. Note that relative addressing would also require updating the header at each intermediate node. Exercises 3.1 and 4.3 show that the channel dependency graphs for dimension-order routing in n-dimensional meshes and hypercubes are acyclic. However, the channel dependency graph for tori has cycles. This topology was analyzed by Dally and Seitz [78], who proposed a design methodology for deadlock-free deterministic routing algorithms.

The methodology starts by considering some connected routing function and its channel dependency graph D. If it is not acyclic, routing is restricted by removing arcs from the channel dependency graph D to make it acyclic. If it is not possible to make D acyclic without disconnecting the routing function, arcs can be added to D by splitting physical channels into a set of virtual channels. As proposed in [78], this methodology establishes a total order among virtual channels,

```
Algorithm: XY Routing for 2-D Meshes
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
   $Xoffset := Xdest - Xcurrent$;
   $Yoffset := Ydest - Ycurrent$;
   if $Xoffset < 0$ then
      $Channel := X-$;
   endif
   if $Xoffset > 0$ then
      $Channel := X+$;
   endif
   if $Xoffset = 0$ and $Yoffset < 0$ then
      $Channel := Y-$;
   endif
   if $Xoffset = 0$ and $Yoffset > 0$ then
      $Channel := Y+$;
   endif
   if $Xoffset = 0$ and $Yoffset = 0$ then
      $Channel := Internal$;
   endif
```

Figure 4.3. The *XY* routing algorithm for 2-D meshes.

```
Algorithm: Dimension-Order Routing for Hypercubes
Inputs: Addresses of current node $Current$
        and destination node $Dest$
Output: Selected output $Channel$
Procedure:
   $offset := Current \oplus Dest$;
   if $offset = 0$ then
      $Channel := Internal$;
   else
      $Channel := \text{FirstOne}(offset)$;
   endif
```

Figure 4.4. The dimension-order routing algorithm for hypercubes.

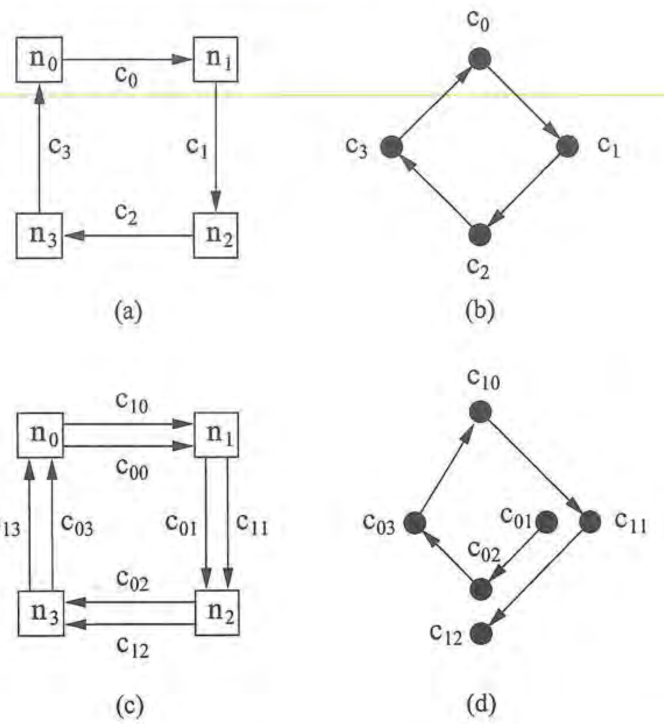


Figure 4.5. Unidirectional rings and their channel dependency graphs.

labeling them accordingly. Every time a cycle is broken by splitting a physical channel into two virtual channels, a new channel index is introduced to establish the ordering between virtual channels. Also, every time a cycle is broken by adding a virtual channel to each physical channel, the new set of virtual channels is assigned a different value for the corresponding index. The next example shows the application of this methodology to unidirectional rings and k-ary n-cubes.

Example 4.1

Consider a unidirectional ring with four nodes denoted $n_i, i = \{0, 1, 2, 3\}$ and a unidirectional channel connecting each pair of adjacent nodes. Let $c_i, i = \{0, 1, 2, 3\}$ be the outgoing channel from node n_i. In this case, it is easy to define a connected routing function. It can be stated as follows: If the current node n_i is equal to the destination node n_j, store the packet. Otherwise, use c_i, $\forall j \neq i$. Figure 4.5a shows the network. Figure 4.5b shows that the channel dependency graph for this routing function contains a cycle. Thus, every physical channel c_i is split into two virtual channels, c_{0i} and c_{1i}, as shown in Figure 4.5c. Virtual channels are ordered according to their indices. The routing function is redefined in such a way that virtual channels are used in strictly increasing order. The new routing function can be stated as follows: If the current node n_i is equal to the destination node n_j, store the packet. Otherwise, use c_{0i}, if $j < i$ or c_{1i}, if $j > i$. Figure 4.5d shows the channel dependency graph

```
Algorithm: Dimension-Order Routing for Unidirectional 2-D Tori
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ then
    $Channel := c_{00}$;
  endif
  if $Xoffset > 0$ then
    $Channel := c_{01}$;
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
    $Channel := c_{10}$;
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    $Channel := c_{11}$;
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
    $Channel := Internal$;
  endif
```

Figure 4.6. The dimension-order routing algorithm for unidirectional 2-D tori.

for this routing function. As can be seen, the cycle has been removed because after using channel c_{03}, node n_0 is reached. Thus, all the destinations have a higher index than n_0, and it is not possible to request c_{00}. Note that channels c_{00} and c_{13} are not in the graph because they are never used.

It is possible to extend the routing function for unidirectional rings so that it can be used for unidirectional k-ary n-cubes. As above, each physical channel is split into two virtual channels. Additionally, a new index is added to each virtual channel. Channels are labeled as c_{dvi}, where $d, d = \{0, \ldots, n-1\}$ is the dimension traversed by the channel, $v, v = \{0, 1\}$ indicates the virtual channel, and $i, i = \{0, \ldots, k-1\}$ indicates the position inside the corresponding ring. The routing function routes packets in increasing dimension order. Inside each dimension, the routing function for rings is used. It is easy to see that this routing function routes packets in strictly increasing order of channel indices. Figure 4.6 shows the dimension-order routing algorithm for unidirectional k-ary 2-cubes. Note that the third subindex of each channel is not indicated because it is only required to distinguish between channels from different routers. So, there is no ambiguity.

Although dimension-order routing is usually implemented in a distributed way using a finite-state machine, it can also be implemented using source routing and distributed table-lookup. See Exercises 4.1 and 4.2 to see how dimension-order routing can be implemented using source routing (street-sign routing) and table-lookup (interval routing), respectively.

Finally, as will be seen in Chapter 7, dimension-order routing is very simple to implement in hardware. Additionally, switches can be decomposed into smaller and faster switches (one for each dimension), thus increasing the speed.

4.3 Partially Adaptive Algorithms

Several partially adaptive routing algorithms have been proposed. Partially adaptive routing algorithms represent a trade-off between flexibility and cost. They try to approach the flexibility of fully adaptive routing at the expense of a moderate increase in complexity with respect to deterministic routing. Most partially adaptive algorithms proposed up to now rely upon the absence of cyclic dependencies between channels to avoid deadlock. Some proposals aim at maximizing adaptivity without increasing the resources required to avoid deadlocks. Other proposals try to minimize the resources needed to achieve a given level of adaptivity.

4.3.1 Planar-Adaptive Routing

Planar-adaptive routing aims at minimizing the resources needed to achieve a given level of adaptivity. It has been proposed by Chien and Kim for n-dimensional meshes and hypercubes [58]. The idea in planar-adaptive routing is to provide adaptivity in only two dimensions at a time. Thus, a packet is routed adaptively in a series of 2-D planes. Routing dimensions change as the packet advances toward its destination.

Figure 4.7 shows how planar-adaptive routing works. A fully adaptive routing algorithm allows a packet to be routed in the m-dimensional subcube defined by the current and destination nodes, as shown in Figure 4.7a for three dimensions. Planar-adaptive routing restricts packets to be routed in plane A_0, then moving to plane A_1, and so on. This is depicted in Figures 4.7b and 4.7c for three and four dimensions, respectively. All the paths within each plane are allowed. The number of paths in a plane depends on the offsets in the corresponding dimensions.

Each plane A_i is formed by two dimensions, d_i and d_{i+1}. There are a total of $(n-1)$ adaptive planes. The order of dimensions is arbitrary. However, it is important to note that planes A_i and A_{i+1} share dimension d_{i+1}. If the offset in dimension d_i is reduced to zero, then routing can be immediately shifted to plane A_{i+1}. If the offset in dimension d_{i+1} is reduced to zero while routing in plane A_i, no adaptivity will be available while routing in plane A_{i+1}. In this case, plane A_{i+1} can be skipped. Moreover, if in plane A_i, the offset in dimension d_{i+1} is reduced to zero first, routing

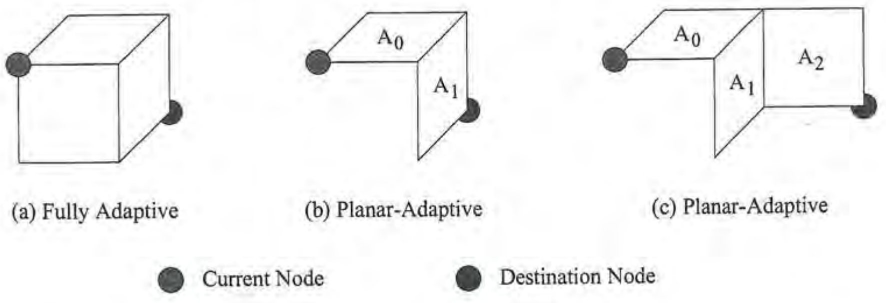


Figure 4.7. Allowed paths in fully adaptive and planar-adaptive routing.

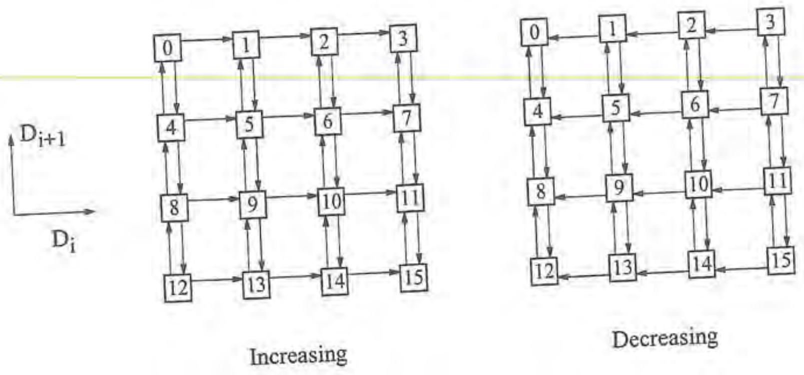


Figure 4.8. Increasing and decreasing networks in plane A_i for planar-adaptive routing.

continues in dimension d_i exclusively, until the offset in dimension d_i is reduced to zero. Thus, in order to offer alternative routing choices for as long as possible, a higher priority is given to channels in dimension d_i while routing in plane A_i.

As defined, planar-adaptive routing requires three virtual channels per physical channel to avoid deadlocks in meshes and six virtual channels to avoid deadlocks in tori. In what follows, we analyze meshes in more detail. Channels in the first and last dimension need only one and two virtual channels, respectively. Let $d_{i,j}$ be the set of virtual channels j crossing dimension i of the network. This set can be decomposed into two subsets, one in the positive direction and one in the negative direction. Let $d_{i,j}+$ and $d_{i,j}-$ denote the positive and negative direction channels, respectively.

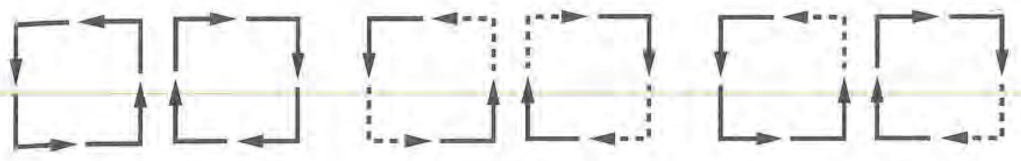
Each plane A_i is defined as the combination of several sets of virtual channels:

$$A_i = d_{i,2} + d_{i+1,0} + d_{i+1,1}$$

In order to avoid deadlock, the set of virtual channels in A_i is divided into two classes: *increasing* and *decreasing* networks. The increasing network is formed by $d_{i,2}+$ and $d_{i+1,0}$ channels. The decreasing network is formed by $d_{i,2}-$ and $d_{i+1,1}$ channels (Figure 4.8). Packets crossing dimension d_i in the positive direction are routed in the increasing network of plane A_i. Similarly, packets crossing dimension d_i in the negative direction are routed in the decreasing network of A_i. As there is no coupling between increasing and decreasing networks of A_i, and planes are crossed in sequence, it is easy to see that there are no cyclic dependencies between channels. Thus, planar-adaptive routing is deadlock-free.

4.3.2 Turn Model

The turn model proposed by Glass and Ni [130] provides a systematic approach to the development of partially adaptive routing algorithms, both minimal and nonminimal, for a given network. As shown in Figure 3.2, deadlock occurs because the packet routes contain turns that form a cycle. As indicated in Chapter 3, deadlock cannot occur if there is not any cyclic dependency between channels. In many topologies, channels are grouped into dimensions. Moving from one dimension to another one produces a turn in the packet route. Changing direction without moving to another dimension can be considered as a 180-degree turn. Also, when physical channels are split into virtual channels, moving from one virtual channel to another one in the same dimension and direction can be



(a) Abstract Cycles in 2-D Mesh. (b) Four Turns (Solid Arrows) Allowed in XY Routing. (c) Six Turns (Solid Arrows) Allowed in West-First Routing.

Figure 4.9. An illustration of the turn model in 2-D mesh.

considered as a 0-degree turn. Turns can be combined into cycles. The fundamental concept behind the turn model is to prohibit the smallest number of turns such that cycles are prevented. Thus, deadlock can be avoided by prohibiting just enough turns to break all the cycles. The following six steps can be used to develop maximally adaptive routing algorithms for n-dimensional meshes and k-ary n-cubes:

1. Classify channels according to the directions in which they route packets.
2. Identify the turns that occur between one direction and another.
3. Identify the simple cycles that these turns can form.
4. Prohibit one turn in each cycle.
5. In the case of k-ary n-cubes, incorporate as many turns as possible that involve wraparound channels, without reintroducing cycles.
6. Add 0-degree and 180-degree turns without reintroducing cycles. These turns are needed if there are multiple channels in the same direction and for nonminimal routing algorithms.

In order to illustrate the use of the turn model, we may consider the case of a 2-D mesh. There are eight possible turns and two possible abstract cycles, as shown in Figure 4.9a. The deterministic XY routing algorithm prevents deadlock by prohibiting four of the turns, as shown in Figure 4.9b. The remaining four turns cannot form a cycle, but neither do they allow any adaptiveness.

However, prohibiting fewer than four turns can still prevent cycles. In fact, for a 2-D mesh, only two turns need to be prohibited. Figure 4.9c shows six turns allowed, suggesting the corresponding *west-first routing algorithm*: route a packet first west, if necessary, and then adaptively south, east, and north. The two turns prohibited in Figure 4.9c are the two turns to the west. Therefore, in order to travel west, a packet must begin in that direction. Figure 4.10 shows the minimal west-first routing algorithm for 2-D meshes, where *Select*() is the selection function defined in Section 3.1.2. This function returns a free channel (if any) from the set of channels passed as parameters. See Exercise 4.5 for a nonminimal version of this algorithm. Three example paths for the west-first algorithm are shown in Figure 4.11. The channels marked as unavailable are either faulty or are being used by other packets. One of the paths shown is minimal, while the other two paths are nonminimal, resulting from routing around unavailable channels. Because cycles are avoided, west-first routing is deadlock-free. For minimal routing, the algorithm is fully adaptive if the destination is on the right-hand side (east) of the source; otherwise, it is deterministic. If nonminimal routing is allowed, the algorithm is adaptive in either case. However, it is not fully adaptive.

```
Algorithm: Minimal West-First Algorithm for 2-D Meshes
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ then
    $Channel := X-$;
  endif
  if $Xoffset > 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($X+, Y-$);
  endif
  if $Xoffset > 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($X+, Y+$);
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
    $Channel := X+$;
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
    $Channel := Y-$;
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    $Channel := Y+$;
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
    $Channel := Internal$;
  endif
```

Figure 4.10. The minimal west-first routing algorithm for 2-D meshes.

There are other ways to select six turns so as to prohibit cycles. However, the selection of the two prohibited turns may not be arbitrary [130]. If turns are prohibited as in Figure 4.12, deadlock is still possible. Figure 4.12a shows that the three remaining left turns are equivalent to the prohibited right turn, and Figure 4.12b shows that the three remaining right turns are equivalent to the prohibited left turn. Figure 4.12c illustrates how cycles may still occur. Of the 16 different ways to prohibit two turns, 12 prevent deadlock and only 3 are unique if symmetry is taken into account. These three combinations correspond to the west-first, north-last, and negative-first routing algorithms. The *north-last routing* algorithm does not allow turns from north to east or from north to west. The *negative-first routing* algorithm does not allow turns from north to west and from east to south.

In addition to 2-D mesh networks, the turn model can be used to develop partially adaptive routing algorithms for n-dimensional meshes, for k-ary n-cubes and for hypercubes [130]. By applying the turn model to the hypercube, an adaptive routing algorithm, namely, *P-cube routing*, can be developed. Let $s = s_{n-1}s_{n-2}\dots s_0$ and $d = d_{n-1}d_{n-2}\dots d_0$ be the source and destination nodes, respectively, in a binary n-cube. The set E consists of all the dimension numbers in which s and

ı of the two
llock is still
ibited right
: prohibited
to prohibit
ɪunt. These
algorithms.
;o west. The
to south.
laptive rout-
By applying
uting, can be
ıation nodes,
which s and

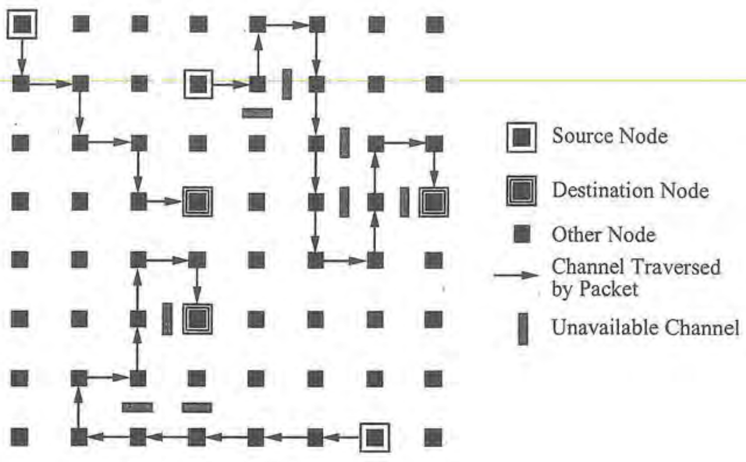


Figure 4.11. Examples of west-first routing in an 8×8 2-D mesh.

d differ. The size of E is the Hamming distance between s and d. Thus, $i \in E$ if $s_i \neq d_i$. E is divided into two disjoint subsets, E_0 and E_1, where $i \in E_0$ if $s_i = 0$ and $d_i = 1$, and $j \in E_1$ if $s_j = 1$ and $d_j = 0$. The fundamental concept of P-cube routing is to divide the routing selection into two *phases*. In the first phase, a packet is routed through the dimensions in E_0 in any order. In the second phase, the packet is routed through the dimensions in E_1 in any order. If E_0 is empty, then the packet can be routed through any dimension in E_1. Figure 4.13 shows the pseudocode for the P-cube routing algorithm. The **for** loop computes the sets E_0 and E_1. This can be done at each intermediate node as shown or at the source node. In the latter case, the selected channel should be removed from the corresponding set. The *digit*() function computes the value of the digit in the given position.

Note that cycles cannot exist only traversing dimensions in E_0 since they represent channels from a node to a higher-numbered node. In a cycle, at least one channel must be from a node to a lower-numbered node. For similar reasons, packets cannot form cycles by only traversing dimensions

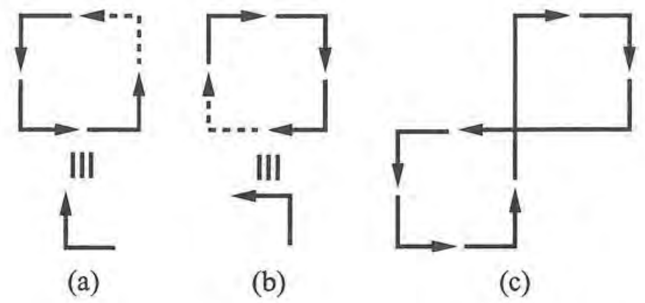


Figure 4.12. Six turns that complete the abstract cycles.

```
Algorithm: Minimal P-Cube Routing for Hypercubes
Inputs: Addresses of current node $Current$
        and destination node $Dest$
Output: Selected output $Channel$
Procedure:
  $E_0 := \{\ \}$;
  $E_1 := \{\ \}$;
  for $i := 0$ to $n-1$ do
    if digit($Current, i$) $= 0$ and digit($Dest, i$) $= 1$ then
      $E_0 := E_0 + \{i\}$;
    endif
    if digit($Current, i$) $= 1$ and digit($Dest, i$) $= 0$ then
      $E_1 := E_1 + \{i\}$;
    endif
  end;
  if $E_0 \neq \{\ \}$ then
    $Channel :=$ Select($E_0$);
  endif
  if $E_0 = \{\ \}$ and $E_1 \neq \{\ \}$ then
    $Channel :=$ Select($E_1$);
  endif
  if $E_0 = \{\ \}$ and $E_1 = \{\ \}$ then
    $Channel := Internal$;
  endif
```

Figure 4.13. The minimal P-cube routing algorithm for hypercubes.

in set E_1. Finally, since packets only use dimensions in E_1 after traversing all of the dimensions in E_0, deadlock freedom is preserved. In effect, the algorithm prohibits turns from dimensions in E_1 to dimensions in E_0. This is sufficient to prevent cycles. By partitioning the set of dimensions to be traversed in other ways that preserve the acyclic properties, one can derive other variants of this algorithm.

Let the sizes of E, E_0, and E_1 be k, k_0, and k_1, respectively; $k = k_0 + k_1$. There exist $k!$ shortest paths between s and d. Using P-cube routing, a packet may be routed through any of $(k_0!)(k_1!)$ of those shortest paths. A similar algorithm was proposed in [184], however, the P-cube routing algorithm can be systematically generalized to handle nonminimal routing as well [130].

4.4 Fully Adaptive Algorithms

This section presents several methodologies for the design of fully adaptive routing algorithms as well as some specific routing algorithms. Where possible, the presentation follows a chronological order, showing the evolution of design methodologies. First, we present some algorithms developed for computer networks in Section 4.4.1, as well as a methodology to adapt those algorithms for wormhole switching in Section 4.4.2. Then we present some methodologies based on the concept of virtual network in Section 4.4.3. Both the methodologies presented in Sections 4.4.2 and 4.4.3 are based on Dally and Seitz's theorem [78] (Corollary 3.1), thus requiring the absence of cyclic dependencies between channels. The resulting routing algorithms require a large number of virtual channels. When the restriction on cyclic dependencies is relaxed, the number of resources needed to avoid deadlock is reduced considerably. In Section 4.4.4 we first present a routing algorithm that is not based on Theorem 3.1, showing the transition. It is followed by a design methodology based on Theorem 3.1. Although this design methodology presents a practical way to design efficient fully adaptive routing algorithms for a variety of topologies, some more adaptivity can be obtained by proposing specific routing algorithms for some topologies. This is done in Section 4.5.

4.4.1 Algorithms Based on Structured Buffer Pools

These routing algorithms were designed for SAF networks using central queues. Deadlocks are avoided by splitting buffers into several classes and restricting packets to move from one buffer to another in such a way that buffer class is never decremented. Gopal proposed several fully adaptive minimal routing algorithms based on buffer classes [134]. These algorithms are known as hop algorithms.

The simplest hop algorithm starts by injecting a packet into the buffer of class 0 at the current node. Every time a packet stored in a buffer of class i takes a hop to another node, it moves to a buffer of class $i+1$. This routing algorithm is known as the positive-hop algorithm. Deadlocks are avoided by using a buffer of a higher class every time a packet requests a new buffer. By doing so, cyclic dependencies between resources are prevented. A packet that has completed i hops will use a buffer of class i. Since the routing algorithm only supplies minimal paths, the maximum number of hops taken by a packet is limited by the diameter of the network. If the network diameter is denoted by D, a minimum of $D+1$ buffers per node are required to avoid deadlock. The main advantage of the positive-hop algorithm is that it is valid for any topology. However, the number of buffers required for fully adaptive deadlock-free routing is very high and this number depends on network size.

The minimum number of buffers per node can be reduced by allowing packets to move between buffers of the same class. In this case, classes must be defined such that packets moving between buffers of the same class cannot form cycles. In the negative-hop routing algorithm, the network is partitioned into several subsets in such a way that no subset contains two adjacent nodes. If S is the number of subsets, then subsets are labeled $0, 1, \ldots, S-1$, and nodes in subset i are labeled i. Hops from a node with a higher label to a node with a lower label are negative. Otherwise, hops are nonnegative. When a packet is injected, it is stored into the buffer of class 0 at the current node. Every time a packet stored in a buffer of class i takes a negative hop, it moves to a buffer of class $i+1$. If a packet stored in a buffer of class i takes a nonnegative hop, then it requests a buffer of the same class. Thus, a packet that has completed i negative hops will use a buffer of class i.

There is not any cyclic dependency between buffers. Effectively, a cycle starting at node A must return to node A and contains at least another node B. If B has a lower label than A, some hop between A and B (possibly through intermediate nodes) is negative, and the buffer class is increased. If B has a higher label than A, some hop between B and A (possibly through intermediate nodes) is negative, and the buffer class is increased. As a consequence, packets cannot wait for buffers cyclically, thus avoiding deadlocks. If D is the network diameter and S is the number of subsets, then the maximum number of negative hops that can be taken by a packet is $H_N = \lceil D(S-1)/S \rceil$. The minimum number of buffers per node required to avoid deadlock is $H_N + 1$. Figure 4.14 shows a partition scheme for k-ary 2-cubes with even k. Black and white circles correspond to nodes of subsets 0 and 1, respectively.

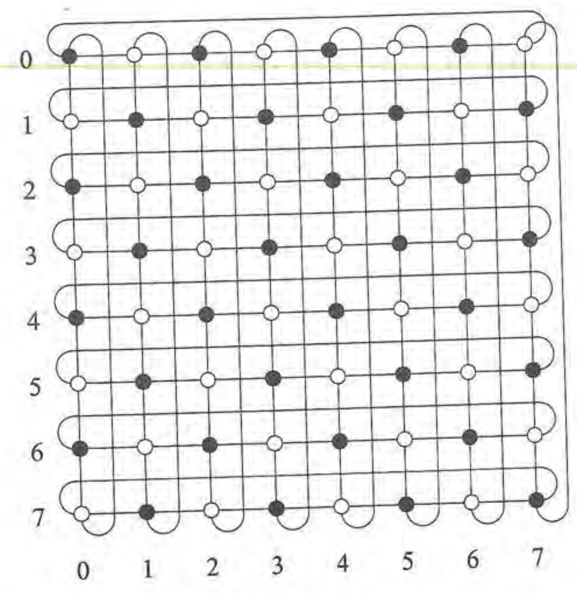


Figure 4.14. Partition scheme for k-ary 2-cubes with even k.

Although the negative-hop routing algorithm requires approximately half the buffers required by the positive-hop algorithm in the best case, this number is still high. It can be improved by partitioning the network into subsets and numbering the partitions, such that there are no cycles in any partition. This partitioning scheme does not require that adjacent nodes belong to different subsets. In this case, a negative hop is a hop that takes a packet from a node in a higher-numbered partition to a node in a lower-numbered partition. The resulting routing algorithm still requires a relatively large number of buffers, and the number of buffers depends on network size. In general, hop routing algorithms require many buffers. However, these algorithms are fully adaptive and can be used in any topology. As a consequence, these routing algorithms are suitable for SAF switching when buffers are allocated in main memory.

4.4.2 Algorithms Derived from SAF Algorithms

Boppana and Chalasani proposed a methodology for the design of fully adaptive minimal routing algorithms for networks using wormhole switching [36]. This methodology starts from a hop algorithm, replacing central buffers by virtual channels. The basic idea consists of splitting each physical channel into as many virtual channels as there were central buffers in the original hop algorithm, and assigning virtual channels in the same way that central buffers were assigned.

More precisely, if the SAF algorithm requires m classes of buffers, the corresponding algorithm for wormhole switching requires m virtual channels per physical channel. Let the virtual channels corresponding to physical channel c_i be denoted as $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$. If a packet occupies a buffer b_k of class k in the SAF algorithm and can use channels $c_i, c_j, \ldots$ to move to the next node, the corresponding wormhole algorithm will be able to use virtual channels $c_{i,k}, c_{j,k}, \ldots$. This situation is depicted in Figure 4.15 where a single physical channel c_i has been drawn.

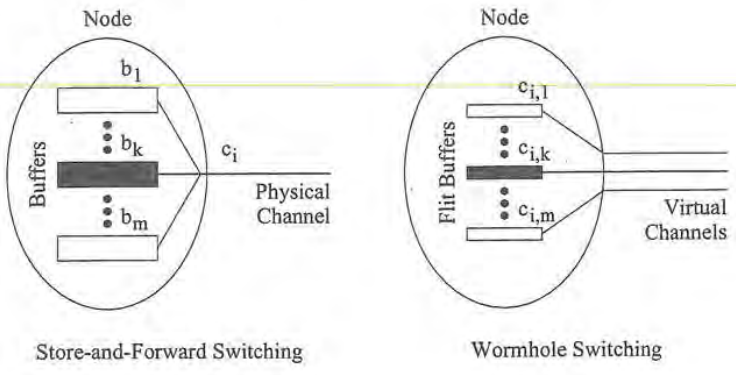


Figure 4.15. Derivation of wormhole routing algorithms from SAF algorithms.

This scheme produces an unbalanced use of virtual channels because all the packets start using virtual channel 0 of some physical channel. However, very few packets take the maximum number of hops. This scheme can be improved by giving each packet a number of bonus cards equal to the maximum number of hops minus the number of hops it is going to take [36]. At each node, the packet has some flexibility in the selection of virtual channels. The range of virtual channels that can be selected for each physical channel is equal to the number of bonus cards available plus one. Thus, when no bonus cards are available, a single virtual channel per physical channel can be selected. If the packet uses virtual channel j after using virtual channel i, it consumes $j - i - 1$ bonus cards.

Using this methodology, the hop algorithms presented in Section 4.4.1 can be redefined for wormhole switching. The routing algorithms resulting from the application of this methodology have the same advantages and disadvantages than the original hop algorithms for SAF networks: They provide fully adaptive minimal routing for any topology at the expense of a high number of virtual channels. Additionally, the number of virtual channels depends on network size, thus limiting scalability.

4.4.3 Virtual Networks

A useful concept to design routing algorithms consists of splitting the network into several virtual networks. A *virtual network* is a subset of channels that are used to route packets toward a particular set of destinations. The channel sets corresponding to different virtual networks are disjoint. Depending on the destination, each packet is injected into a particular virtual network, where it is routed until it arrives at its destination. In some proposals, packets traveling in a given virtual network have some freedom to move to another virtual network. Virtual networks can be implemented by using disjoint sets of virtual channels for each virtual network, and mapping those channels over the same set of physical channels. Of course it is also possible to implement virtual networks by using separate sets of physical channels.

The first design methodologies for fully adaptive routing algorithms were based on the concept of virtual networks [167, 213]. This concept considerably eases the task of defining deadlock-free routing functions. Effectively, deadlocks are only possible if there exist cyclic dependencies between channels. Cycles are formed by sets of turns, as shown in Section 4.3.2. By restricting the set of

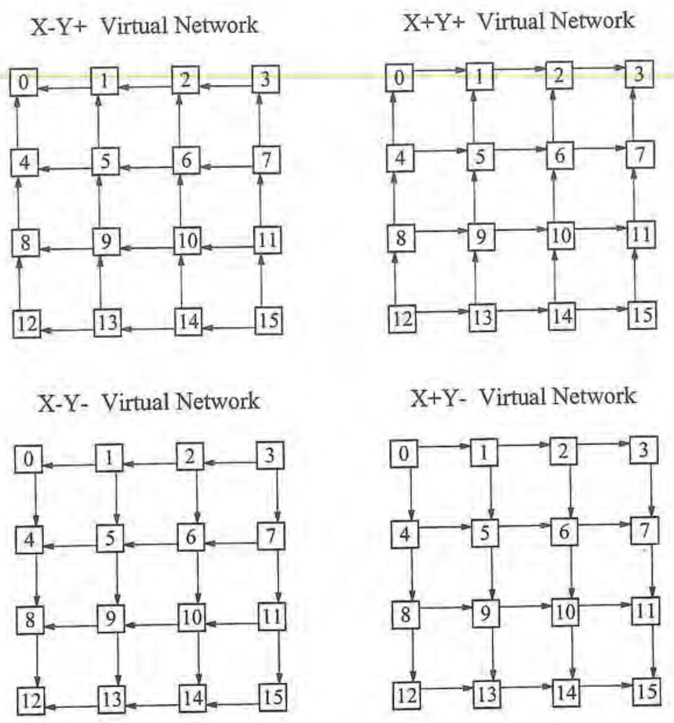


Figure 4.16. Virtual networks for a 2-D mesh.

destinations for each virtual network, it is possible to restrict the set of directions followed by packets. Thus, each virtual network can be defined in such a way that the corresponding routing function has no cyclic dependencies between channels. However, this routing function is not connected because it is only able to deliver packets to some destinations. By providing enough virtual networks so that all the destinations can be reached, the resulting routing function is connected and deadlock-free. Packet transfers between virtual networks are not allowed or restricted in such a way that deadlocks are avoided.

In this section, we present some fully adaptive routing algorithms based on virtual networks. Jesshope, Miller, and Yantchev proposed a simple way to avoid deadlock in n-dimensional meshes. It consists of splitting the network into several virtual networks in such a way that packets injected into a given virtual network can only move in one direction for each dimension [167]. Figure 4.16 shows the four virtual networks corresponding to a 2-D mesh. Packets injected into the $X+Y+$ virtual network can only move along the positive direction of dimensions X and Y. Packets injected into the $X+Y-$ virtual network can only move along the positive direction of dimension X and the negative direction of dimension Y, and so on. Packets are injected into a single virtual network, depending on their destination. Once a packet is being routed in a given virtual network, all the channels belonging to minimal paths can be used for routing. However, the packet cannot be transferred to another virtual network. It is obvious that there are no cyclic dependencies between channels, thus avoiding deadlock.

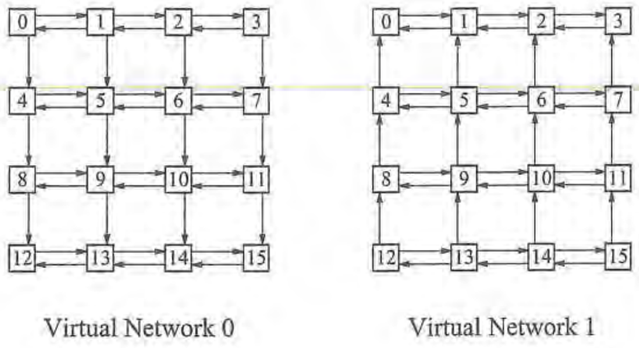


Figure 4.17. Reducing the number of virtual networks for a 2-D mesh.

This strategy can be easily extended for meshes with more than two dimensions. For example, a 3-D mesh requires eight virtual networks, corresponding to the following directions: $X-Y-Z-$, $X-Y-Z+$, $X-Y+Z-$, $X-Y+Z+$, $X+Y-Z-$, $X+Y-Z+$, $X+Y+Z-$, and $X+Y+Z+$. In general, a n-dimensional mesh requires 2^n virtual networks, each one consisting of n unidirectional virtual or physical channels per node (except some nodes in the border of the mesh). Thus, from a practical point of view, this routing algorithm can only be used for a very small number of dimensions (two or three at most). Also, a similar strategy can be applied to torus (k-ary n-cube) networks, as shown in [137] for a 2-D torus.

Linder and Harden showed that the number of virtual networks required for n-dimensional meshes can be reduced to a half [213]. Instead of having a unidirectional channel in each dimension, each virtual network has channels in both directions in the 0th dimension and only one direction in the remaining $n-1$ dimensions. This is shown in Figure 4.17 for a 2-D mesh. With this reduction, an n-dimensional mesh requires 2^{n-1} virtual networks. However, channels in the 0th dimension are bidirectional instead of unidirectional. Therefore, the number of virtual channels per node is equal to $n2^{n-1}$, in addition to an extra virtual channel in dimension 0 for a total of $(n+1)2^{n-1}$ per node. The number of virtual channels across each physical channel is 2^{n-1}, except for dimension 0 which has 2^n virtual channels due to the bidirectional nature. Again, this strategy can only be used for a very small number of dimensions. Note that the virtual networks for a 2-D mesh (see Figure 4.17) are equivalent to the virtual networks for a plane in planar-adaptive routing (see Figure 4.8). Figures 4.18 and 4.19 show the routing algorithm for 2-D meshes.

The reduction in the number of virtual networks does not introduce cyclic dependencies between channels, as far as packets are routed following only minimal paths. Effectively, as shown in Section 4.3.2, cycles are formed by sets of turns. The restriction to minimal paths eliminates all the 180-degree turns. At least two dimensions with channels in both directions are required to form a cycle. Thus, no virtual network has cyclic dependencies between channels.

Linder and Harden also applied the concept of virtual network to k-ary n-cube networks [213]. The basic idea is the same as for meshes: each virtual network has channels in both directions in the 0th dimension and only one direction in the remaining $n-1$ dimensions. However, the existence of wraparound channels makes more difficult to avoid cyclic dependencies. Thus, each virtual network is split into several levels, each level having its own set of virtual channels. Every time a packet crosses a wraparound channel, it moves to the next level. If only minimal paths are allowed, the wraparound channels in each dimension can only be crossed once by each packet. In the worst case,

```
Algorithm: Linder-Harden and PAR for 2-D Meshes (Virtual Network 0)
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ and $Yoffset < 0$ then
     $Channel :=$ Select($X-, Y-$);
  endif
  if $Xoffset < 0$ and $Yoffset = 0$ then
     $Channel := X-$;
  endif
  if $Xoffset > 0$ and $Yoffset < 0$ then
     $Channel :=$ Select($X+, Y-$);
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
     $Channel := X+$;
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
     $Channel := Y-$;
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
     $Channel := Internal$;
  endif
```

Figure 4.18. The Linder-Harden and planar-adaptive routing algorithms for 2-D meshes (virtual network 0). (PAR = Planar-adaptive routing.)

a packet will need to cross wraparound channels in all the dimensions. Thus, one level per dimension is required, in addition to the initial level, for a total of $n + 1$ levels. Figure 4.20 shows the virtual networks and their levels for a 2-D torus. For reference purposes, one of the levels is enclosed in a dashed box. This fully adaptive routing algorithm for k-ary n-cube networks requires 2^{n-1} virtual networks and $n + 1$ levels per virtual network, resulting in $(n+1)2^{n-1}$ virtual channels per physical channel across each dimension except dimension 0 which has $(n+1)2^n$ virtual channels. Thus, fully adaptive routing requires many resources if cyclic dependencies between channels are to be avoided. As we will see in Section 4.4.4, the number of resources required for fully adaptive routing can be drastically reduced by relying on Theorem 3.1 for deadlock avoidance.

Some recent proposals allow cyclic dependencies between channels, relying on some version of Theorem 3.1 to guarantee deadlock freedom [96]. In general, virtual networks are defined in such a way that the corresponding routing functions are deadlock-free. Packet transfers between virtual networks are restricted in such a way that deadlocks are avoided. By allowing cyclic dependencies between channels and cyclic transfers between virtual networks, guaranteeing deadlock freedom is much more difficult [96, 217]. However, virtual networks still have proven to be useful to define deadlock-free routing algorithms. See Exercise 3.3 for an example.

```
Algorithm: Linder-Harden and PAR for 2-D Meshes (Virtual Network 1)
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($X-, Y+$);
  endif
  if $Xoffset < 0$ and $Yoffset = 0$ then
    $Channel := X-$;
  endif
  if $Xoffset > 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($X+, Y+$);
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
    $Channel := X+$;
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    $Channel := Y+$;
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
    $Channel := Internal$;
  endif
```

Figure 4.19. The Linder-Harden and planar-adaptive routing algorithms for 2-D meshes (virtual network 1). (PAR = Planar-adaptive routing.)

4.4.4 Deterministic and Adaptive Subnetworks

In this section we first present a routing algorithm that is not based on Theorem 3.1, followed by a design methodology based on that theorem.

Dally and Aoki proposed two adaptive routing algorithms based on the concept of *dimension reversal* [74]. The most interesting one is the *dynamic* algorithm. This routing algorithm allows the existence of cyclic dependencies between channels, as far as packets do not wait for channels in a cyclic way. Before describing the dynamic algorithm, let us define the concept of dimension reversal. The dimension reversal (DR) number of a packet is the count of the number of times a packet has been routed from a channel in one dimension, p, to a channel in a lower dimension, $q < p$.

The dynamic algorithm divides the virtual channels of each physical channel into two nonempty classes: adaptive and deterministic. Packets injected into the network are first routed using adaptive channels. While in these channels, packets may be routed in any direction without a maximum limit on the number of dimension reversals a packet may make. Whenever a packet acquires a channel, it labels the channel with its current DR number. To avoid deadlock, a packet with a DR of p cannot wait on a channel labeled with a DR of q if $p \geq q$. A packet that reaches a node where all output

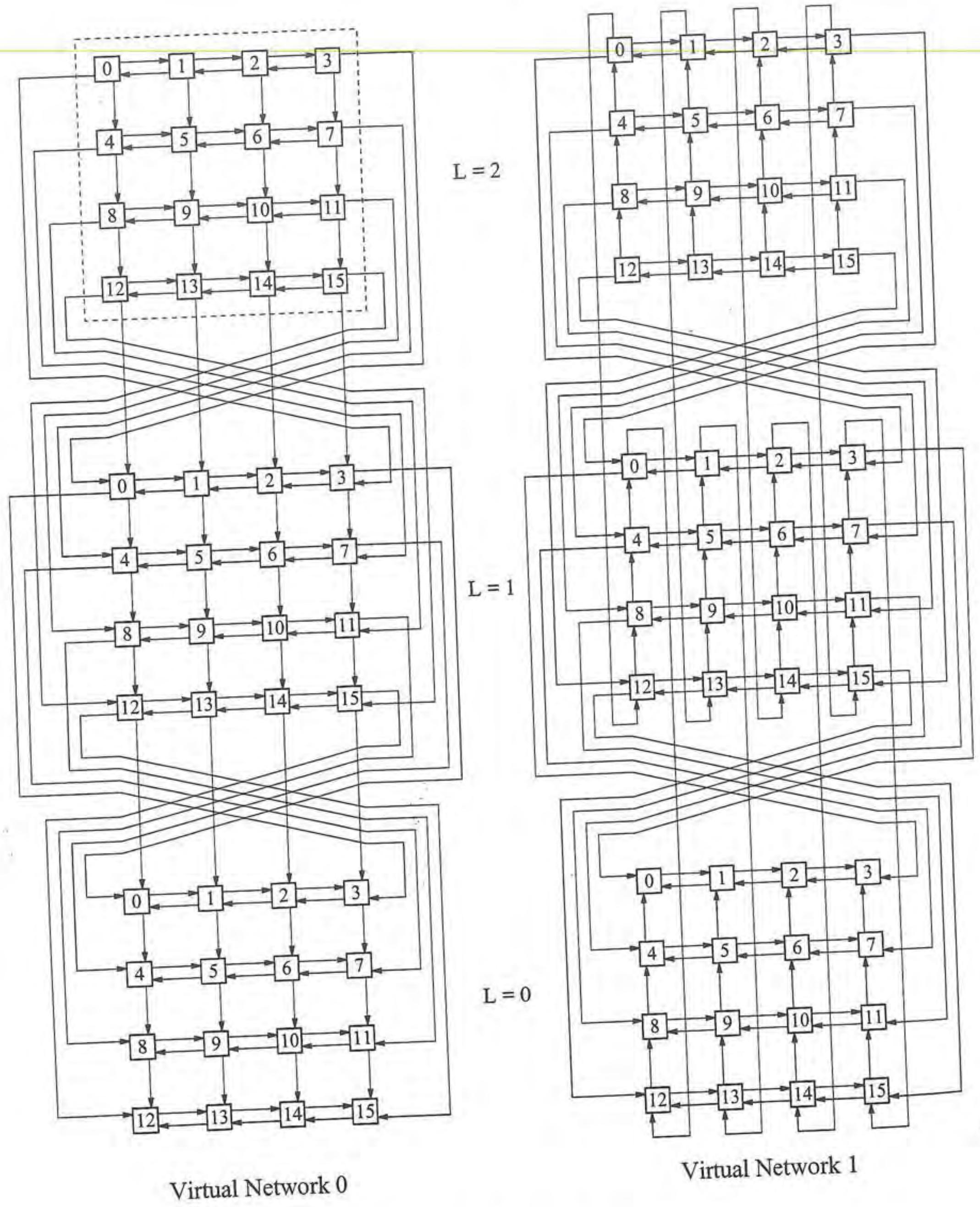


Figure 4.20. Virtual networks for a 2-D torus.

channels are occupied by packets with equal or lower DRs must switch to the deterministic class of virtual channels. Once on the deterministic channels, the packet must be routed in dimension order using only the deterministic channels and cannot reenter the adaptive channels.

The dynamic algorithm represents a first step toward relaxing the restrictions for deadlock avoidance. Instead of requiring the absence of cyclic dependencies between channels as the algorithms presented in Section 4.4.3, it allows those cyclic dependencies as far as packets do not wait for channels in a cyclic way. When a packet may produce a cyclic waiting, it is transferred to the deterministic class of virtual channels where it is routed in dimension order. The dynamic algorithm provides the maximum routing flexibility when packets are routed using adaptive channels. However, that flexibility is lost when packets are transferred to the deterministic channels.

More routing flexibility can be obtained by using Theorem 3.1. Fully adaptive routing algorithms described in Sections 4.4.1, 4.4.2, and 4.4.3 do not allow cyclic dependencies between resources to avoid deadlocks. However, those algorithms require a large set of buffer resources. Using Theorem 3.1, it is possible to avoid deadlocks even when cyclic dependencies between resources are allowed. The resulting routing algorithms require a smaller set of buffers or channels.

Defining routing algorithms and checking if they are deadlock-free is a tedious task. In order to simplify this task, Duato proposed some design methodologies [87, 92, 99]. In general, design methodologies do not supply optimal routing algorithms. However, they usually supply near-optimal routing algorithms with much less effort. In this section, we present a general methodology for the design of deadlock-free fully adaptive routing algorithms that combines the methodologies previously proposed by Duato. For VCT and SAF switching it automatically supplies deadlock-free routing algorithms. For wormhole switching, a verification step is required.

The design methodology presented in this section is based on the use of edge buffers. For SAF switching, a similar methodology can be defined for networks using central queues. This methodology describes a way to add channels to an existing network, also deriving the new routing function from the old one. Channels are added following a regular pattern, using the same number of virtual channels for all the physical channels. This is important from the implementation point of view because bandwidth sharing and propagation delay remains identical for all the output channels.

Methodology 4.1 *This methodology supplies fully adaptive minimal and nonminimal routing algorithms, starting from a deterministic or partially adaptive routing algorithm. When restricted to minimal paths, the resulting routing algorithms have been referred to as* Duato's protocol *(DP) [126]. The steps are the following:*

1. *Given an interconnection network I_1, select one of the existing routing functions for it. Let R_1 be this routing function. It must be deadlock-free and connected. It can be deterministic or adaptive. For wormhole switching, it is recommended that R_1 is minimal. Let C_1 be the set of channels at this point.*

2. *Split each physical channel into a set of additional virtual channels. Let C be the set of all the (virtual) channels in the network. Let C_{xy} be the set of output channels from node x belonging to a path (minimal or not) from node x to node y. Define the new routing function R as follows:*

$$R(x,y) = R_1(x,y) \cup (C_{xy} \cap (C - C_1)) \quad \forall x,y \in N \tag{4.1}$$

That is, the new routing function can use any of the new channels or, alternatively, the channels supplied by R_1. The selection function can be defined in any way. However, it is recommended to give a higher priority to the new channels belonging to minimal paths. For wormhole switching, it is recommended that R is restricted to use only minimal paths.

3. *For wormhole switching, verify that the extended channel dependency graph for R_1 is acyclic. If it is, the routing algorithm is valid. Otherwise, it must be discarded, returning to step 1.*

Step 1 establishes the starting point. We can use either a deterministic or adaptive routing function as the basic one. All the algorithms proposed in previous sections are candidates for selection. However, algorithms proposed in Sections 4.2 and 4.3.2 should be preferred because they require a small amount of resources.

Step 2 indicates how to add more (virtual) channels to the network and how to define a new fully adaptive routing function from the basic one. As defined, this methodology supplies nonminimal routing functions. It is possible to define minimal routing functions by restricting C_{xy} to contain only the output channels belonging to minimal paths from x to y. This methodology can also be applied by adding physical channels instead of virtual ones.

For wormhole switching, step 3 verifies whether the new routing function is deadlock-free or not. If the verification fails, the above proposed methodology may lead to an endless cycle. Thus, it does not supply a fully automatic way to design fully adaptive routing algorithms. Many minimal routing functions pass the verification step. However, very few nonminimal routing functions pass it. This is the reason why we recommend the use of minimal routing functions for wormhole switching.

For VCT and SAF switching, step 3 is not required. The methodology supplies deadlock-free routing functions. Effectively, according to the definition for R given in expression 4.1, $R_1(x,y) = R(x,y) \cap C_1 \quad \forall x,y \in N$. Thus, there exists a subset of channels $C_1 \subset C$ that defines a routing subfunction R_1 that is connected and deadlock-free. Taking into account Theorem 3.2, it is easy to see that R is deadlock-free. This is indicated in the next lemma.

Lemma 4.1 *For VCT and SAF switching, the routing functions supplied by Methodology 4.1 are deadlock-free.*

Methodology 4.1 is not defined for routing functions whose domain includes the current channel or queue containing the packet header. In fact, this methodology cannot be applied to such routing functions because it extends the range of the routing function but not its domain. If the domain is not extended, it is not possible to consider the newly added channels, and the packets stored in them cannot be routed.

Note that if the domain of R were extended by considering the newly added channels, the initial routing function R_1 would be modified. For example, consider the positive hop algorithm described in Section 4.4.1 and adapted to wormhole switching in Section 4.4.2. Routing decisions are taken based on the class of the channel occupied by the packet. Knowledge of the class is necessary to ensure deadlock freedom. If the packet uses a channel supplied by R_1, and subsequently a newly added channel, it is not possible for the receiving router to know the class of the channel previously occupied by the packet.

So, this methodology cannot be directly applied to routing functions that consider the current channel or queue in their domain. However, a closer examination will reveal that in practice this is not as restrictive as it may initially appear because most of those routing functions can be redefined so that they do not require the current channel or queue. For example, the class of the channel occupied by a packet in the positive hop algorithm can be made available by providing a field containing the class in the packet header. If this field is added, the input channel is no longer required in the domain of the routing function.

Examples 4.3 and 4.2 show the application of Methodology 4.1 to a 2-D mesh using SAF switching, and a binary n-cube using wormhole switching, respectively. Exercise 4.4 shows the application of this methodology to k-ary n-cubes as well as some optimizations.

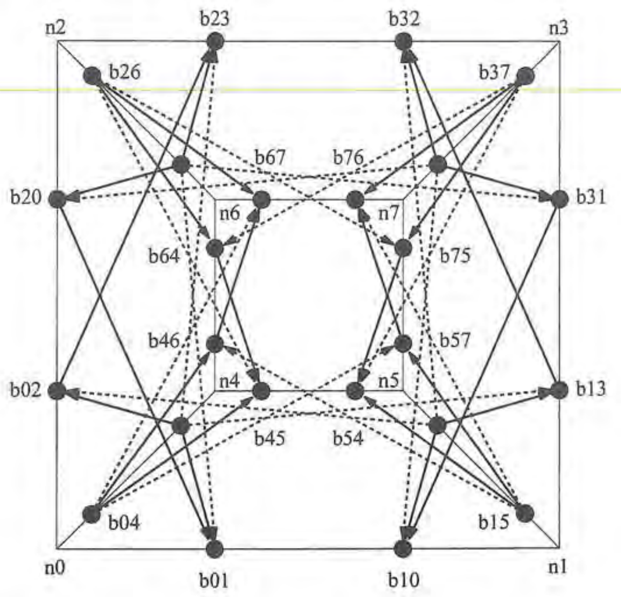


Figure 4.21. Extended channel dependency graph for R_1.

Example 4.2

Consider a binary n-cube using wormhole switching. For the step 1 we can select the e-cube routing algorithm. It forwards packets crossing the channels in order of decreasing dimensions. This routing function is connected and deadlock-free. For the step 2, consider that each physical channel c_i has been split into k virtual channels, namely, $a_{i,1}, a_{i,2}, \ldots, a_{i,k-1}, b_i$. Let C_1 be the set of b channels. The algorithm obtained applying the step 2 can be stated as follows: Route over any useful dimension using any of the a channels. If all of them are busy, route over the highest useful dimension using the corresponding b channel. A useful dimension is one that forwards a packet nearer to its destination.

Figure 4.21 shows the extended channel dependency graph for R_1 on a 3-cube. Black circles represent the unidirectional channels belonging to C_1 and are labeled as bij, where i and j are the source and destination nodes, respectively. As a reference, channels are also represented by thin lines, horizontal and vertical ones corresponding to dimensions 0 and 1, respectively. Also, the nodes of the 3-cube have been labeled as nk, where k is the node number. Thick lines represent channel dependencies, dashed arrows corresponding to indirect dependencies. It can be seen that the graph is acyclic. Then, R is deadlock-free.

Example 4.3

Consider a 2-D mesh using SAF switching. For the step 1 we select dimension-order routing (XY routing). Let R_1 denote this routing function. R_1 is connected and deadlock-free. Split each physical channel c_i into two virtual channels, namely, a_i and b_i. Let C_1 be the set of b_i

```
Algorithm: Duato's Fully Adaptive Algorithm for 2-D Meshes (Duato's Protocol)
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($Xa-, Ya-, Xb-$);
  endif
  if $Xoffset < 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($Xa-, Ya+, Xb-$);
  endif
  if $Xoffset < 0$ and $Yoffset = 0$ then
    $Channel :=$ Select($Xa-, Xb-$);
  endif
  if $Xoffset > 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($Xa+, Ya-, Xb+$);
  endif
  if $Xoffset > 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($Xa+, Ya+, Xb+$);
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
    $Channel :=$ Select($Xa+, Xb+$);
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($Ya-, Yb-$);
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($Ya+, Yb+$);
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
    $Channel := Internal$;
  endif
```

Figure 4.22. Duato's fully adaptive minimal routing algorithm for 2-D meshes (Duato's protocol).

channels. Thus, $C - C_1$ is the set of a_i channels. According to expression 4.1, the routing function R supplies all the outgoing a channels from the current node, also supplying one b channel according to dimension-order routing (XY routing). Taking into account Lemma 4.1, R is deadlock-free. When restricted to minimal paths, R is also deadlock-free for wormhole switching, as shown in Example 3.8. The resulting routing algorithm is shown in Figure 4.22. In this figure, $Xa+$ and $Xb+$ denote the a and b channels, respectively, in the positive direction of the X dimension. A similar notation is used for the other direction and dimension.

4.5 Maximally Adaptive Routing Algorithms

In this section, we present some routing algorithms based on deadlock avoidance that either maximize adaptivity for a given set of resources (channels or buffers) or minimize the use of resources for fully adaptive routing. We present below some routing algorithms based on deadlock recovery that accomplish both.

Some authors have proposed metrics to measure adaptivity [130]. However, such metrics are not related to performance. In fact, although maximally adaptive routing algorithms are optimal or near optimal in the use of resources for deadlock avoidance-based routing, they do not necessarily achieve the highest performance. The reason is that the use of resources may be unbalanced. For instance, different physical channels may have a different number of virtual channels or some virtual channels may be more heavily used than other ones. This may produce an uneven traffic distribution, as shown in [343]. This is not the case for true fully adaptive routing based on deadlock recovery. This algorithm maximizes adaptivity for a given set of resources while balancing the use of resources. As a consequence, this algorithm usually achieves the highest performance.

4.5.1 Algorithms with Maximum Adaptivity

The fully adaptive routing algorithm proposed by Linder and Harden requires two and one virtual channels per physical channel for the first and second dimension, respectively, when applied to 2-D meshes (see Section 4.4.3 and Figure 4.17). If dimensions are exchanged the resulting routing algorithm requires one and two virtual channels per physical channel for the X and Y dimension, respectively. This algorithm is called *double-y*. The double-y routing algorithm uses one set of Y channels, namely $Y1$, for packets traveling $X-$, and the second set of Y channels, namely $Y2$, for packets traveling $X+$.

Based on the turn model, Glass and Ni analyzed the double-y routing algorithm, eliminating the unnecessary restrictions [131]. The resulting algorithm is called *maximally adaptive double-y (mad-y)*. It improves adaptivity with respect to the double-y algorithm. Basically, mad-y allows packets using $Y1$ channels to turn to the $X+$ direction and packets using $X-$ channels to turn and use $Y2$ channels. Figures 4.23 and 4.24 show the turns allowed by the double-y and mad-y algorithms, respectively.

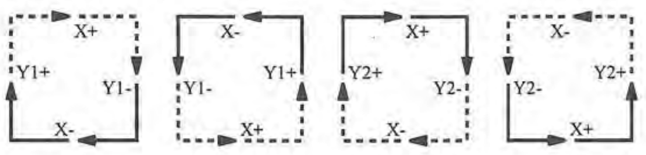


Figure 4.23. Turns allowed (solid lines) by the double-y algorithm.

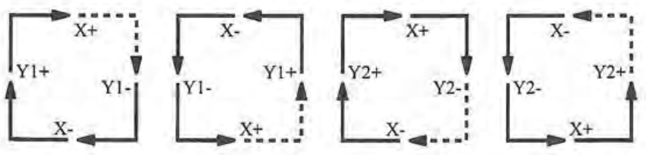


Figure 4.24. Turns allowed (solid lines) by the mad-y algorithm.

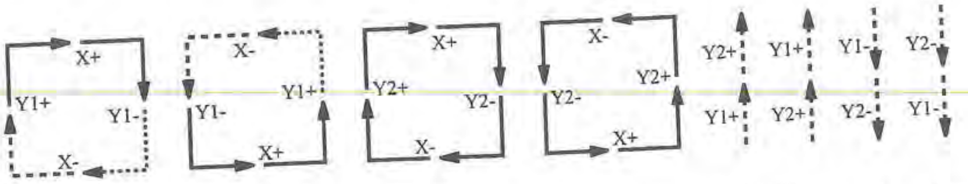


Figure 4.25. Turns allowed (solid lines) by the opt-y algorithm. Dotted lines are prohibited turns. Dashed lines are restricted turns.

The mad-y algorithm has the maximum adaptivity that can be obtained without introducing cyclic dependencies between channels. However, as shown in Theorem 3.1, cycles do not necessarily produce deadlock. Thus, the mad-y algorithm can be improved. It was done by Schwiebert and Jayasimha, who proposed the *opt-y* algorithm [305, 307]. This algorithm is deadlock-free and optimal with respect to the number of routing restrictions on the virtual channels for deadlock avoidance-based routing. Basically, the opt-y algorithm allows all the turns between X and $Y2$ channels as well as turns between $X+$ and $Y1$ channels. Turns from $Y1$ to $X-$ channels are prohibited. Turns from $X-$ to $Y1$ channels as well as 0-degree turns between $Y1$ and $Y2$ channels are restricted. These turns are only allowed when the packet has completed its movement along $X-$ channels (the X-offset is zero or positive). Figure 4.25 shows the turns allowed by the opt-y algorithm.

Defining a routing algorithm by describing the allowed and prohibited turns makes difficult to understand how routing decisions are taken at a given node. The opt-y algorithm is described in Figure 4.26 using pseudocode.

The opt-y algorithm can be generalized to n-dimensional meshes by using the following steps [307]:

- Assign a channel to both directions of each dimension.
- Number the dimensions in some order and add a second virtual channel to both directions of all dimensions except the first dimension.
- Allow packets to route along the second virtual channel at any time.
- For each dimension except the last, select one of the two directions as the chosen direction of that dimension. Prohibit a packet from routing on the first virtual channel of any direction until it has completed routing in the chosen direction of all lower dimensions.
- Allow a packet to make a 0-degree turn between the two virtual channels of a direction only after the packet has completed routing in the chosen direction of all lower dimensions.

Basically, the generalized opt-y algorithm allows fully adaptive minimal routing in one set of virtual channels. If packets have completed their movement along the chosen direction in all the dimensions then fully adaptive routing is also allowed on the second set of virtual channels. Otherwise the second set of virtual channels only allows partially adaptive routing across the dimensions for which packets have completed their movement along the chosen direction in all the lower dimensions.

4.5.2 Algorithms with Minimum Buffer Requirements

Cypher and Gravano [66] proposed two fully adaptive routing algorithms for torus networks using SAF switching. These algorithms have been proven to be optimal with respect to buffer space [64] for deadlock avoidance-based routing. Thus, there is no deadlock avoidance-based fully adaptive

```
Algorithm: Opt-y Fully Adaptive Algorithm for 2-D Meshes
Inputs: Coordinates of current node $(Xcurrent, Ycurrent)$
        and destination node $(Xdest, Ydest)$
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ and $Yoffset < 0$ then
    $Channel :=$ Select$(X-, Y2-)$;
  endif
  if $Xoffset < 0$ and $Yoffset > 0$ then
    $Channel :=$ Select$(X-, Y2+)$;
  endif
  if $Xoffset < 0$ and $Yoffset = 0$ then
    $Channel := X-$;
  endif
  if $Xoffset > 0$ and $Yoffset < 0$ then
    $Channel :=$ Select$(X+, Y2-, Y1-)$;
  endif
  if $Xoffset > 0$ and $Yoffset > 0$ then
    $Channel :=$ Select$(X+, Y2+, Y1+)$;
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
    $Channel := X+$;
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
    $Channel :=$ Select$(Y2-, Y1-)$;
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    $Channel :=$ Select$(Y2+, Y1+)$;
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
    $Channel := Internal$;
  endif
```

Figure 4.26. The opt-y fully adaptive routing algorithm for 2-D meshes.

routing algorithm for tori that requires less buffer space. Obviously, these routing algorithms have cyclic dependencies between queues or channels. The first algorithm (Algorithm 1) only requires three central buffers or queues to avoid deadlock, regardless of the number of dimensions of the torus. The second algorithm (Algorithm 2) uses edge buffers, requiring only two buffers per input channel to avoid deadlock. These routing algorithms are also valid for VCT switching but not for wormhole switching.

The routing algorithms are based on four node orderings. The first ordering, which is called the *right-increasing* ordering, is simply a standard row-major ordering of the nodes. The second

Table 4.1. The right-increasing ordering for an 8×8 torus.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Table 4.2. The left-increasing ordering for an 8×8 torus.

63	62	61	60	59	58	57	56
55	54	53	52	51	50	49	48
47	46	45	44	43	42	41	40
39	38	37	36	35	34	33	32
31	30	29	28	27	26	25	24
23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0

Table 4.3. The inside-increasing ordering for an 8×8 torus.

0	1	2	3	7	6	5	4
8	9	10	11	15	14	13	12
16	17	18	19	23	22	21	20
24	25	26	27	31	30	29	28
56	57	58	59	63	62	61	60
48	49	50	51	55	54	53	52
40	41	42	43	47	46	45	44
32	33	34	35	39	38	37	36

Table 4.4. The outside-increasing ordering for an 8×8 torus.

63	62	61	60	56	57	58	59
55	54	53	52	48	49	50	51
47	46	45	44	40	41	42	43
39	38	37	36	32	33	34	35
7	6	5	4	0	1	2	3
15	14	13	12	8	9	10	11
23	22	21	20	16	17	18	19
31	30	29	28	24	25	26	27

ordering, which is called the *left-increasing* ordering, is the reverse of the right-increasing ordering. The third ordering, which is called the *inside-increasing* ordering, assigns the smallest values to nodes near the wraparound edges of the torus and the largest values to nodes near the center of the torus. The fourth ordering, which is called the *outside-increasing* ordering, is the reverse of the inside-increasing ordering. Tables 4.1 through 4.4 show the node orderings for an 8×8 torus. A transfer of a packet from a node a to an adjacent node b will be said to occur to the *right* (similarly, *left*, *inside*, or *outside*) if and only if node a is smaller than node b when they are numbered in right-increasing (similarly, left-increasing, inside-increasing, or outside-increasing) ordering.

Algorithm 1: Three queues per node are required, denoted A, B, and C. Additionally, each node has an injection queue and a delivery queue. Each packet moves from its injection queue to the A queue in the source node and it remains using A queues as long as it is possible for it to move to the right along at least one dimension following a minimal path. When a packet cannot move to the right following a minimal path, it moves to the B queue in its current node and it remains using B queues as long as it is possible for it to move to the left along at least one dimension following a

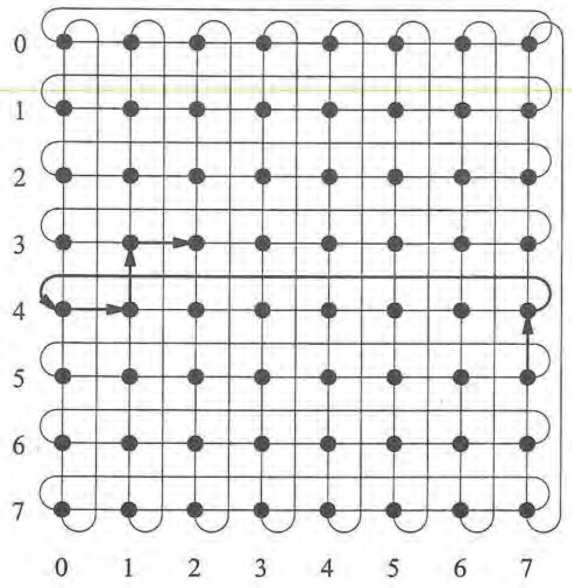


Figure 4.27. Routing example for Algorithms 1 and 2.

minimal path. When a packet cannot move to the left following a minimal path, it moves to the C queue in its current node and it remains moving to the inside using C queues until it arrives at its destination node. It then moves to the delivery queue in its destination node.

Note that a packet in an A queue may move to an A queue in any adjacent node. It may not actually move to the right, but this option must exist on at least one of the minimal paths to the destination. Similarly, a packet in a B queue may move to a B queue in any adjacent node. Also, note that a packet entering a C queue cannot move to an A or B queue. However, the routing algorithm is fully adaptive because a packet in a C queue only needs to move to the inside [66].

Example 4.4

Consider a packet p that is routed from node $(5,7)$ to node $(3,2)$ in an 8×8 torus using Algorithm 1 (Figure 4.27). The packet p will first be stored in the injection queue in node $(5,7)$. Then, it moves to the A queue in node $(5,7)$. At this point, p cannot move to the right. Thus, it moves to the B queue in node $(5,7)$. At this point, p is allowed to move to the left (moving to the B queue in node $(5,0)$ or in node $(4,7)$). Thus, all the B queues in neighboring nodes along minimal paths can be used. Assume that p moves to the B queue in node $(4,7)$, and then p moves to the B queue in node $(4,0)$. At this point, p is still allowed to move to the left (moving to the B queue in node $(3,0)$). Assume that p moves to the B queue in node $(4,1)$, and then to the B queue in node $(3,1)$. At this point, p can no longer move to the left. Thus, it moves to the C queue in node $(3,1)$. Then, p moves to the C queue in node $(3,2)$, reaching its destination node, and moving to the delivery queue in node $(3,2)$. Note that p moved to the inside when moving from node $(3,1)$ to node $(3,2)$.

Algorithm 2: Two edge queues per input channel are required, denoted A and C if the channel moves packets to the outside, and denoted B and D if the channel moves packets to the inside. Additionally, each node has an injection queue and a delivery queue. Assuming that a packet has not arrived at its destination, a packet in an injection, A or B queue moves to an A or B queue of an outgoing channel along a minimal path if it is possible for it to move to the inside along at least one dimension. Otherwise, it moves to the C queue of an outgoing channel along a minimal path. A packet in a C queue moves to a B or C queue of an outgoing channel along a minimal path if it is possible for it to move to the outside along at least one dimension. Otherwise, it moves to the D queue of an outgoing channel along a minimal path. A packet in a D queue can only move to the D queue of an outgoing channel along a minimal path. In any of the previous cases, when a packet arrives at its destination node, it moves to the delivery queue in that node.

Note that Algorithm 2 allows packets to move from C queues back to B queues and from B queues back to A queues. However, when a packet enters a D queue, it must remain using D queues until delivered. Once again, a packet entering a D queue can only move to the inside. However, the routing algorithm is fully adaptive because a packet in a D queue only needs to move to the inside [66].

Example 4.5

Consider a packet p that is routed from node $(5,7)$ to node $(3,2)$ in an 8×8 torus using Algorithm 2. Assume that p follows the same path as in Example 4.4 (see Figure 4.27). The packet p will first be stored in the injection queue in node $(5,7)$. At this point, p is allowed to move to the inside (moving to node $(4,7)$). Thus, p can select between the A queue in node $(5,0)$ and the B queue in node $(4,7)$. Assume that p moves to the B queue in node $(4,7)$. At this point, p can no longer move to the inside. Thus, it can move to the C queues in neighboring nodes along minimal paths. Assume that p moves to the C queue in node $(4,0)$. At this point, p is allowed to move to the outside (moving to the C queue in node $(3,0)$). Thus, p can select between the B queue in node $(4,1)$ and the C queue in node $(3,0)$. Assume that p moves to the B queue in node $(4,1)$. At this point, p is allowed to move to the inside (moving to the B queue in node $(4,2)$). Thus, p can select between the A queue in node $(3,1)$ and the B queue in node $(4,2)$. Assume that p moves to the A queue in node $(3,1)$. As p is still allowed to move to the inside, it moves to the B queue in node $(3,2)$, reaching its destination node, and moving to the delivery queue in node $(3,2)$.

4.5.3 True Fully Adaptive Routing Algorithms

All the routing algorithms described in previous sections use avoidance techniques to handle deadlocks, therefore restricting routing. Deadlock recovery techniques do not restrict routing to avoid deadlock. Hence, routing strategies based on deadlock recovery allow maximum routing adaptivity (even beyond that proposed in [305, 307]) as well as minimum resource requirements. In particular, progressive deadlock recovery techniques, like Disha (see Section 3.6), decouple deadlock handling resources from normal routing resources by dedicating minimum hardware to efficient deadlock recovery in order to make the common case (i.e., no deadlocks) fast. As proposed, sequential recovery from deadlocks requires only one central flit-sized buffer applicable to arbitrary network topologies [8, 9] and concurrent recovery requires at most two central buffers for any topology on which a Hamiltonian path or a spanning tree can be defined [10].

When routing is not restricted, no virtual channels are dedicated to avoid deadlocks. Instead, virtual channels are used for the sole purpose of improving channel utilization and adaptivity. Hence, *true fully adaptive* routing is permitted on all virtual channels within each physical channel, regardless of network topology. True fully adaptive routing can be minimal or nonminimal, depending on whether routing is restricted to minimal paths or not. Note that fully adaptive routing used in the context of avoidance-based algorithms connotes full adaptivity across all physical channels but only partial adaptivity across virtual channels within a given physical channel. On the other hand, true fully adaptive routing used in the context of recovery-based algorithms connotes full adaptivity across all physical channel dimensions as well as across all virtual channels within a given physical channel. Routing restrictions on virtual channels are therefore completely relaxed so that no ordering among these resources is enforced.

As an example, Figure 4.28 shows a true fully adaptive minimal routing algorithm for 2-D meshes. Each physical channel is assumed to be split into two virtual channels a and b. In this figure, $Xa+$ and $Xb+$ denote the a and b channels, respectively, in the positive direction of the X dimension. A similar notation is used for the other direction and dimension. As can be seen, no routing restrictions are enforced, except for paths to be minimal.

Figure 4.28 only shows the routing algorithm. It does not include deadlock handling. This issue was covered in Section 3.6. For the sake of completeness, Figure 4.29 shows a flow diagram of the true fully adaptive nonminimal routing algorithm implemented by Disha. The shaded box corresponds to the routing algorithm described in Figure 4.28 (extended to handle nonminimal routing). If after a number of tries a packet cannot access any virtual channel along any minimal path to its destination, it is allowed to access any misrouting channel except those resulting in 180-degree turns. If all minimal and misrouting channels remain busy for longer than the timeout for deadlock detection, the packet is eventually suspected of being deadlocked. Once this determination is made, its eligibility to progressively recover using the central *deadlock buffer* recovery path is checked. As only one of the packets involved in a deadlock needs to be eliminated from the dependency cycle to break the deadlock, a packet either uses the recovery path (is eligible to recover) or will eventually use one of the normal edge virtual channel buffers for routing (i.e., is not eligible to recover, but the deadlock is broken by some other packet that is eligible). Hence, Disha aims at optimizing routing performance in the absence of deadlocks and efficiently dealing with the rare cases when deadlock may be impending. If deadlocks are truly rare, substantial performance benefits can be gleaned (see Section 9.4.1).

4.6 Nonminimal Routing Algorithms

As indicated in the previous section, routing algorithms based on deadlock recovery can be designed to use nonminimal paths. Some methodologies proposed in previous sections can also be used for the design of avoidance-based nonminimal routing algorithms. This is the case for turn model, Dally and Aoki's algorithm and the methodology proposed in Section 4.4.4 for VCT switching. Also, PAR, hop algorithms, most algorithms based on virtual networks and the methodology proposed in Section 4.4.4 for wormhole switching can be extended so as to consider nonminimal routing.

For networks using wormhole switching, nonminimal routing algorithms usually degrade performance because packets consume more network resources. In particular, blocked packets occupy more channels on average, reducing the bandwidth available to the remaining packets. As a consequence, nonminimal routing algorithms are usually proposed for fault-tolerant routing because they are able to find alternative paths when all the minimal paths are faulty. These algorithms will be studied in Chapter 6.

```
Algorithm: True Fully Adaptive Minimal Algorithm for 2-D Meshes (Disha)
        (not including deadlock recovery)
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$)
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$;
  $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($Xa-, Ya-, Xb-, Yb-$);
  endif
  if $Xoffset < 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($Xa-, Ya+, Xb-, Yb+$);
  endif
  if $Xoffset < 0$ and $Yoffset = 0$ then
    $Channel :=$ Select($Xa-, Xb-$);
  endif
  if $Xoffset > 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($Xa+, Ya-, Xb+, Yb-$);
  endif
  if $Xoffset > 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($Xa+, Ya+, Xb+, Yb+$);
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
    $Channel :=$ Select($Xa+, Xb+$);
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
    $Channel :=$ Select($Ya-, Yb-$);
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    $Channel :=$ Select($Ya+, Yb+$);
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then
    $Channel := Internal$;
  endif
```

Figure 4.28. True fully adaptive minimal routing algorithm for 2-D meshes (Disha).

However, blocked packets are completely removed from the network when VCT switching is used. In this case, taking an unprofitable channel is likely to bring the packet to another set of profitable channels that will allow further progress to the destination. As indicated in Chapter 3, deadlock can be avoided in VCT switching by using deflection routing. This routing technique uses nonminimal paths to avoid deadlock when all the minimal paths are busy.

The Chaos router [188, 189] implements a fully adaptive nonminimal routing algorithm and uses deflection routing to avoid deadlock. This router uses VCT switching and splits messages into fixed-

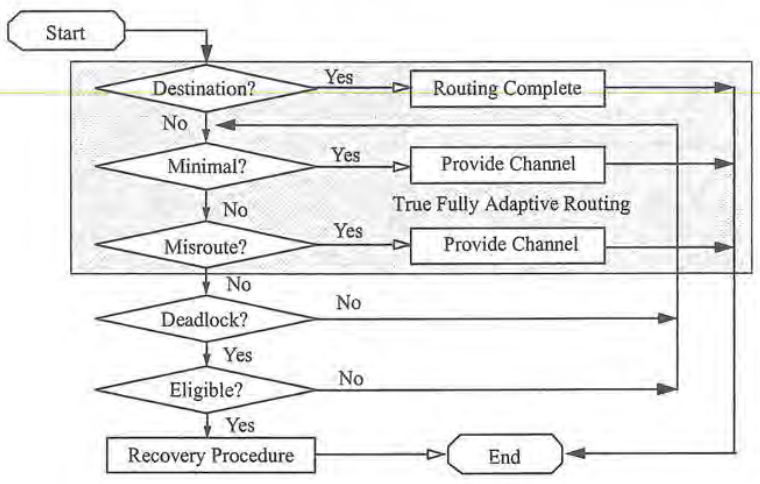


Figure 4.29. Flow diagram of the true fully adaptive nonminimal routing algorithm implemented by Disha.

length packets. It has a central packet queue implemented in hardware to remove packets from the network when they are blocked. The Chaos routing algorithm routes packets following a minimal path whenever possible. If all the minimal paths for a packet are busy, routing is retried for a fixed period of time. When waiting time exceeds a threshold, the packet is stored in the packet queue. To prevent starvation, packets in the queue have the highest priority in accessing a free output channel belonging to a minimal path when it becomes available. Queue overflow is prevented by derouting (misrouting) packets. When a queue slot is requested and the queue is full, a queued packet is randomly selected and forwarded along a nonminimal path.

Derouting packets requires guaranteeing that a nonminimal path is always available. This is equivalent to guaranteeing that some output channel is free. As indicated in Chapter 3, this can be ensured because the number of input channels (including injection channels) is equal to the number of output channels (including delivery channels). However, the Chaos router implements a more elaborate protocol to select the output channel for derouting. The protocol and formal proofs of deadlock freedom can be found in [189].

4.7 Backtracking Protocols

Backtracking protocols work on the premise that it is better to be searching for alternative paths than to be waiting for a channel to become available. This premise is specially true when the channel is faulty because it will not be available until repaired. The application of backtracking protocols to fault-tolerant routing will be studied in Chapter 6. In this section, we only consider the use of backtracking protocols to improve performance.

From a performance point of view, backtracking protocols are suited to circuit switching or some variant of it. Backtracking protocols search the network for a path in a depth-first manner. Potential paths from source to destination are searched by routing a header flit or probe through the network. The header acquires (virtual) channels as it moves toward its destination. When the header cannot continue onward, it backtracks over the last acquired channel, releases it, and continues its search from the previous node. As seen in Chapter 3, deadlock is prevented in circuit switching by reserving all the required resources before starting packet transmission. During the reservation phase, the header backtracks as needed instead of blocking when a channel is not available. Also, livelock is not a problem because backtracking protocols can use history information to avoid searching the same path repeatedly.

Although backtracking protocols can also be implemented with SAF switching, the performance overhead can be substantial compared to the use of deadlock avoidance techniques. From the behavioral point of view, the actual switching technique is unimportant and will not be referenced unless necessary in the remainder of this discussion.

Most backtracking protocols use the routing header to store history information [52, 62]. This significantly increases the size of the header over progressive protocols and, consequently, increases the time required to route the probe through the network. This is particularly a problem for misrouting backtracking protocols, since the number of links traversed during path setup can be very high. To overcome this problem, the history information can be distributed throughout the nodes of the network, reducing the header to a size comparable to that of e-cube [126, 127]. At each node in the network, each input link has a history bit vector h with as many bits as the node has channels. This history vector is associated with the circuit probe that came in on that channel. As each candidate outgoing channel is searched, the corresponding bit is set to "remember" that the channel has been searched. Each node also has a history bit vector h, for the node itself, since the node (not a channel) may be the source of the probe.

In this section, we briefly describe several backtracking protocols. Exhaustive profitable backtracking (EPB) performs a straightforward depth-first search of the network using only profitable links [140]. It is guaranteed to find a minimal path if one exists. This protocol is completely adaptive, profitable, and backtracking. Although the EPB protocol does not repeatedly search the same paths, it can visit a specific node several times (see Example 4.6). This can lead to unnecessary backtracking and longer setup times. The k-family routing paradigm is a family of partially adaptive, profitable, backtracking protocols proposed for binary hypercubes that use a heuristic to help minimize redundancy in the search for a path [62].

k-Family protocols are two-phased, using a heuristic search in the first phase and an exhaustive search in the second. Each protocol is distinguished by a parameter k that determines when the heuristic is used an when exhaustive backtracking is used. When the probe is at a distance from the destination greater than k, the heuristic is used; when the distance is less than or equal to k, an exhaustive profitable search is used. If $k = 1$, the protocol is a strictly heuristic search. As k grows to the distance between the source and destination, the search becomes more and more exhaustive. A k protocol makes use of a history mask contained in the circuit probe. At each level in the search tree, the cumulative history mask of the ancestor nodes determines which links might be explored. The history mask records all the dimensions explored at each link and all dimensions explored at all ancestor nodes. The heuristic limits exploration to dimensions not marked in the history mask. In this manner, the search tree in Example 4.6 is pruned of links that are likely to lead into areas of the network that have been searched already.

The multilink paradigm is a family of profitable backtracking protocols that only make a partial reservation of the path [296]. Instead of sending a probe to setup a path from source to destination, the multilink protocol uses wormhole switching in the absence of contention. When the requested output channel is busy, the router switches to multilink mode. In this mode, it sends a probe to search the network for a path of a given maximum size (multilink). The maximum number of channels that can be reserved at once is a fixed parameter for each protocol (multilink size). Once the path has been established, the router returns to normal mode and data flits advance. If

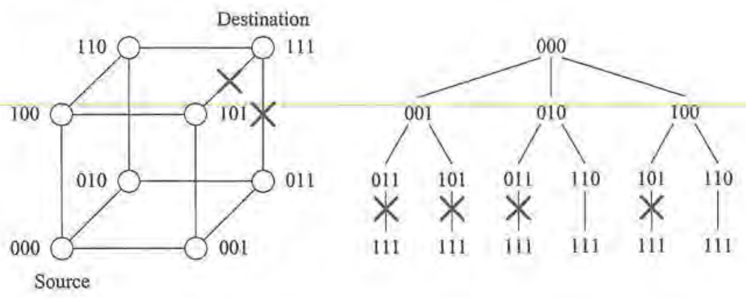


Figure 4.30. Search tree for a profitable backtracking protocol.

another busy channel is found, the router switches to multilink mode again, sending another probe. By limiting the number of channels that can be reserved at once, the search tree is pruned and the setup phase is much faster. However, the resulting routing algorithm is only partially adaptive. Also, deadlock freedom must be ensured by establishing an ordering between channels and multilinks. Additionally, once the data flits advance up to a given node, the probe can only backtrack up to that node. Although this protocol was initially proposed as profitable, it can be easily modified to use misrouting.

The exhaustive misrouting backtracking (EMB) protocol does a depth-first search of the network using both profitable and unprofitable links [126]. It uses the "best first" heuristic, taking profitable links over unprofitable ones. Although, on the average, the established circuits are longer than with a purely profitable protocol, EMB's probability of finding a path is greater. The drawback of this algorithm is that it cannot detect that a packet is undeliverable until it searches every path it can reach in the network. In the worst case, a single probe can tie up large amounts of network resources searching in vain for a nonexistent path. The protocol is just "too optimistic." Also, reserved paths may be too long, unnecessarily wasting channel bandwidth.

Misrouting freedom can be limited by using a partially adaptive two-phase protocol analogous to the k-family protocol where the probe is free to misroute only if it is within a certain distance u of its destination [126]. The first phase of two-phase backtracking (TPB-u) performs an exhaustive profitable search when the probe is at a distance greater than u from the destination. When TPB-u is within u of the destination, it enters the second phase, where it performs an exhaustive misrouting search. A TPB-u probe can switch phases several times during a single route.

The misrouting backtracking protocol with m misroutes (MB-m) is very much like the exhaustive misrouting backtracking protocol, except that it limits to m the maximum number of misroutes allowed at any time [127]. This protocol will be studied in Chapter 6.

In general, backtracking protocols do not improve performance over progressive ones when packets are short because the overhead to set up a path is very high. For very long packets or messages, backtracking protocols may perform better than progressive ones, specially when the number of alternative paths offered by the network is high.

Example 4.6

To illustrate backtracking protocols, we present the network shown in Figure 4.30. We will examine routing from node 000 to node 111 for EPB, k-family with $k = 1$, EMB, TPB-1, MB-1, and multilinks. Figure 4.30 shows the search tree for a profitable backtracking protocol routing a packet from node 000 to node 111 in a binary 3-cube. EPB will perform a depth-first search of the tree (left to right). Each path from the root node to a leaf node corresponds to a path from the source node (000) to the destination node (111) in the network. In Figure 4.30, links (011, 111) and (101, 111) are busy. In routing from 000 to 111, the EPB probe would visit the following nodes in sequence:

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 001 \rightarrow 101 \rightarrow 001 \rightarrow 000 \rightarrow 010 \rightarrow 011 \rightarrow 010 \rightarrow 110 \rightarrow 111$$

The final circuit the probe established would be

$$000 \rightarrow 010 \rightarrow 110 \rightarrow 111$$

Note that the probe had to backtrack all the way back to the source node and visited node 011 twice. The k-family probe would visit the following node sequence:

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 001 \rightarrow 101 \rightarrow 001 \rightarrow 000 \rightarrow 010 \rightarrow 110 \rightarrow 111$$

The resulting circuit would be the same as for EPB. Note that the k-family protocol pruned the second visit to 011 because the history mask passed to 010 from the source indicated that dimension 0 had already been searched. For EMB, TPB-1, and MB-1 the probe's search would proceed as follows:

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 010 \rightarrow 110 \rightarrow 111$$

The final circuit would be

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 010 \rightarrow 110 \rightarrow 111$$

In this case, since misrouting occurs at a node only one hop from the destination, TPB-1 behaves identically to EMB. If it were not possible to route at node 010, EMB would be free to misroute again, but TPB-1 would have to backtrack. MB-1 also behaves identically to EMB because only one misroute is required to search an alternative free path. Again, if it were not possible to route at node 010, MB-1 would have to backtrack. The final circuits established by the profitable protocols will always be the minimal length. The circuits established by the misrouting protocols can be much longer.

The multilinks protocol uses wormhole switching in the absence of contention. Thus, the header would reserve the following path:

$$000 \rightarrow 001 \rightarrow 011$$

As data flits immediately follow the header, it cannot backtrack at node 011. Thus, the header will have to wait for link (011, 111) to become free. The partially optimistic behavior of multilinks protocol prevents it from backtracking. However, if this protocol were modified to support misrouting, a probe would be sent from node 011 instead of waiting for a free channel. This probe would visit the following node sequence:

$$010 \rightarrow 110 \rightarrow 111$$

The resulting circuit would be the same as for MB-1.

4.8 Routing in MINs

In this section, we describe several issues concerning routing in MINs. For array processors, a central controller establishes the path from input to output. In cases where the number of inputs equals the number of outputs, each input synchronously transmits a message to one output, and each output receives a message from exactly one input. Computing the switch settings to realize such a permutation is a complex task. Furthermore, some permutations may not be realizable in one pass through the network. In this case, multiple passes of the data through the network may be required and the goal is to minimize the number of passes. The complexity of the off-line computation of the switch settings is proportional to the number of switches. This section and most multiprocessor applications consider only networks with the same number of inputs and outputs. Contention-free centralized routing will be addressed in Section 4.8.1. Also, see [262] for a brief survey.

On the other hand, in asynchronous multiprocessors, centralized control, and permutation routing are infeasible. So, a routing algorithm is required to establish the path across the stages of a MIN. The simplest solution consists of using source routing. In this case, the source node specifies the complete path. As this solution produces a considerable overhead, we will focus on distributed routing. Routing algorithms for MINs will be described in Section 4.8.2.

There are many ways to interconnect adjacent stages. See Sections 1.7.4 and 1.7.5 for a definition of some connection patterns and MIN topologies, respectively.

4.8.1 Blocking Condition in MINs

The goal of this section is to derive necessary and sufficient conditions for two circuits to block, i.e., require the same intermediate link. In an N processor system, there are exactly N links between every stage of $k \times k$ switches. The network consists of $n = \log_k N$ stages, where each stage is comprised of $\frac{N}{k}$ switches. The intermediate link patterns connect an output of switch stage i to an input of switch stage $i+1$. Blocking occurs when two packets must traverse the same output at a switch stage $i, i = 0, 1, \ldots n-1$. At any stage i, we can address all of the outputs of that stage from 0 to $N-1$. Let us refer to these as the intermediate outputs at stage i. If we take a black box view of the network, we can represent it as shown in Figure 4.31 for an Omega network with 2×2 switches. Each stage of switches and each stage of links can be represented as a function that permutes the inputs. An example is shown in the figure of a path from network input 6, to network output 1. The intermediate outputs on this path are marked as shown. They are 4, 0, and 1 at the output of switch stages 0, 1, and 2, respectively. Our initial goal is the following. Given an input/output pair, generate the addresses of all of the intermediate outputs on this path. Since these networks have a single path from input to output, these addresses will be unique. Two input/output paths conflict if they traverse a common intermediate link or share a common output at any intermediate stage.

In the following, we derive the blocking condition for the Omega network. Equivalent conditions can be derived for other networks in a similar manner. All input/output addresses in the following are represented in base k. For ease of discussion, we will assume a circuit-switched network. Consider establishing a circuit from $s_{n-1}s_{n-2}\ldots s_1s_0$ to $d_{n-1}d_{n-2}\ldots d_1d_0$. Consider the first link stage in Figure 4.31. This part of the network functionally establishes the following connection between its input and output.

$$s_{n-1}s_{n-2}\ldots s_1s_0 \rightarrow s_{n-2}s_{n-3}\ldots s_1s_0s_{n-1} \tag{4.2}$$

The right-hand side (RHS) of the above equation is the address of the output of the k-shuffle pattern and the address of the input to the first stage of switches. As mentioned in the previous

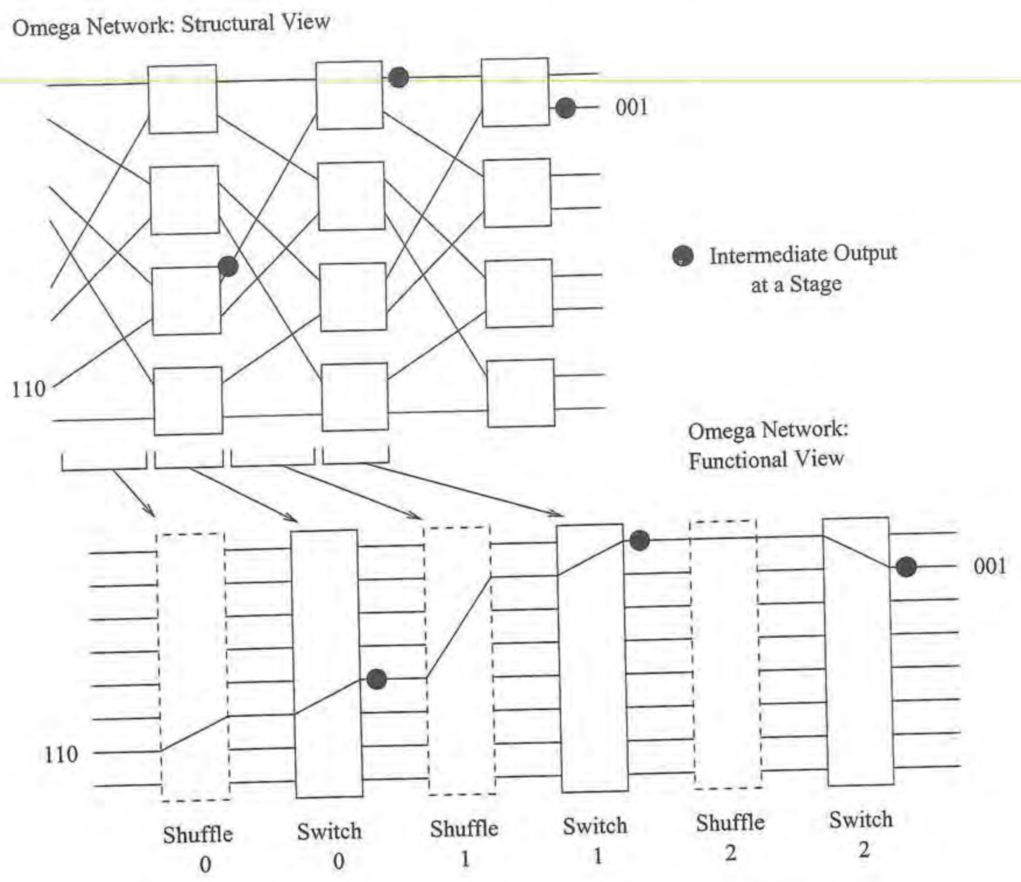

Omega Network: Structural View
001
Intermediate Output at a Stage
110
Omega Network: Functional View
001
110
Shuffle 0
Switch 0
Shuffle 1
Switch 1
Shuffle 2
Switch 2

Figure 4.31. Functional view of the structure of a multistage Omega network.

section, each switch is only able to change the least significant digit of the current address. After the shuffle permutation, this is s_{n-1}. So the output of the first stage of switches will be such that the least significant digit of the address be equal to d_{n-1}. Thus, the input/output connection established at the first stage of switches should be such that the input is connected to the output as follows.

$$s_{n-2}s_{n-3}\ldots s_1s_0s_{n-1} \rightarrow s_{n-2}s_{n-3}\ldots s_1s_0d_{n-1} \tag{4.3}$$

The RHS of the above expression is the address of the input to the second link stage. It is also the output of the first stage of switches and is therefore the first intermediate output. An example is shown in Figure 4.31 for a path from input 6 to output 1 which has a total of three intermediate outputs. Similarly, we can write the expression for the address of the intermediate output at stage i as

$$s_{n-i-2}s_{n-i-3}\ldots s_0d_{n-1}d_{n-2}\ldots d_{n-i-1} \tag{4.4}$$

This is assuming that stages are numbered from 0 to $n-1$. We can now write the blocking condition. For any two input/output pairs (S, D) and (R, T), the two paths can be set up in a conflict-free manner if and only if, $\forall i, 0 \leq i \leq n-1$

$$s_{n-i-2}s_{n-i-3}\dots s_0 d_{n-1}d_{n-2}\dots d_{n-i-1} \neq r_{n-i-2}r_{n-i-3}\dots r_0 t_{n-1}t_{n-2}\dots t_{n-i-1} \tag{4.5}$$

Testing for blocking between two input/output pairs is not quite as computationally demanding as it may first seem. Looking at the structure of the blocking condition we can see that if it is true and the circuits do block, then $n-i-1$ least significant digits of the source addresses are equal and $i+1$ most significant digits of the destination addresses are equal. Let us assume that we have a function $\phi(S, R)$ that returns the largest integer l such that the l least significant digits of S and R are equal. Similarly, let us assume that we have a function $\psi(D, T)$ that returns the largest integer m such that the m most significant digits of D and T are equal. Then two paths (S, D) and (R, T) can be established in a conflict-free manner if and only if

$$\phi(S, R) + \psi(D, T) < n \tag{4.6}$$

where $N = k^n$. As a practical matter, blocking can be computed by sequences of shift and compare operations, as follows:

$$\begin{array}{rcl} s_{n-1}s_{n-2}\dots & \boxed{s_2 s_1 s_0 d_{n-1}d_{n-2}\dots d_3} & d_2 d_1 d_0 \\ r_{n-1}r_{n-2}\dots & \boxed{r_2 r_1 r_0\ t_{n-1}t_{n-2}\dots t_3} & t_2 t_1 t_0 \end{array}$$

The addresses of the two input/output pairs are concatenated. Figuratively, a window of size n slides over both pairs and the contents of both windows are compared. If they are equal at any point, then there is a conflict at some stage. This will take $O(\log_k N)$ steps to perform. To determine if all paths can be set up in a conflict-free manner, we will have to perform $O(N^2)$ comparisons and each taking $O(\log_k N)$ steps resulting in an $O(N^2 \log_k N)$ algorithm. By comparison, the best known algorithm for centralized control to setup all of the switches in advance takes $O(N \log_k N)$ time. However, when using the above formulation of blocking it is often not necessary to perform the worst-case number of comparisons. Often the structure of the communication pattern can be exploited in off-line determination of whether patterns can be setup in a conflict-free manner.

4.8.2 Self-Routing Algorithms for MINs

A unique property of Delta networks is their self-routing property [272]. The self-routing property of these MINs allows the routing decision to be determined by the destination address, regardless of the source address. Self-routing is performed by using routing tags. For a $k \times k$ switch, there are k output ports. If the value of the corresponding routing tag is i $(0 \leq i \leq k-1)$, the corresponding packet will be forwarded via port i. For an n-stage MIN, the routing tag is $T = t_{n-1}\dots t_1 t_0$, where t_i controls the switch at stage G_i.

As indicated in Section 1.7.5, each switch is only able to change the least significant digit of the current address. Therefore, routing tags will take into account which digit is the least significant one at each stage, replacing it by the corresponding digit of the destination address. For a given destination $d_{n-1}d_{n-2}\dots d_0$, in a butterfly MIN the routing tag is formed by having $t_i = d_{i+1}$ for $0 \leq i \leq n-2$ and $t_{n-1} = d_0$. In a cube MIN, the routing tag is formed by having $t_i = d_{n-i-1}$ for $0 \leq i \leq n-1$. Finally, in an Omega network, the routing tag is formed by having $t_i = d_{n-i-1}$ for $0 \leq i \leq n-1$. Example 4.7 shows the paths selected by the tag-based routing algorithm in a 16-node butterfly MIN.

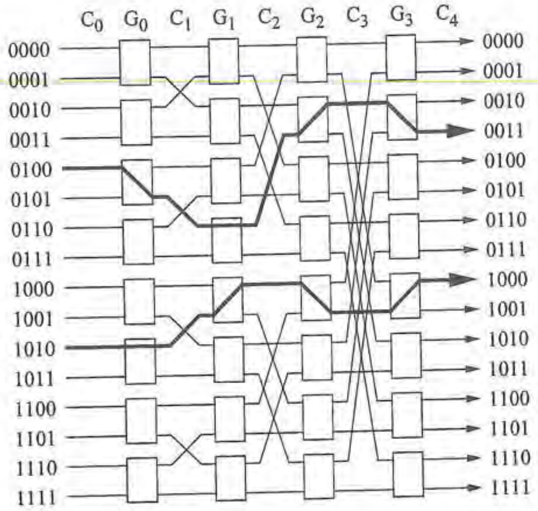


Figure 4.32. Paths selected by the tag-based routing algorithm in a 16-node butterfly MIN.

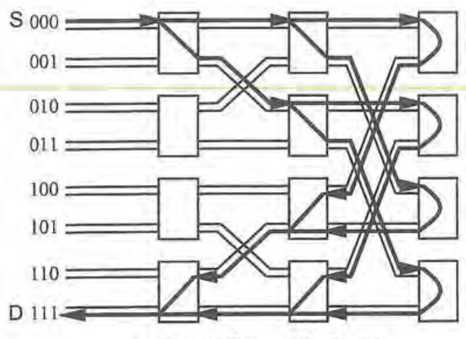
Example 4.7

Figure 4.32 shows a 16-node butterfly MIN using 2×2 switches, and the paths followed by packets from node 0100 to node 0011 and from node 1010 to node 1000. As indicated above, the routing tag for a given destination $d_{n-1}d_{n-2}\ldots d_0$ is formed by having $t_i = d_{i+1}$ for $0 \leq i \leq n-2$ and $t_{n-1} = d_0$. Thus, the routing tag for destination 0011 is 1001. This tag indicates that the packet must take the the upper switch output (port 0) at stages G_2 and G_1, and the lower switch output (port 1) at stages G_3 and G_0. The routing tag for destination 1000 is 0100. This tag indicates that the packet must take the upper switch output at stages G_3, G_1, and G_0, and the lower switch output at stage G_2.

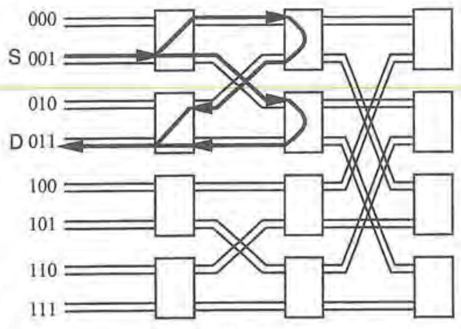
One of the nice features of the above traditional MINs (TMINs) is that there is a simple algorithm for finding a path of length $\log_k N$ between any input/output pair. However, if a link becomes congested or fails, the unique path property can easily disrupt the communication between some input and output pairs. The congestion of packets over some channels causes the known *hot spot* problem [274]. Many solutions have been proposed to resolve the hot spot problem. A popular approach is to provide multiple routing paths between any source and destination pair so as to reduce network congestion as well as to achieve fault tolerance. These methods usually require additional hardware, such as extra stages or additional channels.

The use of additional channels gives rise to dilated MINs. In a d-dilated MIN (DMIN), each switch is replaced by a d-dilated switch. In this switch, each port has d channels. By using replicated channels, DMINs offer substantial network throughput improvement [191]. The routing tag of a DMIN can be determined by the destination address as mentioned for TMINs. Within the network switches, packets destined for a particular output port are randomly distributed to one of the free channels of that port. If all channels are busy, the packet is blocked.

Another approach for the design of MINs consists of allowing for bidirectional communication. In this case, each port of the switch has dual channels. In a butterfly bidirectional MIN (BMIN)



(a) FirstDifference(S, D) = 2



(b) FirstDifference(S, D) = 1

Figure 4.33. Paths available in an eight-node butterfly BMIN.

built with $k \times k$ switches, source address S and destination address D are represented by k-ary numbers $s_{n-1} \ldots s_1 s_0$ and $d_{n-1} \ldots d_1 d_0$, respectively. The function *FirstDifference*(S, D) returns t, the position where the first (leftmost) different digit appears between $s_{n-1} \ldots s_1 s_0$ and $d_{n-1} \ldots d_1 d_0$.

A turnaround routing path between any source and destination pair is formally defined as follows. A turnaround path is a route from a source node to a destination node. The path must meet the following conditions:

- The path consists of some forward channel(s), some backward channel(s), and exactly one turnaround connection.
- The number of forward channels is equal to the number of backward channels.
- No forward and backward channels along the path are the channel pair of the same port.

Note that the last condition is to prevent redundant communication from occurring. To route a packet from source to destination, the packet is first sent forward to stage G_t. It does not matter which switch (at stage G_t) the packet reaches. Then, the packet is turned around and sent backward to destination. As it moves forward to stage G_t, a packet may have multiple choices as to which forward output channel to take. The decision can be resolved by randomly selecting from among those forward output channels which are not blocked by other packets. After the packet has attained a switch at stage G_t, it takes the unique path from that switch backward to its destination. The backward routing path can be determined by the tag-based routing algorithm for TMINs.

Note that the routing algorithms described above are distributed routing algorithms, in which each switch determines the output channel based on the address information carried in the packet. Example 4.8 shows the paths available in an eight-node butterfly BMIN.

Example 4.8

Figure 4.33 shows the paths available in an eight-node butterfly BMIN using 2×2 switches. Figure 4.33a shows the case when the source and destination nodes are 000 and 111, respectively. In this case, FirstDifference$(S, D) = 2$. Thus, packets turn at stage G_2 and there are four different shortest paths available. In Figure 4.33b, the source and destination nodes are 001 and 011, respectively. FirstDifference$(S, D) = 1$, and packets turn at stage G_1.

4.9 Routing in Switch-Based Networks with Irregular Topologies

Recently, switch-based interconnects like Autonet [304], Myrinet [30], and ServerNet [157] have been proposed to build NOWs for cost-effective parallel computing. Typically, these switches support networks with irregular topologies. Such irregularity provides the wiring flexibility required in LANs, also allowing the design of scalable systems with incremental expansion capability. The irregularity also makes the routing on such systems quite complicated.

Switch-based networks consist of a set of switches where each switch can have a set of ports. Each port consists of one input and one output link. A set of ports in each switch are either connected to processors or left open, where as the remaining ports are connected to ports of other switches to provide connectivity between the processors. Such connectivity is typically irregular and the only guarantee is that the network is connected. Typically, all links are bidirectional full-duplex and multiple links between two switches are allowed. Such a configuration allows a system with a given number of processors to be built using less number of switches than a direct network while allowing a reasonable number of external communication ports per processor. Figure 4.34a shows a typical NOW using switch-based interconnect with irregular topology. In this figure, it is assumed that switches have eight ports and each processor has a single port.

The switch may implement different switching techniques: wormhole switching, VCT, or ATM. However, wormhole switching is used in most cases. Several deadlock-free routing schemes have been proposed in the literature for irregular networks [30, 157, 284, 304]. Routing in irregular topologies can be performed by using source routing or distributed routing. In the former case, each processor has a routing table that indicates the sequence of ports to be used at intermediate switches to reach

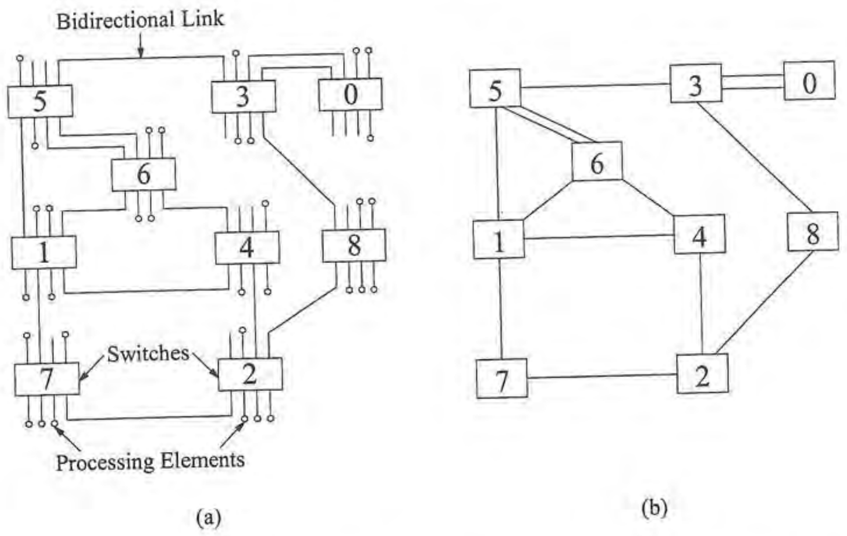


Figure 4.34. (a) An example system with switch-based interconnect and irregular topology. (b) The corresponding graph G.

the destination node. That information is stored in the packet header [30]. In the latter case, processors and switches require routing tables. The use of table lookup routing allows the use of the same switch fabric for different topologies. However, some network mapping algorithm must be executed in order to fill those tables before routing can be performed. The details of the mapping algorithm greatly depend on the underlying hardware support. Typically, switches support some broadcasting capability in the absence of routing tables.

In this section, we briefly describe the deadlock-free routing scheme used in DEC AN1 (Autonet) networks [304]. This routing scheme is valid for all the switching techniques. In addition to providing deadlock-freedom, it provides adaptive communication between some nodes in an irregular network. Also, we describe a fully adaptive routing algorithm for irregular topologies [318, 319] that considerably improves performance over the routing scheme proposed in [304].

Once a packet reaches a switch directly connected to its destination processor, it can be delivered as soon as the corresponding link becomes free. Therefore, we are going to focus on routing packets between switches. As indicated in Chapter 3, the interconnection network I between switches can be modeled by a multigraph $I = G(N, C)$, where N is the set of switches and C is the set of bidirectional links between the switches. It should be noted that bidirectional links were considered as two unidirectional channels in Chapter 3 but not in this section. Figure 4.34b shows the graph for the irregular network in Figure 4.34a.

The Autonet routing algorithm is distributed, and implemented using table-lookup. When a packet reaches a switch, the destination address stored in the packet header is concatenated with the incoming port number and the result is used to index the routing table at that switch. The table-lookup returns the outgoing port number that the packet should be routed through. When multiple routes exist from the source to the destination, the routing table entries return alternative outgoing ports. In case multiple outgoing ports are free, the routing scheme selects one port randomly.

In order to fill the routing tables, a breadth-first spanning tree (BFS) on the graph G is computed first using a distributed algorithm. This algorithm has the property that all nodes will eventually agree on a unique spanning tree. Routing is based on an assignment of direction to the operational links. In particular, the up end of each link is defined as: (1) the end whose switch is closer to the root in the spanning tree; (2) the end whose switch has the lower ID, if both ends are at switches at the same tree level. Links looped back to the same switch are omitted from the configuration. The result of this assignment is that each cycle in the network has at least one link in the up direction and one link in the down direction.

To eliminate deadlocks while still allowing all links to be used, this routing uses the following up/down rule: a legal route must traverse zero or more links in the up direction followed by zero or more links in the down direction. Thus, cyclic dependencies between channels are avoided because a packet may never traverse a link along the up direction after having traversed one in the down direction. Such routing not only allows deadlock-freedom but also adaptivity. The lookup tables can be constructed to support both minimal and nonminimal adaptive routing. However, in many cases, up/down routing is not able to supply any minimal path between some pairs of nodes, as shown in the following example.

Example 4.9

Figure 4.35 shows the link direction assignment for the example irregular network shown in Figure 4.34a. Switches are arranged in such a way that all the switches at the same tree level in the BFS spanning tree (rooted at switch 6) are at the same vertical position in the figure. The assignment of up direction to the links in this network is illustrated. The down direction is along the reverse direction of the link. Note that every cycle has at least one link in the

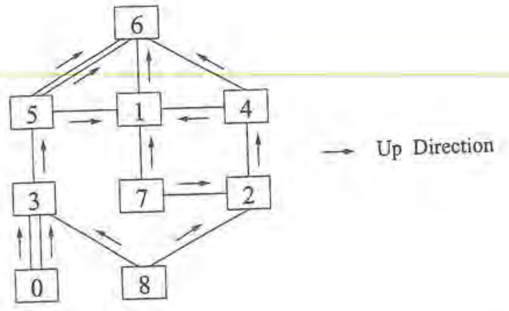


Figure 4.35. Link direction assignment for the irregular network shown in Figure 4.34a.

up direction and one link in the down direction. It can be observed that all the alternative minimal paths are allowed in some cases. For example, a packet transmitted from switch 5 to switch 4 can be either routed through switch 6 or switch 1. In some other cases, however, only some minimal paths are allowed. For example, a packet transmitted from switch 2 to switch 1 can be routed through switch 4 but it cannot be routed through switch 7. In this example, the packet can also be routed through switches 4 and 6, thus following a nonminimal path. It should be noted that any transmission between adjacent switches is always allowed to use the link(s) connecting them, regardless of the direction assigned to that link. However, when two switches are located two or more links away, it may happen that all the minimal paths are forbidden. This is the case for packets transmitted from switch 3 to switch 2. The shortest path (through switch 8) is not allowed. All the allowed paths (through switches 5, 1, 4, through switches 5, 6, 4, through switches 5, 1, 6, 4, and through switches 5, 6, 1, 4) are nonminimal.

The use of nonminimal paths consumes more channel bandwidth than strictly required to transmit packets. This drawback becomes more pronounced as network size increases. Nonminimal paths are imposed by the need to avoid deadlock. However, as indicated in Section 3.1.3, it is not necessary to remove all the cyclic dependencies between channels in order to avoid deadlock.

The design methodology presented in Section 4.4.4 cannot be directly applied to irregular networks implementing wormhole switching. The reason is that most routing functions defined for irregular networks only provide nonminimal paths between some pairs of nodes. As a consequence, the routing functions supplied by that methodology do not pass the verification step (step 3) in most cases.

A general methodology for the design of adaptive routing algorithms for networks with irregular topology was proposed in [319]. It is similar to the design methodology presented in Section 4.4.4 but it provides less routing flexibility. That methodology can be summarized as follows. Given an interconnection network and a deadlock-free routing function defined on it, it is possible to duplicate all the physical channels in the network, or to split them into two virtual channels. In both cases, the graph representation of the new network contains the original and the new channels. Then, the routing function is extended so that newly injected messages can use the new channels without any restriction as long as the original channels can only be used in the same way as in the original routing function. However, once a message reserves one of the original channels, it can no longer reserve any of the new channels again. This constraint is the only difference between this methodology and Methodology 4.1 presented in Section 4.4.4. Indeed, the resulting routing algorithms are very much like the dynamic dimension reversal routing algorithm described in Section 4.4.4, except that packets do not need to carry the dimension reversal number.

It is easy to see that the design methodology described above supplies deadlock-free routing algorithms. Effectively, consider a routing subfunction R_1 identical to the original routing function. The channels supplied by R_1 are used exactly in the same way in the original routing function and in the extended routing function. Hence there are no (direct or indirect) cross-dependencies between channels supplied by R_1. Also, packets using channels supplied by R_1 can no longer use other channels not supplied by R_1. Hence, there are no indirect dependencies between channels supplied by R_1. Therefore, the extended channel dependency graph for R_1 is identical to the channel dependency graph for the original routing function, which is acyclic. Thus, the extended routing function is deadlock-free.

According to the extended routing function defined above, new channels provide more routing flexibility than original channels. Besides they can be used to route messages through minimal paths. However, once a message reserves an original channel, it is routed through the original paths, which, in most cases, are nonminimal. Also, routing through original paths produces a loss of adaptivity.

Following this reasoning, the general methodology proposed in [319] can be refined by restricting the transition from new channels to original channels, since they provide less adaptivity and non-minimal paths in most cases. The extended routing function can be redefined in the following way. Newly injected messages can only leave the source switch using new channels belonging to minimal paths, and never using original channels. When a message arrives at a switch from another switch through a new channel, the routing function gives a higher priority to the new channels belonging to minimal paths. If all of them are busy, then the routing algorithm selects an original channel belonging to a minimal path (if any). To ensure that the new routing function is deadlock-free, if none of the original channels provides minimal routing, then the original channel that provides the shortest path will be used. Once a message reserves an original channel, it will be routed using this kind of channels according to the original routing function until it is delivered.

This enhanced design methodology was proposed in [318] and can be applied to any routing algorithm that avoids deadlock by prohibiting cyclic dependencies between channels. In particular, it can be applied to networks using up/down routing simply by splitting physical channels into two virtual channels (or duplicating each physical channel), and using up/down routing as the original routing function. By restricting the use of original channels as indicated above, most messages are allowed to follow minimal paths, and therefore a more efficient use of resources is made. Also, adaptivity is considerably increased with respect to the original up/down routing algorithm. As a result, latency decreases significantly, and the network is able to deliver a throughput several times higher than the one achieved by the original up/down routing algorithm [318](see Section 9.8).

4.10 Resource Allocation Policies

In this section we briefly describe several policies proposed in the literature for the allocation of network resources. Resource allocation policies can be grouped into two classes: those required to select a resource for a packet when several resources are available, and those required to arbitrate between several packets contending for the same resource. The first class of policies is usually required to select a channel or buffer among the options offered by an adaptive routing function. The function that performs this selection is usually referred to as *selection function.* The second class of policies is usually required to allocate resources like the routing control unit (the circuit that computes the routing algorithm), or the bandwidth of a physical channel when it is multiplexed among several virtual channels. Both classes of policies are studied in the next sections.

4.10.1 Selection Function

Without loss of generality, let us assume that when a packet is routed, the network resources requested by that packet are the output channels at the current node. As indicated in Section 4.1, adaptive routing algorithms can be decomposed into two functions: routing and selection. The routing function supplies a set of output channels based on the current node or buffer and the destination node. The selection function selects an output channel from the set of channels supplied by the routing function. This selection can be random. In this case, network state is not considered (oblivious routing). However, the selection is usually done taking into account the status of output channels at the current node. Obviously, the selection is performed in such a way that a free channel (if any) is supplied. However, when several output channels are available, some policy is required to select one of them. Policies may have different goals, like balancing the use of resources, reserving some bandwidth for high-priority packets, or even delaying the use of resources that are exclusively used for deadlock avoidance. Regardless of the main goal of the policy, the selection function should give preference to channels belonging to minimal paths when the routing function is nonminimal. Otherwise the selection function may produce livelock. In this section we present some selection functions proposed in the literature for several purposes.

In [74], three different selection functions were proposed for n-dimensional meshes using wormhole switching with the goal of maximizing performance: *minimum congestion*, *maximum flexibility*, and *straight lines*. In minimum congestion, a virtual channel is selected in the dimension and direction with the most available virtual channels. This selection function tries to balance the use of virtual channels in different physical channels. The motivation for this selection function is that packet transmission is pipelined. Hence, flit transmission rate is limited by the slowest stage in the pipeline. Balancing the use of virtual channels also balances the bandwidth allocated to different virtual channels. In maximum flexibility, a virtual channel is selected in the dimension with the greatest distance to travel to the destination. This selection function tries to maximize the number of routing choices as a packet approaches its destination. In meshes, this selection function has the side effect of concentrating traffic in the central part of the network bisection, therefore producing an uneven channel utilization and degrading performance. Finally, in straight lines, a virtual channel is selected in the dimension closest to the current dimension. So, the packet will continue traveling in the same dimension whenever possible. This selection function tries to route packets in dimension order unless the requested channels in the corresponding dimension are busy. In meshes, this selection function achieves a good distribution of traffic across the network bisection. These selection functions were evaluated in [74] for 2-D meshes, showing that minimum congestion achieves the lowest latency and highest throughput. Straight lines achieves similar performance. However, maximum flexibility achieves much worse results. These results may change for other topologies and routing functions, but in general minimum congestion is a good choice for the reason mentioned above.

For routing functions that allow cyclic dependencies between channels, the selection function should give preference to adaptive channels over channels used to escape from deadlocks [92]. By doing so, escape channels are only used when all the remaining channels are busy, therefore increasing the probability of escape channels being available when they are required to escape from deadlock. The selection among adaptive channels can be done by using any of the strategies described above. Note that if escape channels are not free when requested, it does not mean that the routing function is not deadlock-free. Escape channels are guaranteed to become free sooner or later. However, performance may degrade if packets take long to escape from deadlock. In order to reduce the utilization of escape channels and increase their availability to escape from deadlock, it is possible to delay the use of escape channels by using a timeout. In this case, the selection function can only select an escape channel if the packet header is waiting for longer than the timeout. The motivation for this kind of selection function is that there is a high probability of an adaptive channel becoming available before the timeout expires. This kind of selection function was referred to as *time-dependent selection function* [89, 95]. In particular, the behavior of progressive deadlock recovery mechanisms (see Section 3.6) can be modeled by using a time-dependent selection function.

The selection function also plays a major role when real-time communication is required. In this case, best-effort packets and guaranteed packets compete for network resources. If the number of priority classes for guaranteed packets is small, each physical channel may be split into as many virtual channels as priority classes. In this case, the selection function will select the appropriate virtual channel for each packet according to its priority class. When the number of priority classes is high, the set of virtual channels may be split into two separate virtual networks, assigning best-effort packets and guaranteed packets to different virtual networks. In this case, scheduling of guaranteed packets corresponding to different priority classes can be achieved by using packet switching [293]. In this switching technique, packets are completely buffered before being routed in intermediate nodes, therefore allowing the scheduling of packets with earliest deadline first. Wormhole switching is used for best-effort packets. Latency of guaranteed packets can be made even more predictable by using an appropriate policy for channel bandwidth allocation, as indicated in the next section.

4.10.2 Policies for Arbitration and Resource Allocation

There exist some situations where several packets contend for the use of resources, therefore requiring some arbitration and allocation policy. These situations are not related to the routing algorithm but are described here for completeness. The most interesting cases of conflict in the use of resources arise when several virtual channels belonging to the same physical channel are ready to transfer a flit, and when several packet headers arrived at a node and need to be routed. In the former case, a virtual channel allocation policy is required while the second case requires a routing control unit allocation policy.

Flit level flow control across a physical channel involves allocating channel bandwidth among virtual channels that have a flit ready to transmit and have space for this flit at the receiving end. Any arbitration algorithm can be used to allocate channel bandwidth including random, round-robin, or priority schemes. For random selection, an arbitrary virtual channel satisfying the above mentioned conditions is selected. For round-robin selection, virtual channels are arranged in a circular list. When a virtual channel transfers a flit, the next virtual channel in the list satisfying the above-mentioned conditions is selected for the next flit transmission. This policy is usually referred to as *demand-slotted round-robin*, and is the most frequently used allocation policy for virtual channels.

Priority schemes require some information to be carried in the packet header. This information should be stored in a status register associated with each virtual channel reserved by the packet. Deadline scheduling can be implemented by allocating channel bandwidth based on a packet's deadline or age (earliest deadline or oldest age first) [73]. Scheduling packets by age reduces the variance of packet latency. For real-time communication, the latency of guaranteed packets can be made more predictable by assigning a higher priority to virtual channels reserved by those packets. By doing so, best-effort packets will not affect the latency of guaranteed packets because flits belonging to best-effort packets will not be transmitted unless no flit belonging to a guaranteed packet is ready to be transmitted. This approach, however, may considerably increase the latency of best-effort packets. A possible solution consists of allowing up to a fixed number of best-effort flits to be transmitted every time a guaranteed packet is transmitted [293].

When traffic consists of messages of very different lengths, a few long messages may reserve all of the virtual channels of a given physical channel. If a short message requests that channel, it will have to wait for long. A simple solution consists of splitting long messages into fixed length packets. This approach reduces waiting time for short messages but introduces some overhead because each packet carries routing information and has to be routed. Channel bandwidth is usually allocated to virtual channels at the granularity of a flit. However, it can also be allocated at the granularity of a block of flits, or even a packet, thus reducing the overhead of channel multiplexing. The latter approach is followed in the T9000 transputer [228]. In this case, buffers must be large enough to store a whole packet.

A different approach to support messages of very different lengths has been proposed in the Segment Router [185, 186]. This router has different buffers (virtual channels) for short and long messages. Additionally, it provides a central queue for short messages. The Segment Router implements VCT switching for short messages and a form of buffered wormhole switching for long messages. Instead of allocating channel bandwidth at the granularity of a flit, the Segment Router allocates channel bandwidth to short messages for the transmission of the whole message. For long messages, however, channel bandwidth is allocated for the transmission of a segment. A *segment* is the longest fragment of a long message that can be stored in the corresponding buffer. Segments do not carry header information and are routed in order following the same path. If a physical channel is assigned to a long message, after the transmission of each segment it checks for the existence of waiting short messages. If such a message exists, it is transmitted over the physical channel followed by the next segment of the long message. In this channel allocation scheme a channel is only multiplexed between one long and (possibly) many short messages. It is never multiplexed between two long messages. However, no priority scheme is used for channel allocation. It should be noted that once the transmission of a short message or segment starts over a channel, it is guaranteed to complete because there is enough buffer space at the receiving end.

A router is usually equipped with a routing control unit that computes the routing algorithm when a new packet header is received, eventually assigning an output channel to that packet. In this case, the routing control unit is a unique resource, requiring arbitration when several packets contend for it. Alternatively, the circuit computing the routing function can be replicated at each input virtual channel, allowing the concurrent computation of the sets of candidate output channels when several packet headers need to be routed at the same time. This is the approach followed in the Reliable Router [75]. However, the selection function cannot be fully replicated because the execution of this function requires updating the channel status register. This register is a centralized resource. Otherwise, several packets may simultaneously reserve the same output channel. So when two or more packets compete for the same output channel, some arbitration is required.

The traditional scheduling policy for the allocation of the routing control unit (or for granting access to the channel status register) is round-robin among input channels. In this scheduling policy, input channel buffers form a logical circular list. After routing the header stored in a buffer, the pointer to the current buffer is advanced to the next input buffer where a packet header is ready to be routed. The routing function is computed for that header, supplying a set of candidate output channels. Then, one of these channels is selected, if available, and the packet is routed to that channel. This scheduling policy is referred to as *input driven* [117]. It is simple but when the network is heavily loaded a high percentage of routing operations may fail because all of the requested output channels are busy.

An alternative scheduling policy consists of selecting a packet for which there is a free output channel. In this strategy, output channels form a logical circular list. After routing a packet, the pointer to the current output channel is advanced to the next free output channel. The router tries to find a packet in an input buffer that needs to be routed to the current output channel. If packets are found, it selects one to be routed to the current output channel. If no packet is found, the pointer to the current output channel is advanced to the next free output channel. This scheduling policy is referred to as *output-driven* [117]. Output-driven strategies may be more complex than input-driven ones. However, performance is usually higher when the network is heavily loaded [117]. Output-driven scheduling can be implemented by replicating the circuit computing the routing function at each input channel, and adding a register to each output channel to store information about the set of input channels requesting that output channel. Although output-driven scheduling usually improves performance over input-driven scheduling, the difference between them is much smaller when channels are randomly selected [117]. Indeed, a random selection usually performs better than round-robin when input-driven scheduling is used. Finally, it should be noted that most selection functions described in the previous section cannot be implemented with output-driven scheduling.

4.11 Engineering Issues

From an engineering point of view, the "best" routing algorithm is either the one that maximizes performance or the one that maximizes the performance/cost ratio, depending on the design goals. A detailed quantitative analysis must consider the impact of the routing algorithm on node design. Even if we do not consider the cost, a more complex routing algorithm may increase channel utilization at the expense of a higher propagation delay and a lower clock frequency. A slower clock also implies a proportional reduction in channel bandwidth if channels are driven synchronously. Thus, router design must be considered to make performance evaluation more realistic. This is the reason why we postpone performance evaluation until a detailed study of the routing hardware is presented.

In this section, we present a brief qualitative performance study. It would be nice if we could conclude that some routing algorithm is the "best" one. However, even if we do not consider the cost, the optimal routing algorithm may vary depending on network traffic conditions. For wormhole switching, fully adaptive routing algorithms that require many virtual channels are not interesting in general because they drastically slow down clock frequency. This is the case for the algorithms presented in Sections 4.4.2 and 4.4.3.

For low-dimensional meshes and a uniform distribution of packet destinations, the performance of deterministic routing algorithms is similar to or even higher than that of partially or fully adaptive algorithms. This is because meshes are not symmetric and adaptive algorithms concentrate traffic in the central part of the network bisection. However, deterministic algorithms distribute traffic across the bisection more evenly. For symmetric networks, like tori and hypercubes, fully adaptive algorithms that require few resources to avoid deadlock (like the ones presented in Section 4.4.4) usually outperform deterministic routers, even for uniform traffic. For switch-based networks with irregular topology, adaptive routing algorithms that allow cyclic dependencies between channels (like the ones presented in Section 4.9) also increase performance over deterministic or partially adaptive routing. This increment is usually much higher than in symmetric networks. The reason is that deterministic or partially adaptive routing algorithms for irregular networks usually route many messages across nonminimal paths, thus offering more chances for improvement.

When traffic is not uniformly distributed, the behavior may differ completely. On one hand, when there are hot spots, partially and fully adaptive algorithms usually outperform deterministic algorithms considerably. This is the case for some synthetic loads like bit-reversal, perfect shuffle, and butterfly [101]. On the other hand, when traffic locality is very high, adaptive algorithms do not help and the lower clock frequency affects performance negatively [101].

Some optimizations like the ones presented in Sections 4.5.1 and 4.5.2 have a small impact on performance. They may actually reduce performance because the use of resources is less balanced than in routing algorithms like the ones designed with Methodology 4.1. On the other hand, the true fully adaptive routing algorithm presented in Section 4.5.3 provides full adaptivity and allows a better balanced use of resources, usually achieving the highest performance. However, this routing algorithm must be combined with deadlock detection and recovery techniques. Currently available detection techniques only work efficiently when all the packets are short and have a similar length.

Traffic pattern is not the only parameter that influences performance. Packet size affects performance considerably. If messages or packets are very short, as is the case in distributed shared-memory multiprocessors, adaptive minimal and even nonminimal routing algorithms usually increase performance but the additional complexity introduced by the arrival of packets out of order may not be worth the additional performance. Additionally, as indicated above, if traffic locality is high, adaptive algorithms are not interesting at all.

For long messages that are not split into small packets, nonminimal routing algorithms are not interesting because bandwidth is wasted every time a long message reserves a nonminimal path. Note that this is not the case for short packets because misrouting is an alternative to waiting for a free minimal path and channels are reserved for a short period of time. However, profitable backtracking algorithms may outperform progressive algorithms when messages are long because they look for alternative minimal paths instead of waiting for a free channel. The main drawback of backtracking algorithms is the overhead produced by the circuits required to keep track of the header history, thus avoiding to search a path twice. Also, note that backtracking algorithms cannot be easily combined with wormhole or VCT switching. They are usually used on networks using some variant of circuit switching. As data does not follow the header immediately, some additional performance degradation is produced.

In summary, the optimal routing algorithm depends on traffic conditions. For uniform or local traffic, deterministic routing algorithms are the best choice. For irregular traffic or hot spots, the adaptive routers usually outperform deterministic ones. However, as we will see in Chapter 9, the impact of the routing algorithm on performance is small when compared with other parameters like traffic characteristics.

4.12 Commented References

This chapter aims at describing the most interesting routing algorithms and design methodologies proposed up to now. References to the original work have been included along with the descriptions. Also, some proposals were presented in Chapter 3 while describing deadlock handling mechanisms. Although the number of routing algorithms proposed in the literature is very high, most of them have some points in common with other proposals. For example, several algorithms proposed in the literature [137, 330] are equivalent or very similar to those obtained by using Methodology 4.1. Other proposals tried to maximize adaptivity, very much like the algorithms proposed in Section 4.5.1. This is the case for the *mesh-route* algorithm proposed in [42]. Also, some proposals combine previously proposed methodologies in an interesting way. The *positive-first, negative-first* routing algorithm for 2-D meshes proposed in [343] combines two algorithms from the turn model in a balanced way, trying to distribute traffic more evenly.

A detailed description of every proposed routing algorithm is beyond the scope of this book. The interested reader can refer to [317] for a survey on multistage interconnection networks and to [262] for a brief survey on switch-based networks. Routing algorithms for direct interconnection networks are surveyed in [126, 241, 254]. These surveys mainly focus on distributed routing using wormhole switching. However, source routing and table-lookup routing are also covered in [254]. Additionally, backtracking protocols are covered in [126]. Nevertheless this chapter covers several proposals that have not been surveyed elsewhere.

EXERCISES

4.1 Indicate the information stored in the packet header for the paths shown in Figure 4.36 when the routing algorithm is implemented using street-sign routing.

Solution To implement the routing algorithm using street-sign routing, the packet header must encode the information indicated in Table 4.5 for the packet destined for node 10. Table 4.6 contains the corresponding information for the packet destined for node 8.

Table 4.5. Routing actions for the packet destined for node 10.

Node	Action
2	Turn right
10	Deliver

Table 4.6. Routing actions for the packet destined for node 8.

Node	Action
12	Turn right
8	Deliver

4.2 Indicate the destination interval for each channel in each node to implement XY routing using interval routing on a 4×4 mesh.

Solution It is possible to implement XY routing using interval routing. However, the implementation is somewhat tricky, and requires labeling the nodes in a different way. Figure 4.37 shows a different node labeling as well as the paths followed by two packets using XY routing. Table 4.7 shows the destination intervals for all the output channels in the network. The output channel has been indicated in the first column for all the nodes in each row. Empty boxes indicate that packets cannot use that channel (the channel does not exist). Note that intervals do not overlap. Also, the union of the intervals for all the output channels of each node is the whole set of nodes, excluding the current one.

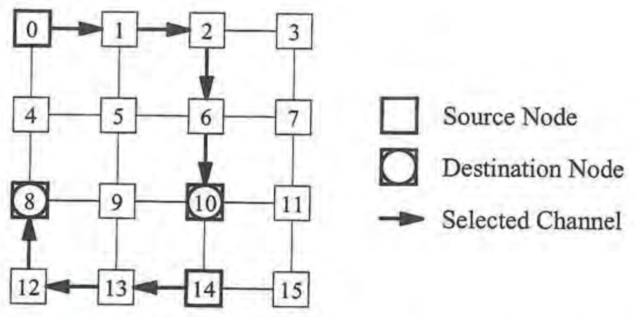


Figure 4.36. Node labeling and routing example for street-sign routing on a 2-D mesh.

Table 4.7. Routing tables for the output channels in a 4 × 4 mesh.

Channel	Node	Interval	Node	Interval	Node	Interval	Node	Interval
$X+$	0	4–15	4	8–15	8	12–15	12	
$X-$	0		4	0–3	8	0–7	12	0–11
$Y-$	0	1–3	4	5–7	8	9–11	12	13–15
$X+$	1	4–15	5	8–15	9	12–15	13	
$X-$	1		5	0–3	9	0–7	13	0–11
$Y+$	1	0–0	5	4–4	9	8–8	13	12–12
$Y-$	1	2–3	5	6–7	9	10–11	13	14–15
$X+$	2	4–15	6	8–15	10	12–15	14	
$X-$	2		6	0–3	10	0–7	14	0–11
$Y+$	2	0–1	6	4–5	10	8–9	14	12–13
$Y-$	2	3–3	6	7–7	10	11–11	14	15–15
$X+$	3	4–15	7	8–15	11	12–15	15	
$X-$	3		7	0–3	11	0–7	15	0–11
$Y+$	3	0–2	7	4–6	11	8–10	15	12–14

4.3 Draw the channel dependency graph for dimension-order routing on a binary 3-cube.

Solution Figure 4.38 shows the channel dependency graph for dimension-order routing on a 3-cube. This graph assumes that dimensions are crossed in decreasing order. Black circles represent the unidirectional channels and are labeled as cij, where i and j are the source and destination nodes, respectively. As a reference, channels are also represented by thin lines, horizontal and vertical ones corresponding to dimensions 0 and 1, respectively. Also, the nodes of the 3-cube have been labeled as nk, where k is the node number. Thick arrows represent channel dependencies. It can be seen that the graph is acyclic.

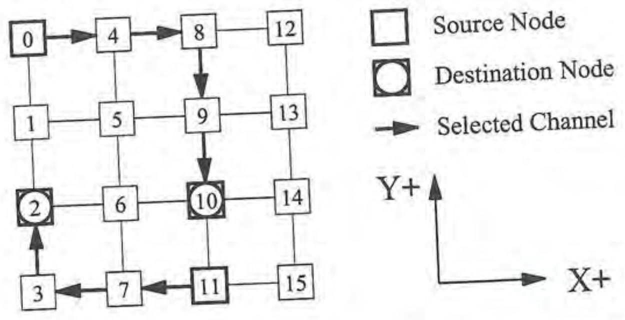


Figure 4.37. Node labeling and routing example for interval routing on a 2-D mesh.

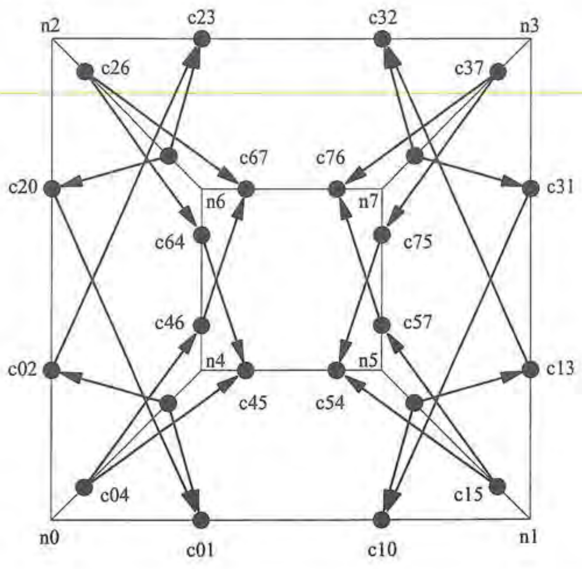


Figure 4.38. Channel dependency graph for dimension-order routing on a 3-cube.

4.4 Define a fully adaptive minimal routing algorithm for bidirectional k-ary n-cubes using Methodology 4.1.

Solution For step 1, it is possible to use the deterministic routing algorithm proposed in Example 4.1. Although this algorithm was proposed for unidirectional k-ary n-cubes, extending it for bidirectional channels is straightforward. Simply, a similar routing algorithm is used for both directions of each ring. As packets only follow minimal paths, there is not any dependency between channels in one direction and channels in the opposite direction. Thus, the routing algorithm is deadlock-free.

Step 2 adds a new virtual channel to each physical channel (one in each direction). The new virtual channels can be used for fully adaptive minimal routing. The resulting routing algorithm is deadlock-free because the extended channel dependency graph has no cycles (drawing this graph is left as an exercise).

As a consequence of extending the routing algorithm for bidirectional rings, some virtual channels are never used and can be removed. This optimization was proposed in [24, 137]. However, this optimization requires the use of different routers depending on the position in the network. A more efficient optimization was proposed in [101]. Instead of using the routing algorithm described in Example 4.1 for step 1, it uses the routing algorithm proposed in Example 3.3 for unidirectional rings, extended for bidirectional k-ary n-cubes by using dimension-order routing. Again, step 2 adds a new virtual channel to each physical channel (one in each direction) that can be used for fully adaptive minimal routing. In this case, no optimization is required after applying Methodology 4.1 because the resulting routing algorithm uses all the virtual channels.

4.5 Consider the minimal-path west-first routing algorithm. Using the turn model, include as many 180-degree turns as possible and describe the resulting algorithm using pseudocode.

Solution For each dimension, it is possible to include a single 180-degree turn. In dimension X, the only possible turn is from west to east. In dimension Y, there is a choice. Let us assume that the turn from north to south is allowed. Figure 4.39 shows the pseudocode for the resulting nonminimal west-first routing algorithm. It is assumed that the *Select*() function uses a static priority to select channels. For the list of parameters of this function, priority decreases from left to right. By doing so, a higher priority is assigned to the channels belonging to minimal paths. Also, note that the routing function requires the use of the current input channel buffer *InChannel* to select the output channel. This is necessary because a given packet may traverse a node twice.

PROBLEMS

4.1 The cube-connected cycle [283] can be defined as a binary hypercube of rings. It is an interconnection network based on the binary n-cube. Each node of a binary n-cube is replaced by an n-cycle, and the cube connection in the nth dimension is attached to the nth node in the cycle. Define a deterministic routing algorithm for the cube-connected cycle using the methodology proposed in Section 4.2.

4.2 The shuffle-exchange interconnection network [325] has two outgoing channels per node: a shuffle channel and an exchange channel. The shuffle channel from node n_i is connected to node n_j where the binary representation of j is the left rotation of the binary representation of i. The exchange channel from node n_i is connected to node n_k where the binary representations of i and k differ in the least significant bit. Define a deterministic routing algorithm for the shuffle-exchange network using the methodology proposed in Section 4.2.

4.3 Using the turn model, propose minimal-path partially adaptive routing algorithms for 3-D meshes that have as few routing restrictions as possible.

4.4 Apply Methodology 4.1 to design fully adaptive minimal routing algorithms for 2-D meshes, using the north-last, west-first and negative-first algorithms as the starting point (step 1). Verify whether the resulting algorithms are deadlock-free if wormhole switching is used.

4.5 Prove that the opt-y routing algorithm described in Section 4.5.1 is deadlock-free by drawing the extended channel dependency graph for a suitable routing subfunction and showing the absence of cycles.

Hint The routing subfunction can be defined by restricting R to use channels X and $Y1$.

4.6 Show that Algorithm 1 in Section 4.5.2 is deadlock-free by using Theorem 3.1.

Hint The routing subfunction R_1 can be defined as follows: When a packet is in an A queue and it is possible for it to move to the right along at least one dimension, R_1 supplies A queues that move the packet to the right. When a packet is in an A queue and it cannot move

```
Algorithm: Nonminimal West-First Algorithm for 2-D Meshes
Inputs: Coordinates of current node ($Xcurrent, Ycurrent$)
        and destination node ($Xdest, Ydest$),
        and current input channel buffer $InChannel$
Output: Selected output $Channel$
Procedure:
  $Xoffset := Xdest - Xcurrent$; $Yoffset := Ydest - Ycurrent$;
  if $Xoffset < 0$ then $Channel := X-$; endif
  if $Xoffset > 0$ and $Yoffset < 0$ then
    if $InChannel = X-$ then $Channel :=$ Select($Y-, X+, Y+, X-$);
    else if $InChannel = Y-$ then $Channel :=$ Select($Y-, X+$);
    else if $InChannel = Y+$ then $Channel :=$ Select($Y-, X+, Y+$);
    else $Channel :=$ Select($Y-, X+, Y+$);
    endif
  endif
  if $Xoffset > 0$ and $Yoffset > 0$ then
    if $InChannel = X-$ then $Channel :=$ Select($Y+, X+, Y-, X-$);
    else if $InChannel = Y-$ then $Channel :=$ Select($X+, Y-$);
    else if $InChannel = Y+$ then $Channel :=$ Select($Y+, X+, Y-$);
    else $Channel :=$ Select($Y+, X+, Y-$);
    endif
  endif
  if $Xoffset > 0$ and $Yoffset = 0$ then
    if $InChannel = X-$ then $Channel :=$ Select($X+, Y-, Y+, X-$);
    else if $InChannel = Y-$ then $Channel :=$ Select($X+, Y-$);
    else if $InChannel = Y+$ then $Channel :=$ Select($X+, Y+, Y-$);
    else $Channel :=$ Select($X+, Y-, Y+$);
    endif
  endif
  if $Xoffset = 0$ and $Yoffset < 0$ then
    if $InChannel = X-$ then $Channel :=$ Select($Y-, X-, Y+$);
    else if $InChannel = Y-$ then $Channel := Y-$;
    else if $InChannel = Y+$ then $Channel :=$ Select($Y-, Y+$);
    else $Channel :=$ Select($Y-, Y+$);
    endif
  endif
  if $Xoffset = 0$ and $Yoffset > 0$ then
    if $InChannel = X-$ then $Channel :=$ Select($Y+, X-$);
    else $Channel := Y+$;
    endif
  endif
  if $Xoffset = 0$ and $Yoffset = 0$ then $Channel := Internal$; endif
```

Figure 4.39. The nonminimal west-first routing algorithm for 2-D meshes.

to the right, R_1 supplies the B queue at current node. When a packet is in a B queue and it is possible for it to move to the left along at least one dimension, R_1 supplies B queues that move the packet to the left. When a packet is in a B queue and it cannot move to the left, R_1 supplies the C queue at current node. When a packet is in a C queue and it has not arrived at its destination, R_1 supplies C queues that move the packet to the inside. Obviously, when a packet has arrived at its destination, R_1 supplies no external channel.

4.7 Show that Algorithm 2 in Section 4.5.2 is deadlock-free by using Theorem 3.1.

Hint The routing subfunction R_1 can be defined as follows: When a packet is in an injection, A or B queue and it is possible for it to move to the inside along at least one dimension, R_1 supplies channels containing B queues. When a packet is in an injection, A or B queue and it can only move to the outside, R_1 supplies channels containing C queues. When a packet is in a C queue and it is possible for it to move to the outside along at least one dimension, R_1 supplies channels containing C queues. When a packet is in a C queue and it can only move to the inside, R_1 supplies channels containing D queues. When a packet is in a D queue and it has not arrived at its destination, R_1 supplies channels containing D queues. Obviously, when a packet is in any queue and it has arrived at its destination, R_1 supplies no external channel.

Chapter 7

Network Architectures

In many instances, particularly in fine-grained parallel machines, the performance of the applications are communication-limited rather than computation-limited. In this case, the design of the network architecture is crucial. By network architecture we refer to the topology of interconnection between the processing nodes, and the data and control path of the routers within the individual nodes. The design of the network architecture is fundamentally a process of making trade-offs between performance parameters such cost, latency, and throughput, and physical constraints such as area, wireability, and I/O limitations. Since physical constraints are continually evolving with advances in fabrication and packaging technology, so too are the appropriate choices of network topology and design alternatives for the router architecture. This chapter discusses basic issues facing the choice of network topologies and presents the design of router architectures that have been successfully employed within these topologies.

Chapter 1 describes many distinct topologies that have been proposed for use in multiprocessor architectures. The majority of these topologies are either configurations of, or isomorphic to, k-ary n-cube networks. These include the binary hypercube, tori, rings, and meshes. Indirect networks such as the indirect binary n-cube and Omega networks are also topologically closely related to these networks. Low dimensional k-ary n-cube networks possess desirable features with respect to their physical implementations, such as constant degree and ease of embedding in physical space. They also possess desirable properties with respect to message performance such as simple distributed routing algorithms. As a result, these networks have been used in the majority of commercial systems developed in the past decade. These include the iPSC series of machines from Intel, the Cray T3D and T3E, the Ametek 2010, MasPar MP-1 and MP-2, and Thinking Machines CM-1 and CM-2. These networks have formed the communications substrate of many commercial machines and are therefore the focus of this chapter. We observed that the network topology is primarily influenced by the packaging technology which places constraints on the wiring density, chip pin-out, and wire lengths. In the presence of these constraints, topological design issues must reconcile conflicting demands in choices of channel width, network dimensionality, and wire length. Thus, this chapter initially focuses on the optimization of the network topology.

While the architecture of the routers is certainly related to the topology of the interconnect, their design is primarily influenced by the switching technique that they are designed to support. Generic issues can be broadly classified into those dealing with internal data and control paths and those dealing with the design of the interrouter physical links. Specific design choices include internal switch design, buffer management, use of virtual channels, and physical channel flow control strategies. The remainder of the chapter presents descriptions of several modern router architectures as examples of specific design choices. Rather than present a comprehensive coverage of available architectures, we have tried to select illustrative examples from the range of currently available architectures.

7.1 Network Topology and Physical Constraints

For a fixed number of nodes, the choice of the number of dimensions of a k-ary n-cube represents a fundamental trade-off between network diameter and node degree. This choice of network dimension also places different demands on physical resources such as wiring area and number of chip I/Os. For a given implementation technology, practical constraints on these physical resources will determine architectural features such as channel widths, and as a result determine the *no-load message latency*: message latency in the absence of traffic. Similarly, these constraints will also determine the degree of congestion in the network for a given communication pattern, although accurate modeling of such dynamic behavior is more difficult. It thus becomes important to model the relationships between physical constraints and topology, and the resulting impact on performance. Network optimization is the process of utilizing these models in selecting topologies that best match the physical constraints of the implementation. The following subsections discuss the dominant constraints and the construction of analytic models that incorporate their effects on the no-load message latency.

7.1.1 Bisection Bandwidth Constraints

One of the physical constraints facing the implementation of interconnection networks is the available wiring area. The available wiring area is determined by the packaging technology, i.e., does the network reside on a chip, multichip module or printed circuit board. In particular, VLSI systems are generally wire-limited: the silicon area required by these systems is determined by interconnect area and the performance is limited by the delay of these interconnections. Since these networks must be implemented in three dimensions, for an N node network the available wiring space grows as $N^{\frac{2}{3}}$ while the network traffic can grow as N. Clearly machines cannot be scaled to arbitrarily large sizes without eventually encountering wiring limits. The choice of network dimension is influenced by how well the resulting topology makes use of available wiring area. Such an analysis requires performance measures that can relate network topology to the wiring constraints.

On such performance measure is the *bisection width* of a network: the minimum number of wires that must be cut when the network is divided into two equal sets of nodes. Since the primary goal is one of assessing bandwidth and resulting wiring demands, only wires carrying data are generally counted, excluding control, power, and ground signals. However, these effects are relatively easy to incorporate in the following analysis. The collective bandwidth over these wires is referred to as the *bisection bandwidth*. For example, consider a 2-D torus with $N = k^2$ nodes. Assume that each pair of adjacent nodes are connected with one bidirectional data channel of width W bits. If this network is divided into two halves, there will be $2W\sqrt{N}$ wires crossing this bisection. The factor of 2 is due to the wraparound channels. Imagine what happens when we bisect a 3-D torus with $N = k^3$ nodes. Each half comprises of $\frac{k^3}{2}$ nodes. A 2-D plane of k^2 nodes has links crossing the bisection, leading to a bisection width of $2Wk^2 = 2Wk^{n-1}$. These cases are illustrated in Figure 7.1. For the sake of clarity the wraparound links in the 3-D topology are not shown. In general, a bisection of a k-ary n-cube will cut channels from a k-ary $(n-1)$-cube leading to a bisection width of $2Wk^{n-1}$.

Using the above approach we can compute the bisection width of a number of popular topologies. These are summarized in Table 7.1. The implicit assumption here is that all of these topologies are regular, i.e., each dimension has the same radix. Consider the case of networks with $N =$

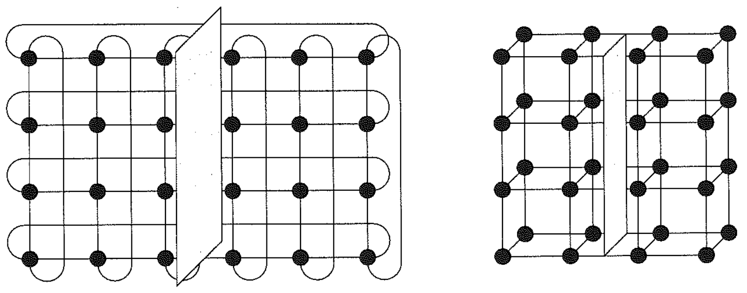


Figure 7.1. Examples of the computation of bisection width: (a) 2-D torus. (b) 3-D torus (wraparound links not shown).

$k_1 \times k_2 \times \ldots \times k_n$ nodes. In this case the minimum bisection width is orthogonal to the dimension with the largest radix. For example, this is the case shown in Figure 7.1. If the largest radix is k_m, then we have $\frac{N}{k_m}$ nodes with channels across this bisection. Thus, the bisection width can be expressed as $\frac{2WN}{k_m}$ bits.

The above computation assumes that each node devotes W pins to communication with a neighbor along a dimension. No assumptions are made about how these pins are used. In general, given a set of W pins between adjacent nodes, these may be utilized as two (opposite) $\frac{W}{2}$ bit unidirectional channels, one W-bit bidirectional channel, or one W-bit unidirectional channel. In the computing bisection width, the manner is which these channels are used is not particularly important. However, it is central to the evaluation of performance since channel widths directly impact message latency. We will see in later sections that assumptions about how these pins are used must be unambiguous and clearly stated.

7.1.2 Node Size Constraints

In addition to wiring space, a second physical constraint is the number of I/Os available per router. This is referred to as the *node size.* The construction of networks under a constant channel width for an arbitrary number of dimensions is impractical since the node size is linear in the number of dimensions. For example, consider a 2-D network and assume a baseline channel width of 16 data

Table 7.1. Examples of bisection width and node size of some common networks ($t \times t$ switches are assumed in the Omega network).

Network	Bisection Width	Node Size
k-ary n-cube	$2Wk^{n-1}$	$2Wn$
Binary n-cube	$\frac{NW}{2}$	nW
n-dimensional mesh	Wk^{n-1}	$2Wn$
Omega network	NW	$2tW$

bits, 8 control bits, and full-duplex channels, i.e., two unidirectional channels between adjacent nodes. This would correspond to 192 signal pins not including power, ground, and some miscellaneous I/Os. This is certainly feasible in current technology. For an n-dimensional network, the router will have 48 pins in each direction in each dimension for a total pin-out of $96n$. If we consider a 10-D network with the same channel width, we will require 960 pins, or close to 500 pins just for the channels in a more modest 5-D network. As we progress to 32-bit channels, the feasible network dimension becomes smaller. Thus, practical considerations place limits on channel width as a function of dimensionality [1, 3].

Even if we consider the bisection width constraints and networks being wiring area limited, it is likely that before networks encounter a bisection width limit, they may encounter the pin-out limit. This is evident from the following observation. Consider the implementation of a k-ary n-cube with the bisection width constrained to be N bits. The maximum channel width determined by this bisection width is given by

$$\text{Bisection width} = 2Wk^{n-1} = N \Rightarrow W = \frac{k}{2} \tag{7.1}$$

Note that this is the maximum channel width permitted under this bisection width constraint. A router does not have to provide this number of pins across each channel. However, if it does, the router will have a channel width of W bits between adjacent routers in each direction in each dimension for a total pin-out of $2nW = nk$. A network of 2^{20} nodes will have a pin-out of 2,048 in a 2-D configuration and a pin-out of 128 in a 4-D configuration. The former is certainly infeasible in 1997 technology for commercial single-chip packaging. In general, both the maximum number of available pins and the wiring area constraint as represented by the bisection width place upper bounds on the channel width. If the maximum number of pins available for data channels is given by P, and the bisection width is given by B, then the channel width is constrained by the following two relations. The feasible channel width is the smaller of the two.

$$W \leq \frac{P}{2n}$$
$$W \leq \frac{B}{2k^{n-1}} \tag{7.2}$$

To understand the effect of channel width on message latency consider the following. For a fixed pin-out, higher-dimensional networks will have smaller channel widths: a fixed number of pins partitioned across a higher number of dimensions. As we reduce the number of dimensions, the channel width increases. For example, if the channel width of an n-dimensional binary hypercube is W_h, then the total number of pins is nW_h (note the absence of the factor of 2 since $k = 2$ is special case with no wraparound channels). If we keep this pin-out constant, then for an m-dimensional torus with k_1^m nodes, the pin-out is $2mW_m$. The channel width of the torus is given by

$$W_m = W_h \times \frac{n}{2} \times \frac{1}{m} \tag{7.3}$$

However, if we keep the number of nodes constant, we have

$$N = 2^n = k_1^m \Rightarrow k_1 = N^{\frac{1}{m}} \tag{7.4}$$

Therefore we see that as the dimension decreases, the linear increase in channel width is offset by an exponential increase in the distance traveled by a message in each dimension. Expressions for message latency must capture these effects in enabling the analysis of trade-offs between bisection width, pin-out, and wiring length.

7.1.3 Wire Delay Constraints

As pointed out in [71], in VLSI systems the cycle time of the network is determined by the maximum wire delay, while the majority of the power is expended in driving these wires. Thus, topologies that can be embedded in two and three dimensions with short wires will be preferred. While even low-dimensional networks may logically appear to have long wires due to the wraparound connections, these can be avoided by interleaving nodes as shown in Figure 7.2 [258, 312]. However, when higher-dimensional networks are embedded in two and three dimensions, longer long wires do result. For the purposes of analyzing the effect of wire length we can determine this increase in wire length as follows [3]. The following analysis is based on an embedding in two dimensions, although it can be extended to three dimensions in a straightforward manner.

A k-ary n-cube is embedded in two dimensions by embedding $\frac{n}{2}$ dimensions of the network into each of the two physical dimensions (assuming an even number of dimensions). After embedding the first two dimensions each additional dimension increases the number of nodes in the network by a factor of k. Embedding in two dimensions provides an increase in the number nodes in each physical dimension by a factor of $\sqrt{k}$. If we ignore wire width and wiring density effects, the length of the longest wire in each physical dimension is also increased by a factor of $\sqrt{k}$. Note that if we had to account for the thickness of the wires, the length of the longest wire would actually grow faster than $\sqrt{k}$ as we increase the number of dimensions (see [3] for a good discussion). Ignoring wire width effects, the length of the longest wire in a 2-D embedding of an n-dimensional network increases by a factor of $k^{\frac{n-2}{2}}$ over the maximum wire length of a 2-D network. For a generalization to embedding in three dimensions and higher, refer to Exercise 7.3.

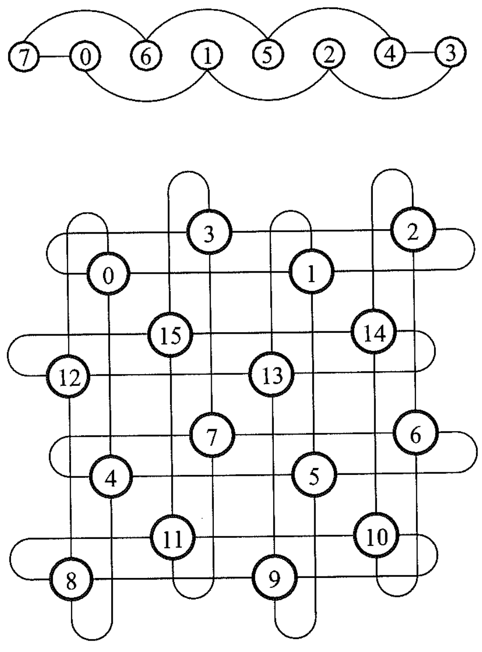


Figure 7.2. Interleaving nodes to avoid the long wire lengths due to the wraparound connections.

The delay along the wire may follow one of several models depending upon the length, the implementation medium, e.g., aluminum on silicon, copper on polyimide, etc. The three common wire delay models include the linear, logarithmic, and constant delay models.

Delay $\propto \log_e l$	Logarithmic delay model
Delay $\propto l$	Linear delay model
Delay $\propto C$	Constant delay model

where l is the wire length. These delay models are incorporated into a model of average message latency developed in the following section.

7.1.4 A Latency Model

Given the impact of the preceding physical constraints, we can now develop a model of the no-load latency of a message in a k-ary n-cube in the presence of these constraints. Such relationships between performance and implementation constraints help us make informed trade-offs in the choice of network topology. In the following we assume a wormhole-switched network, and the development follows that provided in [3].

Consider the base latency of a wormhole-switched message from Equation 2.4.

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \tag{7.5}$$

where t_r is the time to make a routing decision, t_s is the delay through the switch, and t_w is the wire delay across the physical channel. The first term in the right hand side of the expression represents the delay due to the distance between the source and destination. The major components are the routing, switch, and link delays. The routing protocols discussed in Chapter 4 implement decisions that can be quite complex relative to the no-load delay of a data flit through the switch. While the impact of this routing delay is reduced for long messages, it may be nonnegligible for small messages, and is represented by t_r. It is sometimes the case that the switch delays (t_s) are greater than the link delays, and as a result have a nonnegligible impact on the performance of the network. The development of the latency model in this section assumes switch delays dominate wire delays and follows the approach in [3]. Later in this chapter we consider the case where wire delays dominate switch delays. Finally, t_w represents the wire delay across the physical link. The above expression is based on a router architecture where the inputs and outputs are buffered. Thus, once a message pipeline has been set up, the period of this pipeline is determined by the greater of the switch delay and wire delay. If we had used an input only buffered switch, then $\max(t_s, t_w)$ would have simply been replaced by $(t_s + t_w)$.

Under random traffic, the value of the distance D above can be replaced by the average distance between any pair of nodes. With bidirectional links, in an n-dimensional network a message will travel an average of $\frac{k}{4}$ links in a dimension when k is even and $\frac{(k^2-1)}{4k}$ links when k is odd. To capture the effect of wire delays the routing, switching, and wire delays can be normalized to the wire delay between adjacent nodes in a 2-D network. This will permit analysis that would remain relevant with improvements in technology. As new packaging techniques, materials, and interconnect media evolve, wire delays between adjacent nodes in a 2-D embedding will improve. The latency expression using normalized delays will represent performance relative to this improved base 2-D case. Let the switch delay and routing delay be represented as integer multiples of the wire delay in a 2-D network: s and r, respectively. From the earlier discussion, we know the value of t_w relative to a 2-D network. Thus, the latency expression can be written as follows (for networks where k is even):

$$t_{wormhole} = n\frac{k}{4}(r + s + k^{\frac{n}{2}-1}) + \max(s, k^{\frac{n}{2}-1}) \left\lceil \frac{L}{W} \right\rceil \tag{7.6}$$

The above expression is based on a linear delay model for wires. For short wires, delay is logarithmic [71] and for comparison purposes, it is useful (though perhaps unrealistic) to have model based on constant wire delay. The above expression is normalized to the wire delay in a 2-D mesh. Therefore, in the constant delay model, wire delay simply remains at 1. In the logarithmic delay model, delay is a logarithmic function of wire length, i.e., $1 + \log_e l$ [71]. The latency expression under the logarithmic wire delay model can be written as

$$t_{wormhole} = n\frac{k}{4}\left[r + s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right] + \max\left[s, 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left\lceil \frac{L}{W} \right\rceil \tag{7.7}$$

For the constant wire delay model the latency expression is

$$t_{wormhole} = n\frac{k}{4}(r + s + 1) + s\left\lceil \frac{L}{W} \right\rceil \tag{7.8}$$

The latency expressions can be viewed as the sum of two components: a *distance component* which is the delay experienced by the header flit, and a *message component*, which is the delay due to the length of the message. The expressions also include terms representing physical constraints (wire length, channel widths), network topology (dimensions, average distance), applications (message lengths) and router architecture (routing and switching delays). Although in the above expression routing and switching delays are constants, we could model them as an increasing function of network dimension. This appeals to intuition since a larger number of dimensions may make routing decisions more complex, and increase the delay through internal networks that must switch between a larger number of router inputs and outputs. We revisit this issue in the exercises at the end of this chapter.

The above expression provides us with insight into selecting the appropriate number of dimensions for a fixed number of nodes. The minimum latency is realized when the component due to distance is equal to the component due message length. The choice of parameters such as the switching and routing delays, channel widths, and message lengths determine the dimension at which this minimum is realized. The parameters themselves are not independent and are related by implementation constraints. Examples of such analysis are presented in the following sections.

7.1.5 Analysis

In this section we focus on the selection of the optimal dimension given a fixed number of nodes. The optimal dimension realizes the minimal value of the no-load latency. The choice of the optimal dimension is dependent on the technological parameters. Wiring density, pin-out, and wire delays each individually define a dimension that minimizes the no-load latency performance of the network. The following analysis is intended to demonstrate how the topological features of the network are dependent on constraints on each these three physical features, and thus enable us to select the topology that optimizes the performance of an implementation.

Constant Bisection Width

We first consider the case where the wiring area is limited, and this limited wiring resource is represented by a fixed bisection width. For the purposes of this analysis, let us assume this fixed resource is assumed to be equivalent to that of a k-ary n-cube with $k = 2$, and single-bit channels between adjacent nodes. Substituting $k = 2$ and $W = 1$ in the expression in Table 7.1, we obtain this constant bisection width as N. Note that this value differs from the expression for the special case of the n-dimensional binary hypercube in that wrap-around channels are assumed. The following analysis could just as easily be performed with with any known bisection width. From Section 7.1.2, we know that the corresponding channel width that fully utilizes this bisection width is $\frac{k}{2}$. Thus, the latency expression under the constant bisection width constraint can be obtained by substituting for W and can be written as

$$t_{wormhole} = n\frac{k}{4}(r + s + k^{\frac{n}{2}-1}) + \max(s, k^{\frac{n}{2}-1}) \left\lceil \frac{2L}{k} \right\rceil \tag{7.9}$$

In the above latency expression we implicitly assume that all of the W signals devoted to the channel between a pair of routers is used for message transmission, i.e., half-duplex channels. We could have assumed that the W signals were organized as two unidirectional physical channels — one in each direction across the channel. In this case the channel width in the above expression would have been replaced by $\frac{k}{4}$ rather than $\frac{k}{2}$. This illustrates the necessity of being careful about what exactly a W-bit channel refers to in order to maintain fair comparisons between networks. Comparing W-bit bidirectional channels and W-bit, dual, unidirectional channels is clearly not a fair comparison. Although both instances are sometimes described as W-bit channels between adjacent nodes, the latter has twice as many signals as the former. From the perspective of packaging and wiring area demands, we are simply interested in the number of pins devoted to communication with a neighbor in a dimension. A fair comparison between two networks would ensure the same number of signals devoted to communication between neighboring nodes in a dimension.

As we increase the number of dimensions, when the network is laid out in the plane (or even in three dimensions for that matter), the number of interprocessor channels that cross the bisection increases. If this bisection width is held constant, then the individual channel width must decrease, effectively making messages longer and increasing the component of message latency due to message length. The length of the longest wire also increases with increasing dimension increasing the distance component of the latency expression for higher dimensions. For a constant number of nodes, increasing the dimension increases the wire length by a factor of $N^{\frac{1}{(n+1)n}}$. The cumulative effect of all of these trade-offs is illustrated in Figure 7.3. It is apparent that for larger-size systems, the distance component of message latency dominates at low dimensions, thus favoring networks of dimension 3 or less. Smaller message sizes also effectively increase the impact of switching and routing delays. Thus, increasing r, s, and reducing L for the larger size systems (e.g., 2^{20} and 2^{16} nodes) will further increase the optimal dimension to 4 but not much further. For smaller systems (i.e., 2^8) the optimal number of dimensions remains at three since the latency in these systems is less dependent on the distance component. This behavior appeals to intuition since messages must travel through a greater number of routers in larger systems and thus would be more sensitive to switching and routing delays. Note that the large values of latency are due to the model component that provides an exponential penalty for wire length at high dimensions.

In general, we find that the trade-offs are a bit different for a larger number of nodes versus a smaller number of nodes. Larger systems are more sensitive to switch and wire delays and when these are large, can encourage the use of a larger number of dimensions, e.g., three to six. Smaller message sizes also have the effect of increasing the impact of switch and routing delays. Smaller

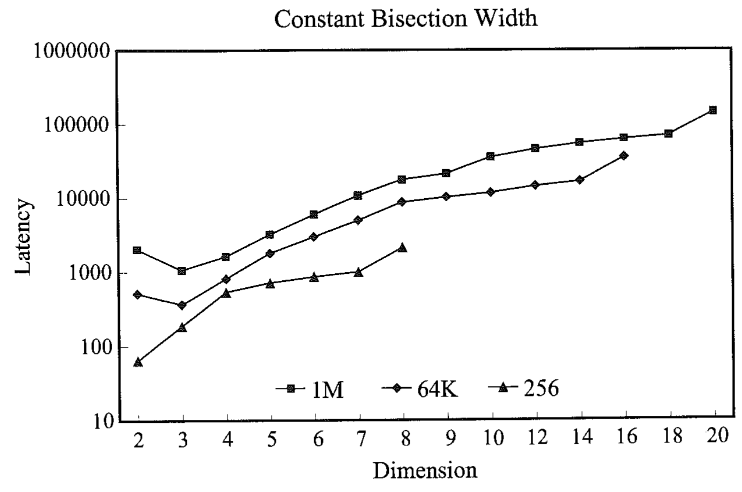


Figure 7.3. Effect of constant bisection width on the no-load message latency.

size systems are less sensitive to these delays. Rather, the relative impact of narrower channels has a greater effect on message latency than the switching and routing delays. The optimal number of dimensions for smaller systems tends to be about two to three. Finally, the use of linear wire delay models penalizes higher-dimension networks. However, use of the logarithmic wire delay models does not change the results appreciably. The optimal number of dimensions for large systems (> 1K) ranges from three to six, and for small systems (< 512) remains at two for realistic wire delay models. For small systems, the (unrealistic) constant delay model and small message sizes increases the optimal number of dimensions. However for the case of 16K and 1M nodes, the optimal number of dimensions rises to 9. These results are summarized in Tables 7.2a and 7.2b.

Finally we note that the behavior is dependent on two additional factors. The latency expressions are based on the no-load latency. The presence of link congestion and blocking within the network can produce markedly different behavior, particularly if the message destinations and lengths are nonuniform. Reference [3] presents one approach to incorporating models of network congestion into the latency expressions. A second issue is the design of the routers. If routers employ input

Table 7.2. Optimal number of dimensions for constant bisection bandwidth and: (a) Message size of 32 bits. (b) Message size of 256 bits.

(a)

Wire Delay Model	1M	16K	256
Linear	3	3	2
Logarithmic	7	6	2
Constant	9	7	3

(b)

Wire Delay Model	1M	16K	256
Linear	3	3	2
Logarithmic	4	3	2
Constant	6	4	2

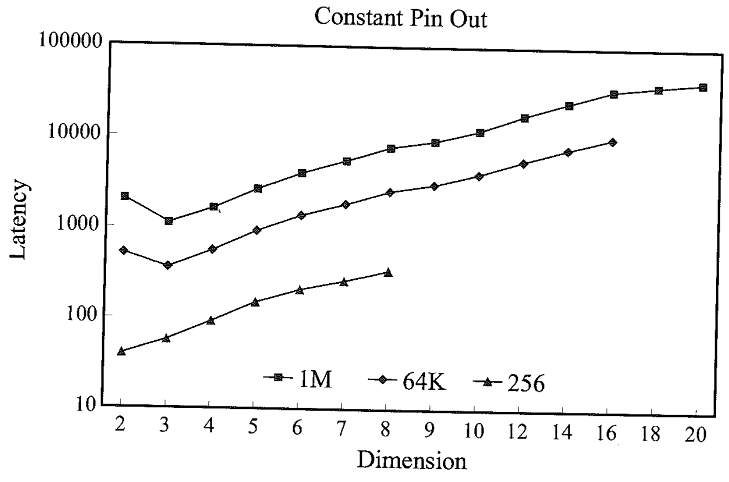


Figure 7.4. Effect of constant pin-out on the no-load message latency.

buffering, the cycle time of the message pipeline is the sum of the switch (t_s) and wire (t_w) delays. When both input and output buffering is employed, the cycle time is reduced to max (t_s, t_w). Thus, the latter model would favor higher dimensions since it is less dependent on wire delays particularly when switching delays are relatively large.

Constant Node Size

The preceding analysis can lead to designs where the individual nodes must have extremely large pin-out (often impractical in 1996–1997 technology) to make full use of the bisection bandwidth. In these instances network performance is limited not by the wiring resource but chip I/O or pin resources [1]. As we did for bisection width, we can re-write the latency expression to capture the dependency of latency on the network dimension assuming the number of chip I/Os are fixed. The following expression is written assuming linear wire delay and P pins per chip devoted to communication channels. Each router is assumed to communicate over W signals with an adjacent router in each direction in each dimension. Thus $P = 2nW$ and substituting $W = \frac{P}{2n}$ the no-load latency expression using a linear wire delay model becomes

$$t_{wormhole} = n\frac{k}{4}(r + s + k^{\frac{n}{2}-1}) + \max(s, k^{\frac{n}{2}-1})\left\lceil\frac{2nL}{P}\right\rceil \qquad (7.10)$$

The factor of 2 comes from the availability of channels in each direction within a dimension, and the use of W signals between adjacent nodes. For binary hypercubes we would have $W = \frac{P}{n}$ since there is only one direction in each dimension. A plot illustrating the optimal number of dimensions is shown in Figure 7.4 for the linear wire delay model provided in the above equation, parameterized as shown in the figure. The message size is fixed to that of a nominal-sized cache line in a shared-memory system, or medium size message (32 bytes) and we assume a nominal number of pins devoted to communication (256 pins). The optimal number of dimensions for large numbers of nodes with message size of 256 bits is 3, whereas for the smaller-sized system it is 2. If the wire delay model is

Table 7.3. Optimal number of dimensions for constant pin-out and: (a) Message size of 32 bits. (b) Message size of 256 bits.

(a)

Wire Delay Model	1M	16K	256
Linear	3	3	3
Logarithmic	9	7	4
Constant	9	8	4

(b)

Wire Delay Model	1M	16K	256
Linear	3	3	2
Logarithmic	6	4	2
Constant	9	7	3

changed to the logarithmic delay model the optimal number of dimensions increases to 6, 4, and 2 for the 1M, 16K, and 256-node systems respectively. Reducing the message size has a substantial effect further increasing the optimal number of dimensions. These results are summarized in Tables 7.3a and 7.3b.

The exponential increase in the distance in each dimension overshadows the linear increase in channel width as the network dimension is reduced. Under the constant node size constraint model, higher dimensionality is more important than wider channels [1]. The use of input and output buffering further encourages the use of higher dimensions. This model is also particularly sensitive to wire delays as shown in Figure 7.4.

This analysis is very dependent upon the structure of the physical channel. The model assumes that the number of pins devoted to communication with a neighboring node in a dimension is organized as a single bidirectional channel of width W bits. The channel width is a major determinant of message latency. The pins could have been organized as two unidirectional channels of width $\frac{W}{2}$ bits each, significantly altering the impact of message size. Our expectation is that with the use of bidirectional channels, the network would be relatively more sensitive to traffic patterns and load since messages can only cross the physical channel in one direction at a time. Such conflicts may be reduced by using full-duplex channels, or making the channels unidirectional. In the latter case the average distance a message travels within a dimensions is doubled (to $\frac{k-1}{2}$), and overall average message distance would be increased. The preceding analysis could be easily repeated for any of these relevant cases with appropriate adjustments to the expression that relates channel widths to the number of available data pins. The important point to note is that when pin resources are limited, they can be used in different ways with consequently differing impact on performance.

It is also important to note that while the analysis has been under a single constraint, the bisection width and node size are not independent. A fixed node size and dimensionality will produce a fixed bisection width. Using the model of the physical channel between adjacent routers as described in the preceding paragraphs, we observe that the channel width $W = \frac{P}{2n}$. Based on this channel width, the relationship between node size and bisection width can be written from Table 7.1 as

$$Bisection\ width = \frac{P}{n}\ k^{n-1} \tag{7.11}$$

Bisection width is linearly related to node size and inversely related to the network dimensionality. As networks are scaled to larger sizes, the technology-dependent question is one of whether bisection width or node size limits are encountered first. The answer is based on a moving target, and may be different at any given point in the technology curve.

Constant Wire Throughput

The analysis described in the preceding sections were based on the assumption that switching delays were dominant. Therefore the latency model represented these delays (and routing delays) as values normalized to the wire delay of between adjacent routers in a 2-D implementation. As the speed of operation continues to rise and systems employ larger networks, wire delays begin to dominate. This section develops the latency model in the case where it is wire delays that are dominant rather than switching delays.

When the delay along the wire is small relative to the speed of transmission through the wire media and propagation delay through the drivers, the lines can be modeled as simple capacitive loads. This is the case for lower-dimensional networks. When higher-dimensional networks are laid out in 2 or 3 dimensions, the propagation delay along the longest wire can be substantial. In reality, a signal connection actually operates as a transmission line. As the speeds of network operation increase, and high-dimensional networks are mapped to two and three dimensions producing longer wires, a more accurate analysis utilizes models of physical channel behavior based on transmission lines. Consider the following analysis [31]. A signal propagates along a wire at a speed given by the following expression.

$$v = \frac{c_0}{\sqrt{\epsilon_r}} \tag{7.12}$$

The value of c_0 is the speed of light in vacuum and ϵ_r is the relative permittivity of the dielectric material of the transmission line. If the wire is long enough relative to the signal transition times, a second signal can be driven onto the line before the first signal has been received at the source. The maximum number of bits on the wire is a function of the frequency of operation, wire length, and signal velocity. These factors are related by the following expression:

$$n = \frac{fl}{v} \tag{7.13}$$

where f is the clock frequency, l is the length of the line, and v is the signal velocity.

Given the number of dimensions and layout of the network, the maximum number of bits that can be placed on the wire can be computed from this expression. Typical values for v have been reported as 0.1 – 0.2 m/s [16, 31]. These values provide maximum wire lengths for nonpipelined channels for a 1 GHz switching speed. For $v = 0.2$ m/s and a switching speed of 1 GHz, a wire of length one meter can support the concurrent transmission of 5 bits. While link pipelining has been use in local area networks, until recently its use in multiprocessor interconnects has been quite limited. This is rapidly changing with the continuing increase in clock speeds and confluence of traditional local area interconnect and multiprocessor backplane interconnect technologies [30, 156].

While latency remains unaffected, link throughput would now be decoupled from wire delay and limited by the switching speeds of the drivers and wire media rather than physical length of the wires. Multiple bits can concurrently be in transmission along a wire. From the latency equations developed in earlier sections, it is clear that wire delays play an important role, and as a result, link pipelining fundamentally changes the trade-offs that dictates the optimal number of dimensions. Channel bandwidth is now limited by switching speed rather than wire lengths. Scott and Goodman [310] performed a detailed modeling and analysis of the benefits of link pipelining in tightly coupled multiprocessor interconnects. Their analysis utilized a base case of a 3-D unidirectional torus. To remain consistent with the preceding discussion, we apply their analysis techniques to the 2-D case and latency expressions developed above. Thus, we will be able to compare the effects of link pipelining, switch delays, wiring constraints, and pin-out on the optimal choice of network dimension. The preceding latency expression for wormhole-switched networks can be rewritten as

$$t_{wormhole} = n\frac{k}{4}(r + s + w_{avg}) + s\left\lceil\frac{L}{W}\right\rceil \quad (7.14)$$

The above expression is normalized to the delay between adjacent nodes in a 2-D network. The value of w_{avg} is the average wire delay expressed relative to this base time period. Note the latency term due to message length is now dependent only on the switching delay which determines the rate at which bits can be placed on the wire.

While nonpipelined networks have their clock cycle time dictated by the maximum wire length, pipelined networks have the clock cycle time determined by the switching speed of the routers. The distance component of message latency depends on the length of the wires traversed by the message. However, the wire length between adjacent nodes will depend on the dimension being traversed. The average wire length can be computed as follows [310].

To avoid the long wires due to the wraparound channels, node positions are interleaved as shown in Figure 7.2. Recall that the wire length is computed relative to the wire length between two adjacent nodes in a 2-D mesh. Let the wire length between two nonadjacent nodes in Figure 7.2 be denoted by l_2. In this case there are $(k-2)$ links of length l_2 and two links of length equal to the base wire length in a 2-D layout (see the 1-D interleaved placement in Figure 7.2). If we assume that this base wire length is one unit and that l_2 is twice the base wire length between adjacent nodes, we can write the average wire length in this dimension as

$$\text{Average wire length} = l_2\frac{k-1}{k} = 2\frac{k-1}{k} \quad (7.15)$$

In practice, l_2 may be a bit longer than twice the base wire length. In the following we retain the variable l_2 in the expressions. Consider topologies with an even number of dimensions. When an n-dimensional topology is mapped into the plane, $\frac{n}{2}$ dimensions are mapped into each physical dimension. As each pair of dimensions are added, the number of nodes in each physical dimension is increased by a factor of k and the average wire length in each dimension grows by a factor of k. Mapping each additional pair of dimensions increases this average wire length in each physical dimension by a factor of k. Consider all the logical dimensions mapped to one of the two physical dimensions. The sum of the mean wire lengths in each of these dimensions is given by

$$l_2\frac{k-1}{k} + l_2\frac{k-1}{k}k + l_2\frac{k-1}{k}k^2 + \ldots + l_2\frac{k-1}{k}k^{\frac{n-2}{2}} \quad (7.16)$$

We can similarly compute the sum of the mean wire length in each dimension mapped to the other physical dimension. The sum of these two expressions divided by the total number of dimensions gives the average wire length of the topology. This simplified expression appears as

$$l_2\frac{k-1}{k}(1 + k + \ldots + k^{\frac{n-2}{2}})\frac{2}{n} \quad (7.17)$$

Using the sum of the geometric series and simplifying, we have

$$w_{avg} = \frac{2l_2(N^{\frac{1}{2}} - 1)}{nk} = \frac{4(N^{\frac{1}{2}} - 1)}{nk} \quad (7.18)$$

Substituting in the equation for latency, we have an expression for the latency in a link-pipelined network that is embedded in two dimensions.

Optimal Number of Dimensions — Summary

To generate an intuition about the effect of physical constraints on network performance we can make the following observations. Message latency can viewed as the sum of two components: a distance component and a message size component represented as

$$\text{Message latency } = \alpha \times Dist(n) + \beta \times Msg\_Length(n) \tag{7.19}$$

For a fixed number of nodes, the function $Dist(n)$ decreases as the network dimension increases. Under bisection width and constant pin-out constraints the function $Msg\_Length(n)$ increases with increasing network dimension since channel widths decrease. Minimum latency is achieved when the components are equal. The coefficient of the distance component is a function of switching, routing, and wire delays: α determines how fast the distance component of latency reduces with increasing dimension. This appeals to intuition as large switching delays encourage the use of a smaller number of intermediate routers by a message, i.e., a larger number of dimensions. Similarly, β, which is a function of switching and wire delay, determines how fast the message component increases with increasing network dimension. The optimal number of dimensions is determined as a result of specific values of router architecture and implementation technology that fix the values of α and β.

It is important to note that these models provide the optimal number of dimensions in the absence of contention. This analysis must be coupled with the detailed performance analysis usually via simulation as described in Chapter 9, or analytic models of contention(e.g., as in [1, 3, 128]).

7.1.6 Packaging Constraints

It is clear that physical constraints have a substantial impact on the performance of the interconnection network. In practice, electronic packaging is a major determinant of these physical constraints. By *packaging* we refer to the hierarchy of interconnections between computational elements. Systems start with communicating elements sharing a silicon surface. Bare die may then be packaged in single-chip carrier or a multichip module [302], which in turn may be mounted on one or both sides of a printed circuit board. Multiple boards are mounted in a card cage, and multiple card cages may be connected and so on. This is graphically illustrated in Figure 7.5. The materials and fabrication technology at each level of this hierarchy are distinct, leading to interconnects with very different physical characteristics and constraints. For example, wire pitch may vary from a few micrometers on a die to a few hundred micrometers on a printed circuit board. Such parameters clearly determine available wiring area and therefore achievable bisection bandwidth. With multichip modules and area array I/Os the number of I/Os available out of a die becomes proportional to chip area rather than chip perimeter [301]. As the preceding analysis has demonstrated, the choice of topology is strongly influenced by such physical constraints and therefore by the packaging technology.

Packaging technology continues to evolve, keeping pace with rapid advances in semiconductor technology. The advent of new packaging technologies, e.g., low-cost multichip modules (MCM), 3-D chip stacks, etc. will lead to major changes in the relationship between these physical constraints, and as a result significantly impact the cost, performance, and reliability of the next generation of systems in general, and multiprocessor interconnects in particular [84, 123]. An accurate analysis of the effect of packaging technology and options will enable the identification of topologies that effectively utilize available packaging technology, where "form fits function [71]."

From the point of view of the network topology, the fundamental packaging question concerns the placement of die, package, board, and subsystem boundaries and the subsequent performance impact of these choices. Die boundaries may be determined in part by technology as we see advances

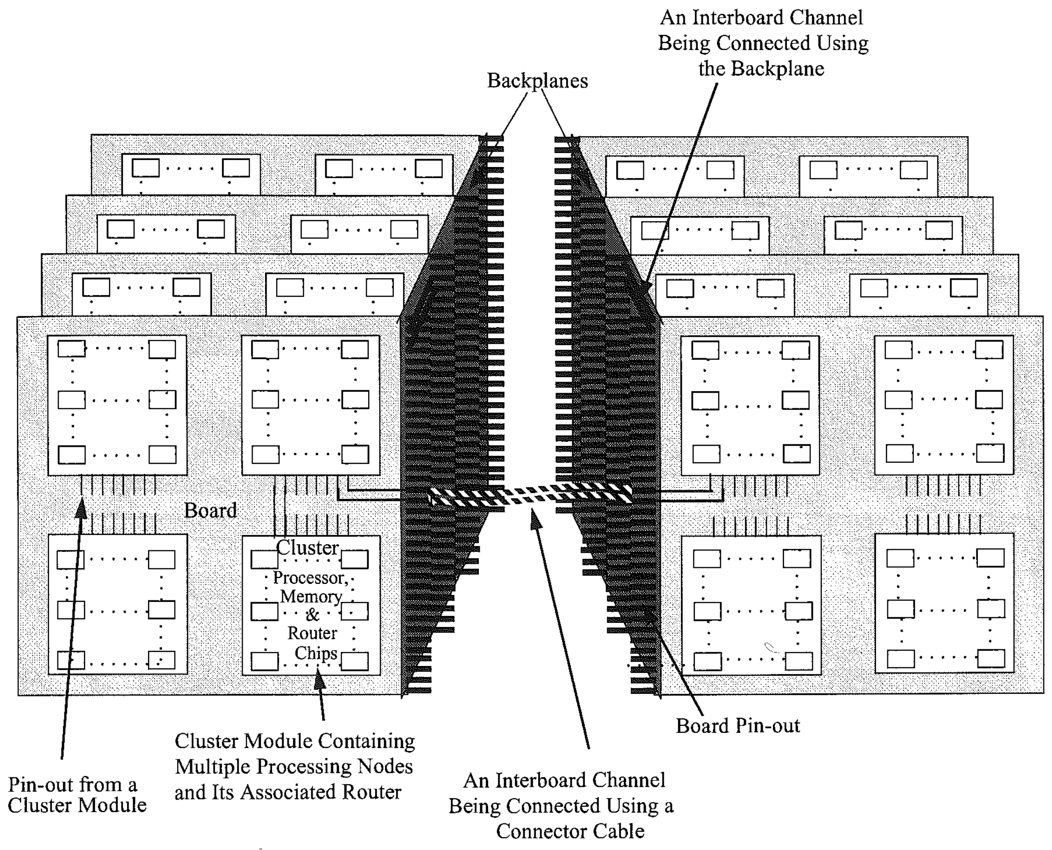

An Interboard Channel Being Connected Using the Backplane
Backplanes
Board
Cluster
Processor, Memory & Router Chips
Cluster Module Containing Multiple Processing Nodes and Its Associated Router
Pin-out from a Cluster Module
An Interboard Channel Being Connected Using a Connector Cable
Board Pin-out

Figure 7.5. Illustration of a packaging hierarchy. (From [19].)

that place multiple processors and the associated routers and interconnect on a single chip. The placement of other boundaries such as multichip carriers and boards are determined by trade-offs in cost, performance, thermal density, reliability, pin-out, etc. For example, the network used to connect 16 processors on a single chip may be quite different from that used on a printed circuit board to connect 64 such chips. Therefore these partitioning decisions are critical. To make the most effective use of packaging technology, we need accurate models of performance to predict the impact of a specific choice of partitions for a given network topology. In a more general setting, such models have been used as elements of a "design for packageability" style of system-level design [84]

Technology projections from groups such as the Semiconductor Industries Association provide assessments of the evolution of implementation technology parameters such as feature size, wafer sizes, clock rates, etc. There have been several recent efforts aimed at assessing the impact of packaging technology on the design and optimization of future network topologies. As a representative example of a relatively new arena of work, the following discussion presents one such approach to assessing the impact of packaging technology on network design.

Basak and Panda [18, 19] provide a model and framework for designing clustered multiprocessor systems in a two-level packaging hierarchy. A k-ary n-cube cluster-c topology is proposed, where each cluster consists of c nodes. While the intracluster network is left undefined and open, the intercluster network is a k-ary n-cube with k^n clusters for a total of $N = k^n \times c$ nodes. With a fixed number of nodes, by varying the number of clusters one can change the physical characteristics of the intercluster and intracluster networks. For example, with single-chip nodes the intracluster network may occupy the surface a board while the intercluster network may appear between boards. In the authors' terminology, the state of the art in packaging at each level offers or supplies physical resources such as bisection bandwidth and pin-out capability. A specific clustering of nodes produces a demand for bisection width, area, and pin-out at each level, e.g., board or chip. The proposed design paradigm is one of reconciling the demand and supply in the most efficient manner possible. Towards this end, analytic models are developed for supplied and demanded resources such as bisection bandwidth, node pin out, and area. Optimization algorithms can utilize these models to produce the most efficient designs.

For example, the demanded bisection bandwidth of an N node system is estimated as follows. Assume that each node injects λ bits/cycle into the network, where a cycle is the wire delay between adjacent routers. As in preceding sections this cycle time is assumed to be normalized to that of the delay between adjacent nodes in a 2-D mesh. On the average, we have $c\lambda$ bits generated within a cluster. With random destinations, a $(1 - \frac{c}{N})$ fraction of these injected bits traverse the intercluster network, and half of this traffic crosses the bisection. Aggregating over all nodes, the demanded bisection bandwidth of a k-ary n-cube cluster-c topology in bits/cycle is

$$\text{Demanded bisection bandwidth} = \frac{N\lambda(1 - \frac{c}{N})}{2} \tag{7.20}$$

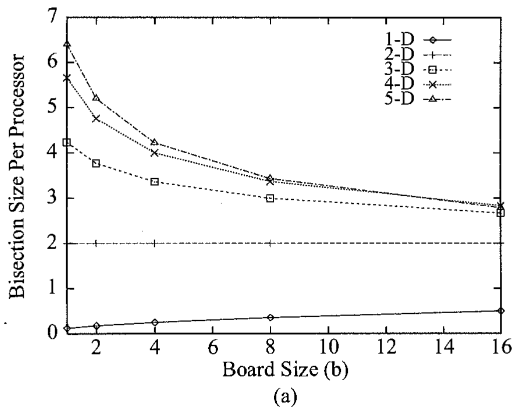
As described in [19], the value of λ can be derived from various projections of computation to communication ratios, processor speeds, and emerging network link speeds. Typical estimates can be found in [298]. Now consider the available bisection bandwidth of a packaged clustered network. From the expressions in Table 7.1, we have the bisection bandwidth of the intercluster network as

$$\text{Bisection bandwidth} = \frac{2\frac{N}{c}W}{k_{max}} \tag{7.21}$$

In a regular network where $k_1 = k_2 = \ldots = k_n, k_{max} = (\frac{N}{c})^{\frac{1}{n}}$. The issue here is that the channel width is limited by the pin-out at the chip or board level. In the following, the constraints on the channel width are derived based on a regular network where $k_1 = k_2 = \ldots = k_n$. The extension to distinct radices is straightforward [19]. Consider boards with a capacity of $B = b^n$ clusters, where $b < k$, i.e., no dimension can completely fit on a board and traversal of any dimension forces a interboard communication. Along each dimension we have b^{n-1} nodes with channels traversing interboard boundaries (since the cross section in this dimension has b^{n-1} nodes). Including channels in both directions, the total number of signals coming off the board is given by

$$\text{Board pinout requirements} = 2\sum_{i=1}^{n} b^{n-1}W = 2nb^{n-1}W \tag{7.22}$$

By equating the above to the number of available board-level pins, we can derive the constraint on channel width. The number of available pins may be a function of the perimeter of the board as found in conventional technology, or it may be proportional to the area of the board representing the use of free-space optical communication or the use of elastomeric connectors [255]. The above derivation is based on the assumption that the intraboard bisection width is not the limiting factor



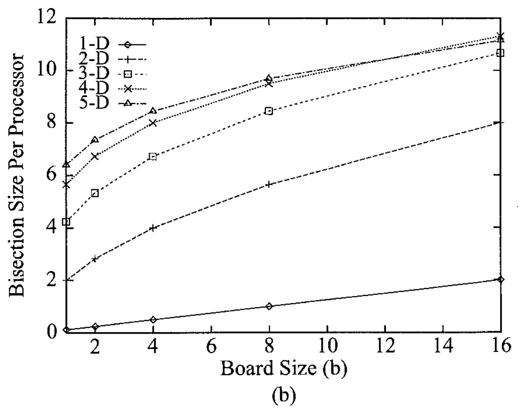


Figure 7.6. Examples of the effect of packaging technology: (a) Periphery pin-out technology. (b) Surface pin-out technology. (From [19].)

on channel width, but rather the interboard pin-out is the limiting factor. This appears to be reasonable in 1997 technology. The alternative is easily modeled by using the intraboard bisection bandwidth as the limiting factor for channel width. Assuming a maximum available board-level pin-out of P_B, we have

$$W = \frac{P_B}{2nb^{n-1}} = \frac{P_B}{2nBb^{-1}} \tag{7.23}$$

Note that B, the number of clusters on a board is a function of the of the board size and cluster size, and therefore a function of packaging technology. By substituting the above expression for channel width in the expression for bisection width, the dependency between bisection width, board size, cluster size, and pin-out can be studied. These expressions relate parameters that define a generation of packaging technology with architectural features that define the network in manner that permits the analysis of the impact of packaging constraints.

An example of the trade-offs that can be studied is illustrated in Figure 7.6. Based on the preceding expressions, these curves from [19] illustrate the relationship between the board size and offered intercluster bisection bandwidth on a per processor basis. This relationship is shown for the cases where the number of I/Os is proportional to board periphery and board area. For the former case, the analysis indicates that for higher-dimensional networks, smaller board sizes offer a higher per-processor bisection bandwidth. The converse is true for the case where the number of I/Os is proportional to board area. A detailed study of the relationships between network and packaging parameters, as well as a precise definitions of the packaging parameters captured in these models can be found in [19].

7.2 Router Architectures

The router architecture is largely determined by the switching technique that is supported. The majority of modern commercial routers found in high performance multiprocessor architectures and emerging switched networks for workstation clusters utilize some form of cut-through switching

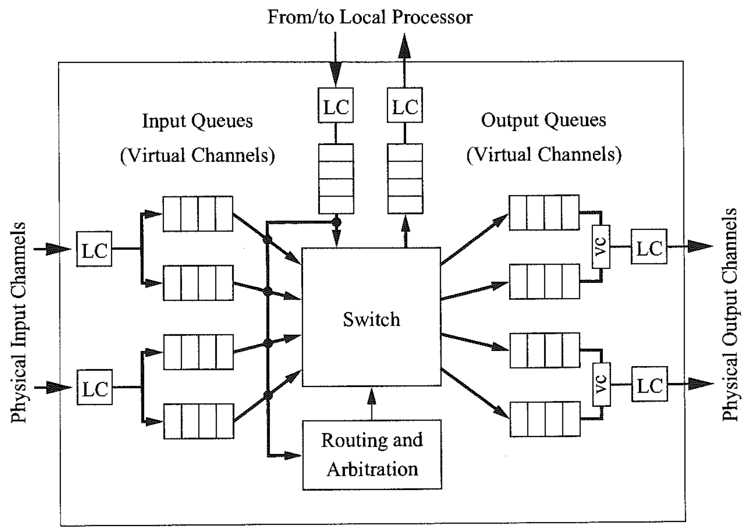


Figure 7.7. Canonical router architecture.

or some variant of it, e.g., VCT switching, wormhole switching, buffered wormhole switching, etc. Therefore this chapter largely focuses on a discussion of issues and designs of such routers. While the router implementations for wormhole and VCT switching differ in many of the architectural trade-offs, they share many common features derived from the use of some form of cut-through switching. The common issues and features facing the design of routers can be categorized as intrarouter or interrouter, and are discussed below. This discussion is followed by descriptions of recent router designs, emphasizing their unique features and reinforcing the commonly held trade-offs.

7.2.1 Intrarouter Performance

In an effort to capture the commonalities and enable quantifiable comparisons, Chien [57] developed an abstract model for the architecture of routers in wormhole-switched k-ary n-cubes. This model is largely concentrated on the intrarouter architecture while the performance of interrouter link operation is very sensitive to packaging implementations. The basic wormhole router functions can be captured in an abstract router architecture as shown in Figure 7.7. We are interested in the implementation complexity of each of the components in the figure.

- *Crossbar switch.* This component is responsible for connecting router input buffers to router output buffers. High-speed routers will utilize crossbar networks with full connectivity, while lower-speed implementations may utilize networks that do not provide full connectivity between input buffers and output buffers.
- *Link controller (LC).* Flow control across the physical channel between adjacent routers is implemented by this unit. The link controllers on either side of a channel coordinate to transfer flow control units. Sufficient buffering must be provided on the receiving side to account for delays in propagation of data and flow control signals. When a flow control event signaling a

full buffer is transmitted to the sending controller, there must still be sufficient buffering at the receiver to store all of the phits in transit, as well as all of the phits that will be injected during the time it takes for the flow control signal to propagate back to the sender. If virtual channels are present, the controller is also responsible for decoding the destination channel of the received phit.

- *Virtual channel controller (VC).* This component is responsible for multiplexing the contents of the virtual channels onto the physical channel. With tree-based arbitration, the delay can be expected to be logarithmic in the number of channels.

- *Routing and arbitration unit.* This logic implements the *routing function.* For adaptive routing protocols, the message headers are processed to compute the set of candidate output channels, and generate requests for these channels. If relative addressing is being used in the network, the new headers, one for each candidate output channel, must be generated. For oblivious routing protocols header update is a very simple operation. Alternatively, if absolute addressing is used, header processing is reduced since new headers do not need to be generated.

  This unit also implements the *selection function* component of the routing algorithm: selecting the output link for an incoming message. Output channel status is combined with input channel requests. Conflicts for the same output must be arbitrated (in logarithmic time), and if relative addressing is used, a header must be selected. If the requested buffer(s) is (are) busy, the incoming message remains in the input buffer until a requested output becomes free. The figure shows a full crossbar that connects all input virtual channels to all output virtual channels. Alternatively, the router may use a design where full connectivity is only provided between physical channels and virtual channels arbitrate for crossbar input ports. Fast arbitration policies are crucial to maintaining a low flow control latency through the switch.

- *Buffers.* These are FIFO buffers for storing messages in transit. In the above model, a buffer is associated with both the input physical channels and output physical channels. The buffer size is an integral number of flow control units. In alternative designs, buffers may be associated only with inputs (input buffering) or outputs (output buffering). In VCT switching sufficient buffer space is available for a complete message packet. For a fixed buffer size, insertion and removal from the buffer is usually not on the router critical path.

- *Processor interface.* This component simply implements a physical channel interface to the processor rather than to an adjacent router. It consists of one or more injection channels from the processor and one or more ejection channels to the processor. Ejection channels are also referred to as delivery channels or consumption channels.

While the above router model is representative of routers constructed to date, router architectures are evolving with different implementations. The routing and arbitration unit may be replicated to reduce arbitration time and channels may share ports on the crossbar. The basic functions appear largely unaltered but with distinct implementations to match the current generation of technology.

If the routers support adaptive routing, the presence of multiple choices makes the routing decision more complex. There is a need to generate information corresponding to these choices and to select among these choices. This naturally incurs some overhead in time as well as resources, e.g., chip area. Similarly, the use of virtual channels, while reducing header blocking delays in the network, makes the link controllers more complex by requiring arbitration and more complex flow control mechanisms.

Table 7.4. A parameterization of the component delays in a wormhole-switched router. (From [57].)

Module	Parameter	Gate Count	Delay
Crossbar	P (ports)	$O(P^2)$	$c_0 + c_1 \times \log P$
Flow control unit	None	O(1)	c_2
Address decoder	None	O(1)	c_3
Routing decision	F (freedom)	$O(F^2)$	$c_4 + c_5 \times \log F$
Header selection	F (freedom)	$O(\log F)$	$c_6 + c_7 \times \log F$
VC controllers	V (#VCs)	$O(V)$	$c_8 + c_9 \times \log V$

The two main measures of intrarouter performance [57] are the routing latency or header latency, and the flow control latency. From Figure 7.7 it is apparent that the latency experienced by the header flit(s) through a router is comprised of several components. After the header has been received, it must be decoded and the connection request generated. Since multiple headers may arrive simultaneously, the routing and arbitration unit arbitrates among multiple requests and one of the headers is selected, and routed. This involves computing an output and if it is available, setting the crossbar. The updated header and subsequent data may now be driven through the switch, and will experience some multiplexing delay through the link controllers as they multiplex multiple virtual channels across the physical channel. Once a path has been set up, data flits may now flow through the router with no further involvement with the routing and arbitration unit, but may compete for physical channel bandwidth with other virtual channels.

The flow control latency determines the rate at which flits can be transmitted along the path. In the terminology of pipelines, the flow control latency is the message pipeline stage time. From the figure, we can see that the flow control latency experienced by a data flit is the sum of the delay through the channel flow control, the time to drive the flit through the crossbar, and the multiplexing delay experienced through the link controller, as well as the time to read and write the FIFO buffers. The delay through the link controller is often referred to as the *flit multiplexing delay* or *channel multiplexing delay*. A general model for these delays is quite difficult to derive since the latencies are sensitive to the implementation. One approach to developing such model for wormhole-switched networks is described in [57]. A canonical model of a wormhole-switched router is developed, containing units for address decoding, flow control, and switching. Implementations of the various components (e.g., tree of gates, or selectors for each output) are modeled and expressions for the implementation complexity of each component are derived, but parameterized by technology dependent constants. The parameterized model is illustrated in Table 7.4. By instantiating the constants with values based on a particular implementation technology, realistic delays can be computed for comparing alternative designs. In [57], the values are instantiated for a class of implementations based on gate arrays to derive estimates of flow control latency through the router. The utility of the model stems from the application to many classes of routers. Other router architectures using similar components for partially adaptive or fully adaptive routing can be constructed, and the parameterized model used to estimate and compare intrarouter latencies. In fact, comparisons are possible across technology implementations.

A major determinant of the intrarouter delays is the size and structure of the switch. The use of a crossbar switch to connect all input virtual channels to all output virtual channels is feasible for low-dimensional networks with a small number of virtual channels. A 2-D network with four

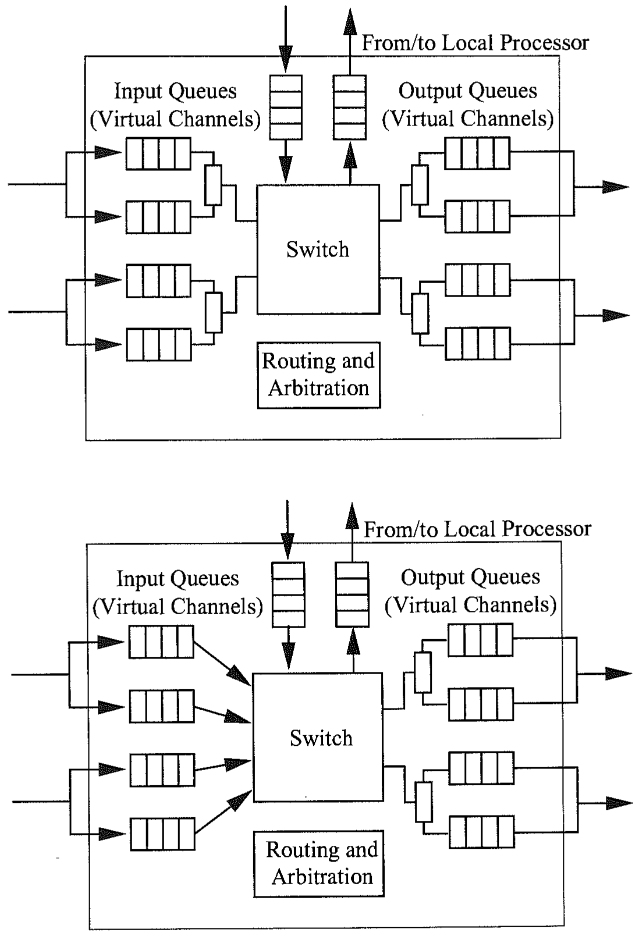

From/to Local Processor
Input Queues (Virtual Channels)
Output Queues (Virtual Channels)
Switch
Routing and Arbitration
From/to Local Processor
Input Queues (Virtual Channels)
Output Queues (Virtual Channels)
Switch
Routing and Arbitration

Figure 7.8. Alternative router switch organizations.

virtual channels per link would require a 16×16 crossbar and a 3-D network would require a 24×24 switch. Considering that these channels may be byte or word wide, it is apparent that alternatives begin to become attractive. One alternative is to have the switch size determined by the number of physical channels. The data rate into and out of the switch is limited by the bandwidth of the physical channels, while virtual channels decouple buffer resources (and therefore messages) from the physical channel. Thus, channel utilization can remain high, while the construction of crossbar switches remains feasible for higher-dimensional networks. However, input messages must now arbitrate for both switch inputs and outputs. In adaptively routed networks, this arbitration cost grows quadratically with the degree of adaptivity. An intermediate organization provides switch inputs for each input virtual channel, but outputs only for each physical channel. Thus, arbitration is simplified by only having inputs arbitrate for ports on one side of the switch. These alternatives are illustrated in Figure 7.8.

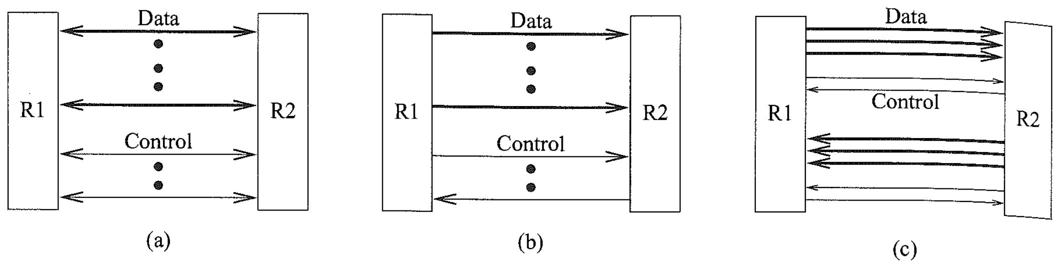


Figure 7.9. An example of: (a) Half-duplex organization. (b) Unidirectional organization. (c) Full-duplex organization.

7.2.2 Physical Channel Issues

The behavioral aspects of interrouter flow control, particularly in the presence of virtual channels is discussed at some length by Dally [73] and Bolding [31] as well as in detail in the descriptions of the individual routers. The following discussion of the issues in physical channel design is largely drawn from these efforts.

In Chapter 2 we have seen that flow control is performed at two levels. Message flow control manipulates logical units of information such as flits. These logical units are the granularity of information for sharing network resources such as interrouter links, intrarouter data paths, and allocating buffers. For example, when a virtual channel competes for, and acquires a physical link, a flit is transferred in its entirety before the control of the channel is relinquished. The transfer of a single flit is not interleaved with other flits. The granularity of a flit determines the effectiveness with which messages share buffers, links, and datapaths. The considerations and performance implications are similar to those facing the choice of thread granularity for multithreaded programs sharing a CPU.

Physical channel flow control may require several cycles to transfer a flit across the channel, e.g., a 64-bit flit across a 16-bit-wide physical channel. In this case, the physical flow control operates on 16-bit phits. By making the flit and phit sizes equivalent, simpler protocols can be used for operating the physical channel. The synchronization of physical transfers can also be used to signify the transfer of flits and the availability of buffer space. Otherwise the message flow control across the channel for blocking messages and allocating and freeing buffers must operate at a different rate than the physical channel signaling rate, increasing the complexity of the router interfaces and intrarouter data paths. In packet-switched and VCT-switched networks, buffer allocation is clearly at the level of logical units such as packets, while channel flow control is typically at finer granularities. In circuit-switched networks, the issue of flow control between routers is encountered during path setup. Once the hardware path has been established message flow control operations are performed between the source and destination routers.

The fixed number of pins devoted to communication between adjacent routers may be organized in several ways. The channel may be organized as (1) a bidirectional half-duplex channel, (2) a unidirectional channel, or (3) a bidirectional full-duplex channel. An example of the three alternatives is illustrated in Figure 7.9. The use of bidirectional half-duplex or unidirectional channels maximizes channel widths. With the exception of pins devoted to control, message transmission between adjacent routers can make full use of the available pins between two routers. However, with unidirectional channels, the topology must be a tori or similar "wrapped" topology to ensure that all

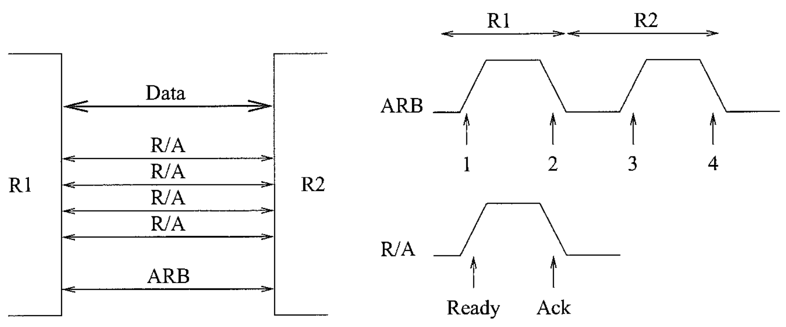


Figure 7.10. An example of the operation of a half-duplex channel.

pairs of nodes may communicate. As a result, the use of unidirectional channels doubles the average distance traveled by a message in tori, increasing the distance component of message latency. Half-duplex links have the disadvantage that both sides of the link must arbitrate for the use of the link. The arbitration must be fair and provide access to both sides of the channel when data are available to be transmitted. Fair arbitration requires status information to be transmitted across the channel to indicate the availability of data to be transmitted. Such traffic may consume bandwidth through messages (in-band transmission) or may be signaled using dedicated pins (out-of-band transmission). We would like this arbitration to be efficient in that the link bandwidth devoted to arbitration signals is minimized. In addition to consuming bandwidth, this arbitration time can increase the flow control latency across the link reducing link utilization as traffic increases. In contrast, this arbitration overhead is avoided in unidirectional full-duplex links, and therefore links can generally be run faster. However, channel widths are reduced i.e., approximately halved as bandwidth is statically allocated in each direction. When the data are being transmitted in only one direction across the channel, 50% of the pin bandwidth is unused. A full-duplex link is only fully utilized when messages are being transmitted in both directions. Depending on the exact pin counts and channel protocol, this organization also serves to roughly double message length and the corresponding component of message latency.

If virtual channels are used, additional bits/signals must be used to distinguish between virtual channels that share the physical channel. These additional bits may use bandwidth or be transmitted across dedicated control pins. An example of the operation of a physical channel that supports multiple virtual channels is provided in Example 7.1.

Example 7.1

The Network Design Frame [79] utilizes half-duplex physical channels to support two virtual channels. The sequence of operations involved in a transfer across this channel are illustrated in Figure 7.10. The channel consists of an 11-bit bidirectional data channel. At power up, one side of the channel is the default owner and has the privilege to transmit information. When the idle side of the channel wishes to transmit a flit, a request is generated by driving the ARB signal high. Channel ownership is transferred when the current owner drives the signal low, and roles are reversed. For example, in the figure $R1$ owns the channel at time 1 and ownership is transferred to $R2$ at time 2. Now $R1$ drives the ARB signal high at time 3 to request ownership and becomes the owner again at time 4. A flit transfer is synchronized

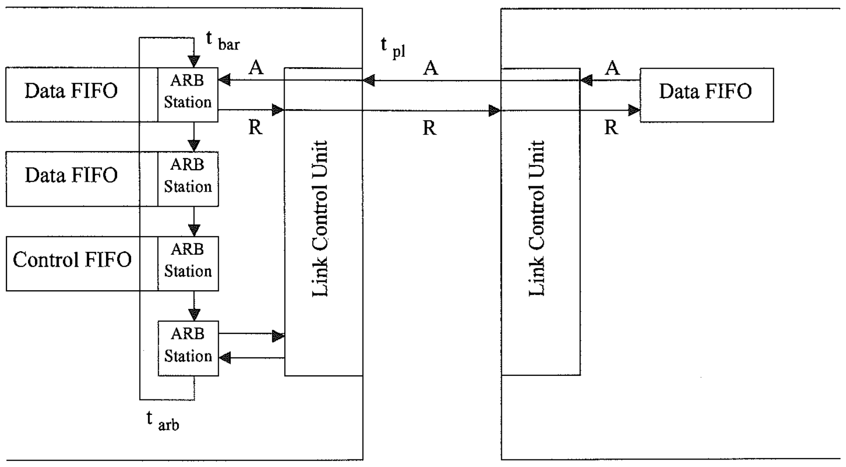


Figure 7.11. An example of overlapping channel arbitration and channel transmission.

using a single request/acknowledge (R/A) signal for each virtual channel in each direction across the physical channel, leading to a total of four R/A signals. A transfer is initiated by driving the corresponding R/A signal high. When the receiving side of the channel is ready to accept another flit, the receiver pulls the R/A signal low.

In the Network Design Frame, one virtual channel has a higher priority than the other. Demand-driven, fair allocation of bandwidth across the physical channel to multiple virtual channels on both sides of the link will require fast, fair arbitration schemes. Ideally we would like arbitration delays to be completely overlapped with transmission and therefore completely hidden. The following example provides one solution for hiding arbitration delays in self-timed router architectures.

Example 7.2

The channel structure and operation of the Ariadne router is illustrated in Figure 7.11 [7]. The operation of this channel is also self-timed, and each virtual channel is granted access to the physical channel using a demand-driven mutual exclusion ring [303]. Within a router output port, each virtual channel owns an arbitration station on the mutual exclusion ring. In addition there is a station on the ring corresponding to the port on the other end of the channel. A signal representing the privilege to drive the physical channel circulates around this ring. The cycle time across a physical link is determined by the interaction between two activities. The first is the arbitration among virtual channels competing for the physical channel. The second is the handshake cycle time for transmission of a flit across the physical channel. A barrier synchronization between these activities can be implemented as part of the channel flow control mechanism as illustrated in Figure 7.11.

The efficiency of this approach can be captured in the following expression. Let t_{arb} represent the period of the arbitration oscillator, i.e., the time for the privilege to circulate through all stations on one side of the channel. Let t_{pl} represents the handshake time to transfer a phit over the physical link and place a request to send the next phit. When a virtual channel is granted the privilege from its ARB Station, phit transmission is stalled until the previous transmission has been completed. When transmission begins, the privilege can be released, serving to overlap transmission latency with the next round of arbitration. When more than one channel is contending for the link, arbitration can be completely hidden. When only one virtual channel is transmitting, the privilege may circulate all the way around the ring before the next phit can be transmitted, potentially slowing down transmission. The privilege is available at intervals of t_{arb}, except the first time when it is available after $\frac{3}{4}t_{arb}$. Thus in general, with one virtual channel transmitting, the privilege is available at points in time of $\frac{3}{4}t_{arb} + mt_{arb}$ for some $m > 0$. Once the physical cycle is complete and the privilege has returned, a small barrier synchronization latency t_{bar} is incurred before the privilege is again released. Thus, the link cycle time with one active virtual channel is

$$t_c = \frac{3}{4}t_{arb} + mt_{arb} + t_{bar}$$

for the minimum m such that $\frac{3}{4}t_{arb} + mt_{arb} + t_{bar} > t_{pl}$. The value of $t_c - t_{pl}$ represents the degree to which arbitration cannot be overlapped with flit transmission across the link.

At low loads, half-duplex links are advantageous since they deliver the full bandwidth of the available pins to the messages whereas full-duplex links remain under utilized. However, at high loads the full bandwidth of the link is utilized in both cases, although arbitration traffic in the case the half-duplex links will produce relatively lower utilization. The disparity in performance is greater when link pipelining is employed [31]. When channel ownership must be switched in the case of a half-duplex link, transmission can begin only after the channel has been drained of the bits currently in transit. In addition, the first phit transmitted in the new direction incurs the full latency of the physical link. Thus, arbitration overheads relative to full-duplex operation correspondingly increase since a greater percentage of time is spent in switching directions across the channel. A second problem is in the propagation of status information. With pipelined links, the received status of flow control information may not reflect the actual status at the time it is received. This is an issue for both half-duplex and full-duplex pipelined links.

For example, the transmission of flow control information to indicate the lack of buffer space arrives too late at the sender: multiple phits are already in transit and will be dropped due to lack of space. However, full link utilization can be maintained without loss with a buffer on the receiver of sufficient depth to store all the phits in transit across the channel. The relationship between the buffer depth, wire delay, bandwidth, etc. can be derived as follows [107]. Assume that propagation delay is specified as P ns per unit length and the length of the wire in the channel is L units. When the receiver requests the sender to stop transmitting, there may be LPB phits in transit on a channel running at B Gphits/s. In addition, by the time the flow control signal propagates to the sender, an additional LPB phits may be placed in the channel. Finally, if the processing time at the receiver is F ns (e.g., flow control operation latency), this delay permits the sender to place another FB phits on the channel. The available buffer space in the receiver must be large enough to receive all of these phits when a *stop* signal is transmitted to the sender. These *slack buffers* [314] serve to hide the wire latency and the flow control latency. The size of the buffers can be estimated as follows.

$$\text{Available buffer space} = 2LPB + FB \text{ phits} \tag{7.24}$$

In cases where long wire latencies dominate, the use of link pipelining favors the use of full-duplex channels, particularly under high loads. This is typically the case in the emerging class of low latency interconnects for workstation/PC clusters. To reduce the wiring costs of long interconnects as well

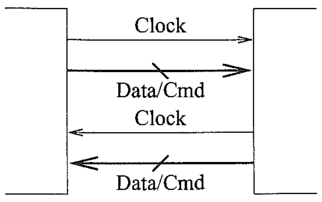


Figure 7.12. An example of the operation of a full-duplex channel with data/command encoding.

as maintain high link utilization, full-duplex channel techniques have been developed using slack buffers and pin-efficient flow control. An example is described in Example 7.3.

Example 7.3

The structure of an example full-duplex channel is shown in Figure 7.12. This channel is used in the routing fabric of Tandem's ServerNet [156], and differs from the preceding example is several ways. Physical channel flow control is synchronous and the clock is transmitted across the link. At the transmitting end of the channel, the clock is sent through similar delay paths as the data so that it may be used to sample the data at the receiver. Buffer management and message-level flow control is via commands that are transmitted in the reverse direction across the 9-bit data/command channel. The 9-bit encoding provides for 256 data symbols and 20 command symbols. The specific encoding used in ServerNet is shown below.

Bit 8	Bit 7	Bit 6	Function
0	0	0	Command
0	0	1	Error
0	1	0	Error
1	0	0	Error
0	1	1	Data <7:6> = 00
1	0	1	Data <7:6> = 01
1	1	0	Data <7:6> = 10
1	1	1	Data <7:6> = 11

The most significant 3 bits determine how the remaining 6 bits will be interpreted. For command words, these 6 bits are encoded as a 3-of-6 code where all command words have exactly three 1s and three 0s, resulting in 20 such command words. Examples of command symbols that may be used include *STOP, SEND, END-OF-MSG*, and *IDLE*. The data symbols are encoded as four groups of 64 symbols. For example, from the above table reception of 011101011 would be interpreted as the data word 00101011, while reception of 110101011 would be interpreted as 10101011. The error codes are used to ensure a minimum Hamming distance of two between command symbols and data symbols. Commands interspersed within the data stream are used for buffer management and message flow control in addition to error signaling. Slack buffers are used to hide the latency of the transmission of flow control commands.

A similar physical channel structure and approach is used within the routers for Myrinet. Both Tandem's ServerNet and Myricom's Myrinet bring traditional multiprocessor network technology to the workstation and PC cluster environment. Interconnects can be substantially longer and therefore the delays for asynchronous handshaking signals become larger. The use of synchronous channels and slack buffers to overlap flow control delays as described above is motivated in part by operation in this high latency environment. Another advantage of encoding commands is the reduction in pin-out requirements. The percentage of pins devoted to data is higher although some of the bandwidth is now devoted to control.

There are also many other technological features that impact the choice of full- or half-duplex links. We point to one technique for maximizing the use of available pins on the router chips: simultaneous bidirectional signaling [48, 76, 193]. This technique allows simultaneous signaling between two routers across a single signal line. Thus, full-duplex bidirectional communication of a single bit between two routers can be realized with one pin (signal) rather than two signals. This is achieved by transmitting [76] a logic 1 (0) as a positive (negative) current. The received signal is the superposition of the two signals transmitted from both sides of the channel. Each transmitter generates a reference signal which is subtracted from the superimposed signal to generate the received signal. The result is a considerable savings over the number of I/O pins required, and consequent reduction in the packaging cost. The low voltage swing employed in the Reliable Router [76] results in additional reduction in power consumption, but necessitates different noise reduction techniques. This approach would enable wider, half-duplex links and larger number of dimensions. Currently the main difficulty with this scheme is the complexity of the signaling mechanism and the susceptibility to noise. Both of these disadvantages can be expected to be mitigated as the technology matures.

Collectively, from the preceding observations we can make the following general observations. For large systems with a higher number of dimensions, wire delays would dominate. The use of higher clock speeds, communication intensive applications and the use of pipelined links in this environment would favor full-duplex links. Lower dimensionality, and communication traffic below network saturation would tend to favor the use of half-duplex links. Cost considerations would encourage the use of low(er) cost packaging which would also favor half-duplex links and the use of command/data encodings to reduce the overall pin count and therefore package cost.

7.2.3 Wormhole Routers

A Generic Dimension-Ordered Router and its Derivatives

A generic dimension-ordered router can be represented as shown in Figure 7.13 [57]. The abstract router design in Figure 7.7 can be refined to take advantage of the structure of dimension-ordered routing. In dimension-ordered routing, messages complete traversal in a dimension prior to changing dimension. Thus, it is not necessary to provide paths from any input link to any output link. An incoming message on dimension X will either continue along dimension X (continuing in either the positive or negative direction), or proceed to the next dimension to be traversed. Thus, each dimension can be serviced by a 3×3 crossbar as shown in Figure 7.13 for a 2-D router. The first dimension accepts messages from the local node and the last dimension ejects messages to the local node. A message must cut through each dimension crossbar. For example, consider a message that is injected into the network that must only traverse the Y dimension. It is injected into the X dimension crossbar where routing logic forwards it to the Y dimension crossbar, and subsequently out along the Y dimension link.

If dimension offsets are used in the header, routing decisions are considerably simplified. A check for zero determines if the traversal within the dimension has been completed. The header

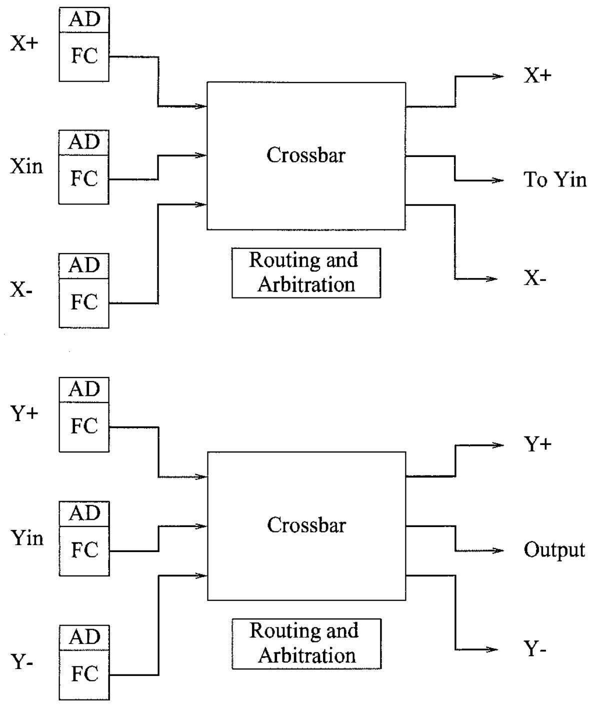


Figure 7.13. Organization of a canonical wormhole router. (AD = Address decoder; FC = Flow control.)

is updated and forwarded through the output port. An example of a router that uses such a partitioned data path is the network design frame [79]. This is a router designed for use in 2-D mesh-connected networks. Each physical channel is an 11-bit bidirectional channel supported by five control lines to implement channel flow control. Each link supports two virtual channels in each direction. Four of the control signals are bidirectional lines used for request/acknowledge flow control for each virtual channel. The fifth bidirectional control line is used to implement a token-passing protocol to control access to the bidirectional data lines. The physical channel flow control protocol is described in Example 7.1. The two virtual channels implement two virtual networks, at two priority levels. Latency-sensitive traffic uses the higher priority network. Since physical bandwidth is rarely completely utilized, the use of virtual channels to separate traffic is an attractive alternative to two distinct physical networks. Within the router, each virtual network implementation is well represented by the canonical organization shown in Figure 7.13. A 3×3 nonblocking switch (reference the multistage topologies in Chapter 1) can be implemented with four 2×2 switches. Using the partitioned design in Figure 7.13, the four crossbar switches can be implemented using twelve 2×2 switching elements, rather than having a 9×9 full crossbar switch with 11-bit channels at each port. This represents an area savings of approximately 95%.

The first two flits of the message header contain the offsets in the X and Y direction respectively. The routing units perform sign and zero checks to determine if the message is to continue in the same dimension, transition to the next dimension, or be ejected. Output link controllers perform

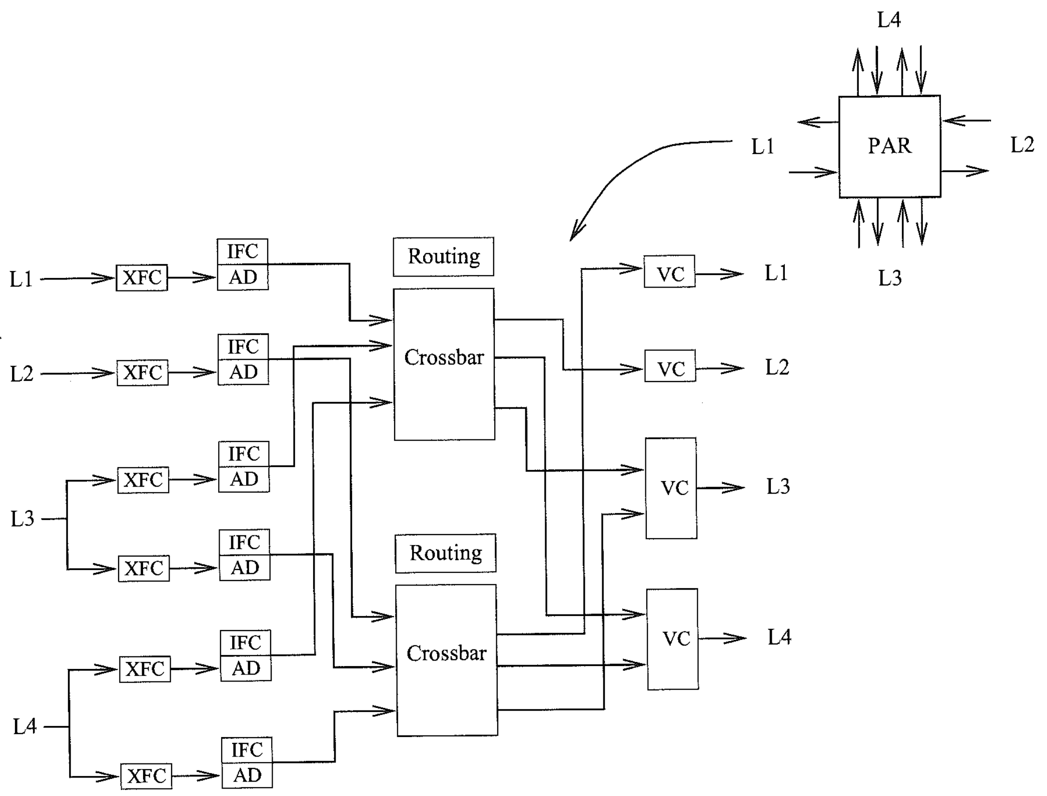


Figure 7.14. Organization of a planar-adaptive router. (AD = Address decoder; IFC = Internal flow control; PAR = Planar-adaptive router; VC = Virtual channel controller; XFC = External flow control.)

arbitration between the two virtual channels to grant access to the physical link. If the router does not have access to the bidirectional channel, there is a delay before the token is requested and transferred from the other side of the channel.

These wormhole routers exploit the routing restrictions of dimension-ordered routing to realize efficient router architectures. If adaptive routing is permitted, the internal router architectures begin to become more complex, especially as the number of dimensions and the number of virtual channels increase. This is simply a direct result of the increased routing flexibility. However, the idea of using partitioned datapaths to reduce the implementation complexity can still be applied if adaptivity can be controlled. The advantages of the partitioned datapath found in dimension-ordered routers can be combined with limited adaptive routing to achieve a more balanced, less complex design. The planar-adaptive routing techniques described in Chapter 4 is an attempt to balance hardware complexity with potential advantages of increased routing freedom. The basic strategy is to adaptively route in a sequence of adaptive planes. The block diagram of a 2-D router is shown in Figure 7.14 [11]. Note how the datapath is partitioned. Each crossbar provides the switching for the increasing or decreasing network. Messages only travel in one or the other. Higher-dimensional networks can be supported by providing the same basic organization in each successive pair of dimensions. This technique can also be generalized to support higher degrees of adaptivity by increasing the number of dimensions within which adaptive routing is permitted. Figure 7.14 illustrates the organization for adaptive routing in two dimensions at a time. Alternatives may permit adaptive routing in three dimensions at a time. A 5-D network may permit adaptive routing in successive 3-D cubes. These routers are referred to as *f-flat* routers, where the parameter f signifies the number of dimensions within which adaptive routing is permitted at any one time. Aoyama and Chien [11] use this baseline design to study the performance impact of adaptive routing via the design of f-flat routers. While performance may be expected to improve with increasing adaptivity, the hardware complexity certainly grows with increasing f.

This idea of using partitioned datapaths to reduce internal router complexity can be used in fully adaptive routers as well in order to offset the competing factors of increased adaptivity and low router delay. The internal router architecture can be designed to fully exploit the expected routing behavior as well as the capabilities of the underlying routing algorithm to achieve highest possible performance. For instance, if most packets tend not to change dimensions or virtual channel classes frequently during routing (i.e., routing locality in dimension or in virtual channel class), then it is not necessary for the internal datapath to provide packets with direct access to all output virtual channels, even in fully adaptive routing. Each crossbar switch may provide access to all the virtual channels in the same dimension. Alternatively, a given crossbar switch may provide access to a single virtual channel in each dimension. In both cases, each switch must provide a connection to the next switch.

For instance, in the Hierarchical Adaptive Router [215], the internal datapath is partitioned into ordered virtual channel networks which includes all dimensions and directions. Deadlock is avoided by enforcing routing restrictions in the lowest virtual network. Although this router architecture exploits routing locality in virtual channel class, the lowest virtual channel network (the escape resource) presents a potential bottleneck. While this ordering between virtual channel networks is required for deadlock avoidance-based routing, it is a restriction that can be relaxed in recovery-based routing. Choi and Pinkston [61] proposed such an enhancement for a Disha deadlock recovery-based router design.

The Intel Teraflops Router

The Teraflops system is an architecture being designed by Intel Corporation in an effort to produce a machine capable of a sustained performance of 10^{12} floating-point operations per second and a peak performance of 1.8×10^{12} floating-point operations per second. *Cavallino* is the name for the network interface chip and router that forms the communication fabric for this machine [48]. The network is a k-ary 3-cube, with a distinct radix in each dimension. Up to eight hops are permitted in the Z dimension, 256 hops in the X dimension and 64 hops in the Y dimension. The router has six ports and the interface design supports 2-D configurations where the two ports in the Z dimension are connected to two processing nodes rather than other routers. This architecture provides a flexible basis for constructing a variety of topologies.

Unlike most other routers, Cavallino uses simultaneous bidirectional signaling. Physical channels are half-duplex. A receiver interprets incoming data with respect to the reference levels received and the data being driven to determine the value being received. The physical channel is 16 bits wide (phits). Message flits are 64 bits and require four clock cycles to traverse the channel. Three additional signals across the channel provide virtual channel and flow control information. The buffer status transmitted back to the sending virtual channel is interpreted as almost full rather than an exact number of available locations. This scheme allows for some slack in the protocols to

;. This
ımber
ization
ı three
These
ensions
se this
outers.
plexity

ısed in
ınd low
routing
ıossible
classes
en it is
virtual
virtual
ss to a
tion to

ed into
ıvoided
tecture
escape
vorks is
r-based
r-based

oduce a
a peak
ıetwork
ıetwork
ı the Z
x ports
ion are
flexible

ıannels
eceived
16 bits
Three
ı. The
rather
ocols to

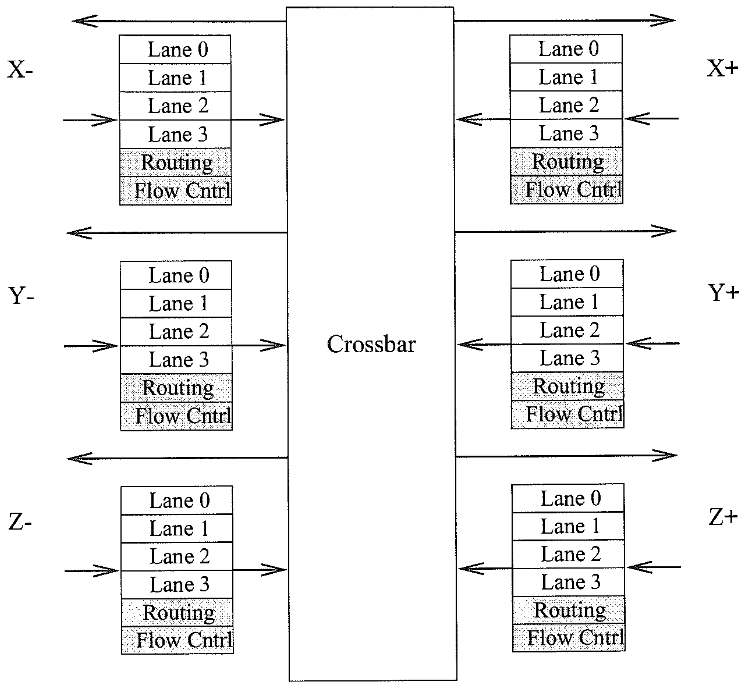


Figure 7.15. Architecture of the Intel teraflops router.

tolerate signaling errors. The physical channel operates at 200 MHz, providing an aggregate data rate of 400 Mbytes/s in each direction across the link. Four virtual channels are multiplexed in each direction across a physical link.

The internal structure of the router is shown in Figure 7.15. A message enters along one virtual channel and is destined for one output virtual channel. The routing header contains offsets in each dimension. In fact, routing order is $Z-X-Y-Z$. Using offsets permits fast routing decisions within each node as a message either turns to a new dimension or continues along the same dimension. The router is input buffered. With six input links and four virtual channels/link, a 24-to-1 arbitration is required on each crossbar output. The speed of operation and the length of the signal paths led to the adoption of a two-stage arbitration scheme. The first stage arbitrates between corresponding virtual channels from the six inputs, e.g., a 6-to-1 arbiter for virtual channel 0 on each link. The second stage is a 4-to-1 arbitration for all channels with valid data competing for the same physical output link. The internal data paths are 16 bits, matched to the output links. The ports were designed such that they could be reset while the network was operational. This would enable removal and replacement of routers and CPU boards while the network was active.

A block diagram of the network interface is shown in Figure 7.16. The router side of the interface utilizes portions of the router architecture configured as a three-port design. One port is used for message injection and ejection while two other ports are used for connection to two routers. This flexibility permits the construction of more complex topologies. This is often desired for the purposes of fault tolerance or simply increased bandwidth. Each output virtual channel is 384 bytes deep while

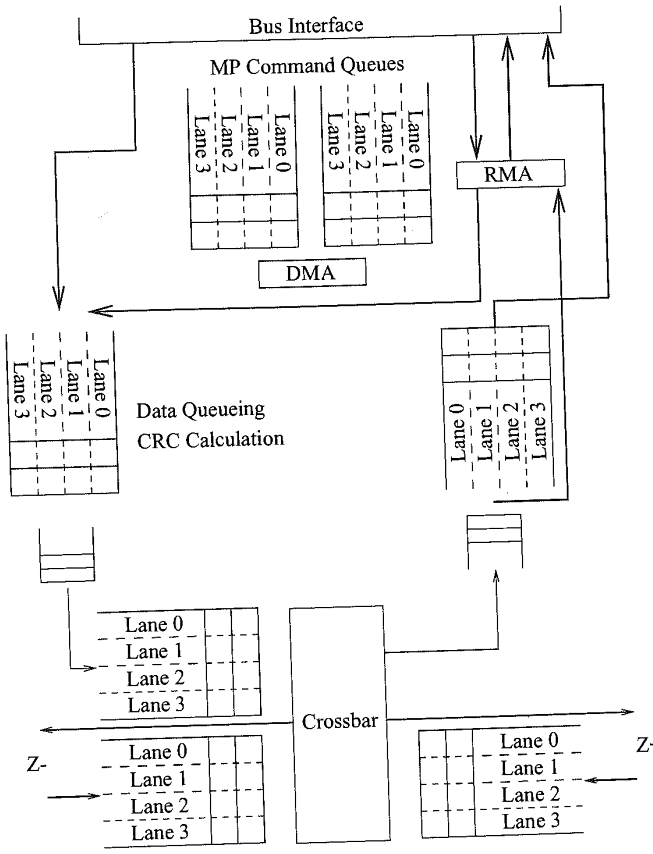


Figure 7.16. Interface architecture for the Cavallino router. (CRC = Cyclic redundancy check.)

each input virtual channel is 256 bytes deep. There are two additional interesting components of the interface. The first is the remote memory access (RMA) engine. The RMA engine has access to a memory-resident mapping table that maps processor addresses into the physical addresses of remote nodes. Request messages are created for remote accesses and response messages are created to service remote requests. The RMA engine also provides support for remotely accessible atomic operations, e.g., read and clear. This functionality supports synchronization operations. The second interesting feature is the direct memory access (DMA) interface. The DMA engine supports eight channels and corresponding command queues (one for each input and output virtual channel). Each channel can store up to eight commands. This volume of posted work is supported by command switching by the DMA engine, e.g., when injection is blocked due to congestion. The access to the control of the DMA engine is memory-mapped, and user access is permitted to support the implementation of low-overhead, lightweight messaging layers. Throughput through the DMA engine approaches 400 Mbytes/s.

While forming the backbone of the Teraflops machine, the Cavallino chip set is also regarded as a high-performance chip set that can be used for configuring high-performance NOWs.

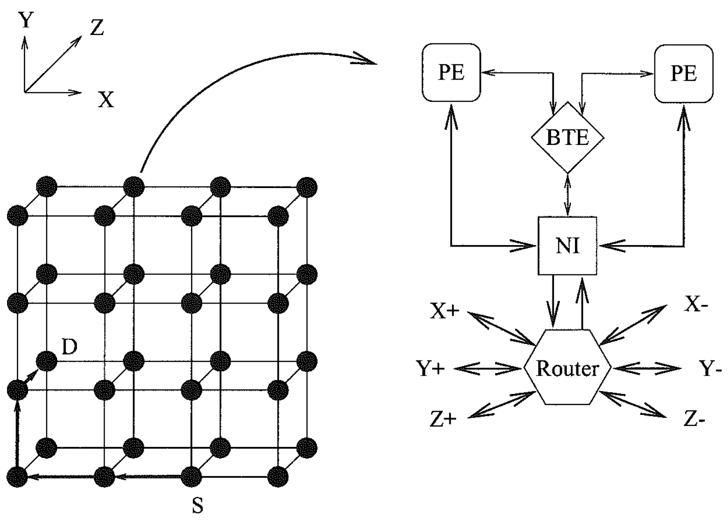


Figure 7.17. Organization of the T3D system. (BTE = Block transfer engine; NI = Network interface.)

The Cray T3D

The Cray T3D utilizes a *k*-ary 3-cube interconnect. An example of a $4 \times 4 \times 2$ configuration of processing element (PE) nodes is is illustrated in Figure 7.17. Each PE node consists of two DEC Alpha 150 MHz processors sharing a single network interface to the local router. The network has a maximum radix of 8 in the X and Z dimensions and 16 in the Y dimension. Therefore, the maximum configuration is 1,024 PE nodes or 2,048 processors. The system is packaged with two PE nodes per board. This choice of radix is not arbitrary. Rather it is based on the observation that under random traffic, messages will travel $\frac{1}{4}$ the way around each dimension in the presence of bidirectional links. If we consider only the packets that travel in say, the $X+$ direction, then the average distance traveled by these packets will be $\frac{1}{8}$ the distance in that direction. If the injection and ejection ports of the network interface have the same capacity as the individual links then a radix of 8 in each dimension can be expected to produce a balanced demand on the links and ports.

The router architecture is based on fixed-path, dimension-order wormhole switching. The organization of the relevant components is shown in Figure 7.18. There are four unidirectional virtual channels in each direction over a physical link. These four virtual channels are partitioned into two virtual networks with two virtual channels/link. One network is referred to as the request network and transmits *request packets.* The second network is the reply network carrying only *reply packets.* The two virtual channels within each network prevent deadlock due to the wraparound channels as described below. Adjacent routers communicate over a bidirectional, full-duplex physical channel, with each direction comprised of 16 data bits and 8 control bits as shown in Figure 7.18. The 4 forward control channel bits are used to identify the type of message packet (request or response) and the destination virtual channel buffer. The four acknowledge control signals are used for flow control and signify empty buffers on the receiving router. Due to the time required for channel flow control (e.g., generation of acknowledgments), as long as messages are integral multiples of 16 bits (see message formats below), full link utilization can be achieved. Otherwise some idle physical

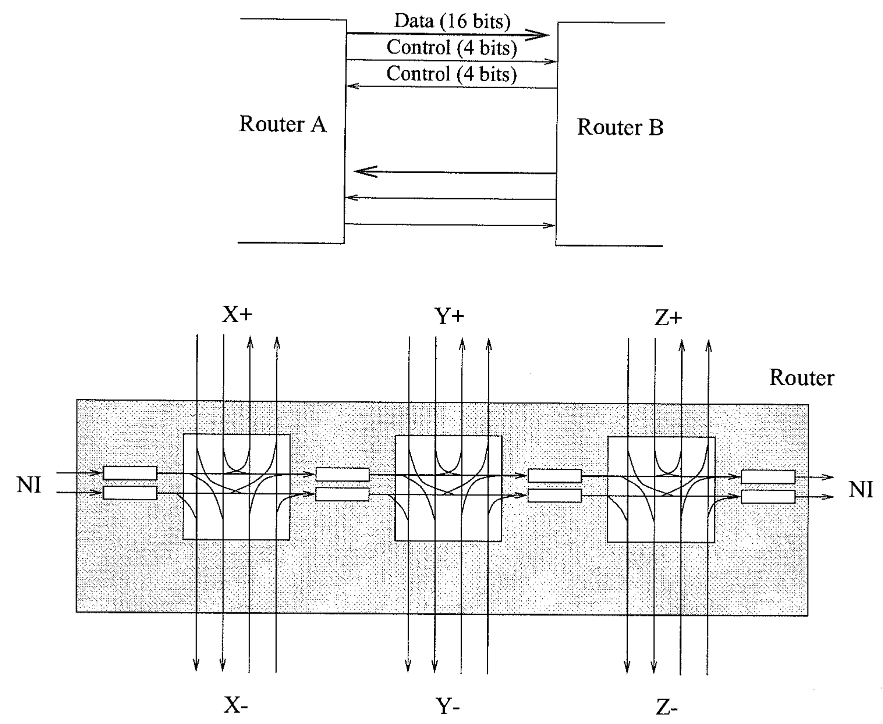


Figure 7.18. Architecture and operation of the T3D router.

channel cycles are possible. The router is physically partitioned into three switches as shown in the figure. Messages are routed to first correct the offset in X, then Y, and finally Z dimensions. The first switch is connected to the injection channel from the network interface and is used to forward the message in the $X+$ or $X-$ directions. When the message has completed traversal in the X dimension, the message moves to the next switch in the intermediate router for forwarding along the Y dimension, and then to the Z dimension. Message transition between switches is via *interdimensional virtual channels.* For the sake of clarity the figure omits some of the virtual channels.

Message packets comprise of a header and body. Sample formats are shown in Figure 7.19. The header is an integral number of 16-bit phits that contain routing information, control information for the remote PE, and possibly source information as well. The body makes use of check bits to detect errors. For example, a 64-bit word will be augmented with 14 check bits (hence the 5-phit body in Figure 7.19) and four-word message bodies will have 56 check bits (necessitating 20-phit bodies). The messages are organized as sequences of flits, with each flit comprised of 8 phits. Virtual channel buffers are 1 flit deep.

The T3D utilizes *source routing.* Header information identifies the sequence of virtual channels that the message will traverse in each dimension. In order to understand the routing strategy, we must first be familiar with the PE addressing scheme. While all PEs have physical addresses, they also possess logical addresses and virtual addresses. Logical addresses are based on the logical

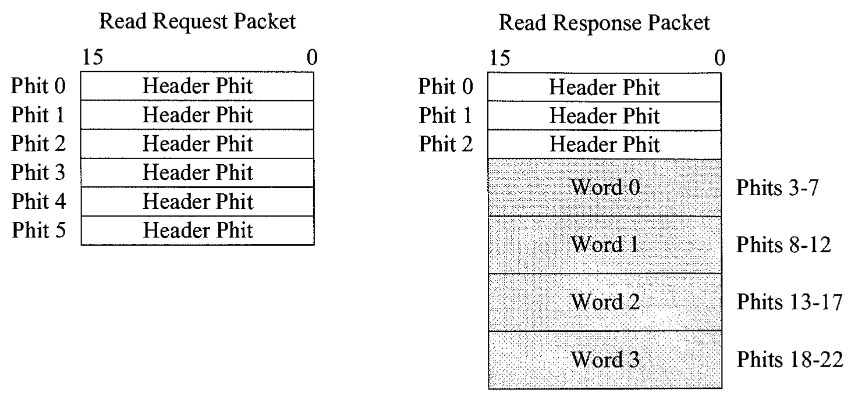


Figure 7.19. T3D packet formats.

topology of the installation. Thus, nodes can be addressed by the coordinate of their logical position in this topology, and are in principle independent of their physical location. This approach permits spare nodes to be mapped into the network to replace failed nodes by assigning the spare node the same logical address. The virtual address is assigned to nodes within a partition allocated to a job. The virtual address is interpreted according to the shape of the allocated partition. For example, 16 nodes can be allocated as an $8 \times 2 \times 1$ topology, or as a $4 \times 2 \times 2$ topology. In the latter case the first 2 bits of the virtual address are allocated to the X offset and the remaining 2 bits to the Y and Z offsets. When a message is transmitted, the operating system or hardware translates the virtual PE address to a logical PE address. This logical PE address is used as an index into a table that provides the offsets in each dimension to forward the message to the correct node. In this manner if a spare PE is mapped in to replace a faulty PE, the routing tables at each node simply need to be updated.

Dimension-order routing prevents cyclic dependencies between dimensions. Deadlock within a dimension is avoided by preventing cyclic channel dependencies as shown in Figure 7.20. One channel is identified by the software as the *dateline* communication link. Consider a request message that

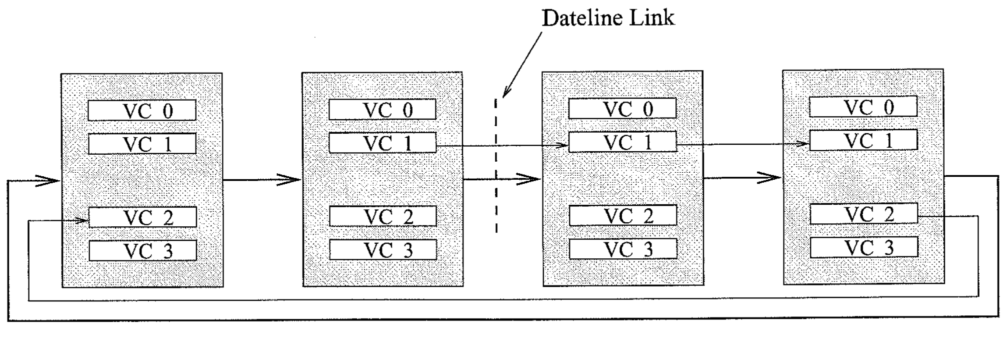


Figure 7.20. Deadlock avoidance within a dimension. (VC = Virtual channel.)

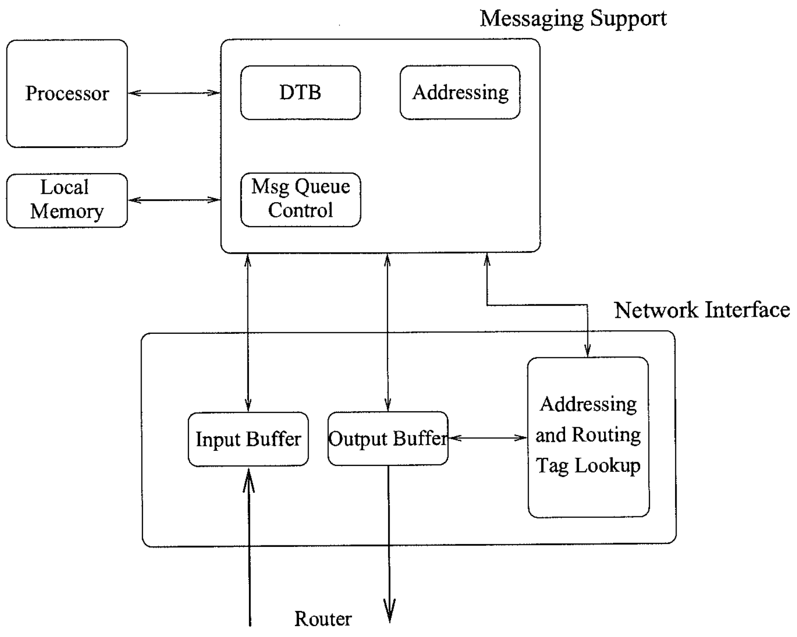


Figure 7.21. Message processing datapath. (DTB = Data translation buffer.)

must traverse this dimension. If the message will traverse the dateline communication link, it will be forwarded along virtual channel 1, otherwise it will be forwarded along virtual channel 0. The message does not switch between virtual channels within a dimension. The dateline effectively identifies the wraparound channel, and the routing restrictions implement the Dally and Seitz [78] restrictions to guarantee deadlock-free routing. When the routing tables are constructed, they implement these restrictions. Such restrictions, while realizing deadlock-free routing, result in unbalanced utilization of the virtual channels. The flexibility of source-based routing enables static optimization of the message paths to balance traffic across virtual channels and further improve link utilization [311].

The T3D provides architectural support for messaging. The routing tables are supported in hardware, as is virtual to logical PE address translation prior to routing tag lookup. Message headers are constructed after the processor provides the data and address information. When a message is received, it is placed in a message queue in a reserved portion of memory and an interrupt is generated. If the store operation to the message queue fails, a negative acknowledgment is returned to the source where messages are buffered to enable retransmission. Successful insertion into the queue generates a positive acknowledgment to the source. Request and reply packets can implement a shared address space (not coherent shared memory). When the processor generates a memory address, support circuitry maps that address into a local or nonlocal address. Nonlocal references result in request packets being generated to remote nodes. A block diagram illustrating the organization of the support for message processing within the node is shown in Figure 7.21.

The T3D links operate at 150 MHz or 300 Mbytes/s. The reported per hop latency is two clock cycles [311] while the maximum sustained data bandwidth into the node (i.e., excluding packet headers, acknowledgments, etc) is 150 Mbytes/s per node or 75 Mbytes/s per processor.

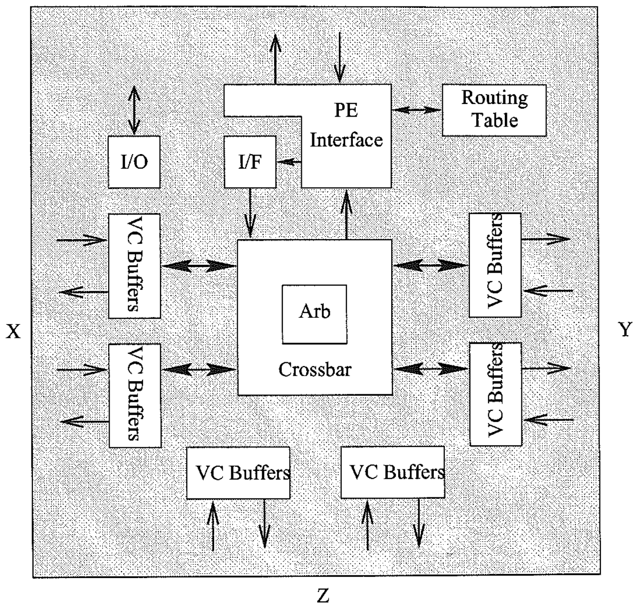


Figure 7.22. Architecture of the T3E router. (Arb = Arbitrator; I/F = Interface.)

The Cray T3E

The T3E represents a second-generation multiprocessor system [309, 312]. The network and router architecture of the T3E retains many of the operational characteristics of the T3D but is very different in other ways. The system design evolved to substantially improve latency tolerance, resulting in a shift in emphasis in the design of the network from reducing latency to providing sustained bandwidth. This resulted a change in the balance between processors and network bandwidth — only one processor/network node in the T3E — and the use of standard-cell CMOS (single-chip) rather than ECL (three-chip) technology. Along with doubling of the link bandwidth, this produces a factor of four increase in the per processor bandwidth. The topology is still that of a 3-D torus with a maximum radix of 8, 32, and 8 in the X, Y, and Z dimensions, respectively. A block diagram of the T3E router is shown in Figure 7.22.

A T3E packet is 1 – 10 flits, and each flit is 5 phits. The network physical channel is full-duplex with 14-bit data producing a flit size of 70 bits. A flit can carry one 64-bit word with some additional control information. The router operates on a 75 MHz clock. During each clock cycle 5 phits are transmitted across the channel, leading to a link transmission rate of 375MHz and link bandwidth of 600 Mbytes/s.

Five virtual channels are multiplexed in each direction over each physical channel. Four of the channels essentially function as they do in the T3D (with differences identified below) while the fifth channel is used for fully adaptive routing. Each adaptive channel can buffer up to 22 flits while each of the regular channels can buffer up to 12 flits. The channels utilize credit based flow control for buffer management with acknowledgments in the reverse direction being piggybacked on other messages or transmitted as idle flits. Round-robin scheduling across active virtual channels within a group determines which channel will compete for crossbar outputs. The winning channel can request a virtual channel in the deterministic direction, or an adaptive channel in the highest-ordered direction yet to be satisfied. If both are available, the adaptive channel is used. Once an output virtual channel is allocated, it must remain allocated, while other virtual channel inputs may still request and use the same physical output channel. Finally, output virtual channels must arbitrate for the physical channel. This process is also round robin although the adaptive channel has the lowest priority to start transmitting.

Routing is fully adaptive. The deterministic paths use four virtual channels for the request/reply network as in the T3D. However, the path is determined by ordering dimensions *and* the direction of traversal within each dimension. This style of routing can be regarded as *direction order* rather than dimension order. The ordering used in the T3E is $X+$, $Y+$, $Z+$, $X-$, $Y-$, and $Z-$. In general, any other ordering would work just as well. By applying the concept of channel classes from Chapter 3, it is apparent that each direction corresponds to a class of channels, and the order in which channel classes are used by a message is acyclic. Thus, routing is deadlock-free. In addition, the network supports an initial hop in any positive direction, and permits one final hop in the $Z-$ direction at the end of the route. These initial and final hops add routing flexibility and are only necessary if a normal direction-order route does not exist. Each message has a fixed path that is determined at the source router. A message can be routed along the statically configured path, or along the adaptive virtual channel over any physical link along a minimal path to the destination. However, the implementation does not require the extended channel dependency graph to be acyclic. The adaptive virtual channels are large enough to buffer two maximal messages. Therefore indirect dependencies between deterministic channels do not exist. This permits optimizations in the manner in which virtual channels are assigned to improve virtual channel utilization and minimize imbalance due to routing restrictions. Finally, adaptive routing can be turned off via control fields within the routing tables, and on an individual packet basis through a single bit recorded in the message. Since adaptively routed messages may arrive out of order at the destination, the latter capability enables the processor to issue sequences of messages that are delivered in order.

Memory management and the network interface operation have been made more flexible than the first-generation support found in the T3D. An arbitrary number of message queues are supported in both user memory and system memory. Message queues can be set up to be interrupt-driven, polled, or interrupt-driven based on some threshold on the number of messages. In general, message passing is more tightly coupled with operations for the support of shared-memory abstractions.

The Reliable Router

The Reliable Router chip is targeted for fault-tolerant operation in 2-D mesh topologies [76]. The block diagram of the Reliable Router is shown in Figure 7.23. There are are six input channels corresponding to the four physical directions in the 2-D mesh, and two additional physical ports: the local processor interface, and a separate port for diagnostics. The input and output channels are connected through a full crossbar switch, although some input/output connections may be prohibited by the routing function.

While message packets can be of arbitrary length, the flit length is 64 bits. There are four flit types: head, data, tail, and token. The format of the head flit is as shown in Figure 7.24. The size of the physical channel or phit size is 23 bits. The channel structure is illustrated in Figure 7.23. To permit the use of chip carriers with fewer than 300 pins, these physical channels utilize half-duplex channels with simultaneous bidirectional signaling. Flits are transferred in one direction across the physical channel as four 23-bit phits called *frames*, producing 92-bit transfers. The format of a data flit and its constituent frames are as shown in Figure 7.25. The 28 bits in excess of the flit size are

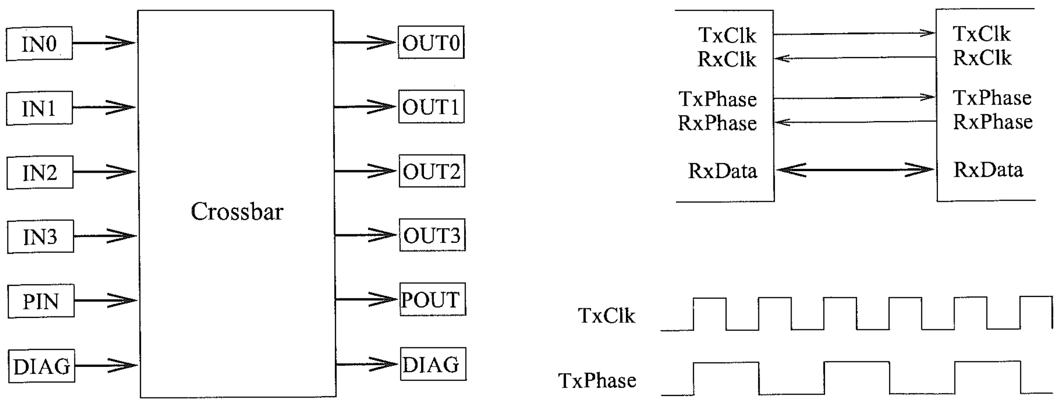


Figure 7.23. Block diagram of the Reliable Router.

used for byte parity bits (BP), kind of flit (Kind), virtual channel identification (VCI), flow control to implement the unique token protocol (Copied Kind, Copied VCI, Freed), to communicate link status information (U/D, PE), and two user bits (USR1, USR0). For a given direction of transfer, the clock is transmitted along with the four data frames as illustrated in the figure. Data are driven on both edges of the clock. To enable the receiver to distinguish between the four frames and reassemble them into a flit, the transmitting side of the channel also sends a pulse on the *TxPhase* signal which has the relative timing as shown. The flit is then assembled and presented to the routing logic. This reassembly process takes two cycles. Each router runs off a locally generated 100 MHz clock removing the problems with distributing a single global clock. Reassembled flits pass through a synchronization module for transferring flits from the transmit clock domain to the receive clock domain with a worst-case penalty of one cycle (see [76] for a detailed description of the data synchronization protocol). The aggregate physical bandwidth in one direction across the channel is 3.2 Gbits/s.

Bit Field	63:12	11	10	9:5	4:0
Contents	User info	Priority	Diagnostic	Address in Y	Address in X

Figure 7.24. Reliable Router head flit format.

Bit Field	22	21	20:18	17	16	15:0
Frame 0	PE	USR0	VCI	BP1	BP0	Data[15:0]
Frame 1	Copied kind		Copied VCI	BP3	BP2	Data[31:16]
Frame 2	U/D	USR1	Kind	BP5	BP4	Data[47:32]
Frame 3	Freed			BP7	BP6	Data[63:48]

Figure 7.25. Reliable Router frame format for data flits.

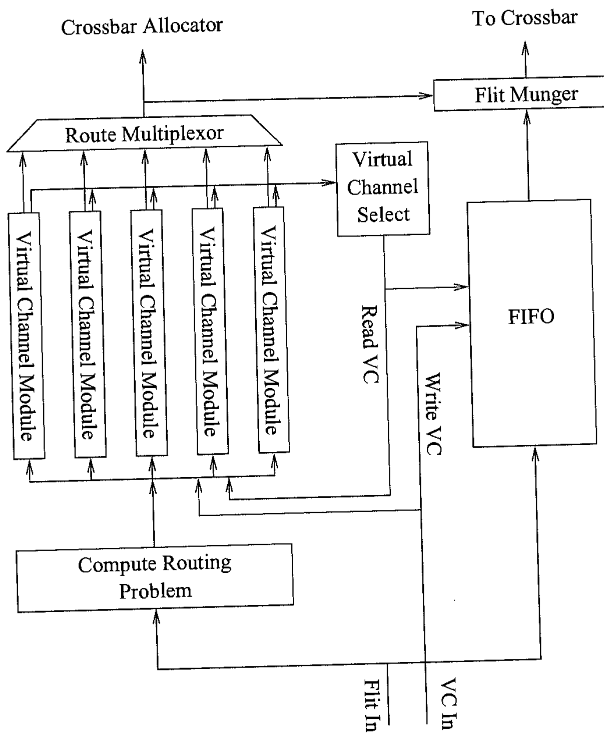


Figure 7.26. Block diagram of the input controller.

While the output controllers simply transmit the flit across the channel, generating the appropriate control and error checking signals, the input controllers contain the core functionality of the chip, and are organized as illustrated in Figure 7.26. The crossbar provides switching between the physical input channels and the physical output channels. Each physical channel supports five virtual channels, which share a single crossbar port: two channels for fully adaptive routing, two channels for dimension-ordered routing, and one for fault handling. The fault handling channels utilize turn model routing. The two dimension-ordered channels support two priority levels for user packets. The router is input buffered. The FIFO block is actually partitioned into five distinct FIFO buffers — one corresponding to each virtual channel. Each FIFO buffer is 16 flits deep. Bandwidth allocation is demand driven: only allocated channels with data to be transmitted or channels with new requests (i.e., head flits) compete for bandwidth. Virtual channels with message packets are accorded access to the crossbar bandwidth in a round robin fashion by the scheduler in the Virtual Channel Controller. Data flits simply flow through the virtual channel FIFO buffers, and the virtual channel number is appended to the flit prior to the transmission of the flit through the crossbar and across the physical channel to the next router. Flow control guarantees the presence of buffer space. When buffers at the adjacent node are full, the corresponding virtual channel does not compete for crossbar bandwidth.

A more involved sequence of operations takes place in routing a head flit. On arrival the head flit is transferred to both the FIFO buffer and a block that compares the destination address with the local node address. The result is the *routing problem* which contains all of the information used to route the message. This information is stored in the virtual channel module, along with routing information: the address of the output controller and the virtual channel ID. Recall from the protocol description provided in Section 6.7.2, the header information must be retained in the router to construct additional messages in the presence of a fault. To minimize the time spent in arbitration for various shared resources, the routing logic is replicated within each virtual channel module. This eliminates the serialized access to the routing logic and only leaves arbitration for the output controllers which is handled by the crossbar allocator. The routing logic first attempts to route a message along an adaptive channel, failing which a dimension-order channel is requested. If the packet fails in arbitration, on the next flit cycle, the router again first attempts an adaptive channel. The virtual channel module uses counters and status signals from adjacent nodes (see *Freed* bits in Figure 7.25) to keep track of buffer availability in adjacent routers. A virtual channel module is eligible to bid for access to the crossbar output only if the channel is routed, buffer space is available on the adjacent node, and there is data to be transmitted. When a fault occurs during transmission, the channel switches to a state where computation for retransmission occurs. After a period of two cycles, the channel switches back to a regular state and competes for access to the crossbar outputs. A header can be routed and the crossbar allocated in two cycles, i.e., 20 ns. The worst case latency through the router is projected to be eight cycles or 80 ns.

Arbitration is limited to the output of the crossbar, and is resolved via three packet priority levels: two user priority levels, and the highest level reserved for system packets. Starvation is prevented by changing the packet priority to level 3 after an input controller has failed to gain access to an output controller after seven tries. Packet selection within a priority level is randomized.

The implementation of the unique token protocol for reliable transmission necessitates some special handling and architectural support. The token at the end of a message must be forwarded only after the corresponding flit queue has successfully emptied. Retransmission on failure involves generation of duplicate tokens. Finally, flow control must span two routers to ensure that a duplicate copy of each data flit is maintained at all times in adjacent routers. When a data flit is successfully transmitted across a channel, the copy of the data flit two routers upstream must be deallocated. The relevant flow control signals are passed through frames as captured in Figure 7.25.

SGI SPIDER

The SGI SPIDER (Scalable Pipelined Interconnect for Distributed Endpoint Routing) [118] was designed with multiple missions in mind: as part of a conventional multiprocessor switch fabric, as a building block for large-scale nonblocking central switches, and as a high-bandwidth communication fabric for distributed graphics applications. The physical links are full-duplex channels with 20 data bits, a frame signal, and a differential clock signal in each direction. Data are transferred on both edges of a 200 MHz clock realizing a raw data rate of 1 Gbyte/s in each direction. The chip core operates at 100 MHz, providing 80-bit units that are serialized into 20-bit phits for transmission over the channel. The organization of the SPIDER chip is shown in Figure 7.27.

The router implements four 256-byte virtual channels over each physical link. The units of buffer management and transmission are referred to as *micropackets*, equivalent to the notion of a flit. Each micropacket is comprised of 128 bits of data, 8 bits of sequencing information, 8 bits of virtual channel flow control information, and 16 CRC bits to protect against errors. The links implement automatic retransmission on errors until the sequence number of the transmitted flit is acknowledged. The micropacket format is shown in Figure 7.28. The number on top of each field indicates the field size in bits. Virtual channel buffer management utilizes credit-based flow control. On initialization, all channels credit their counterparts with buffer space corresponding to their size.

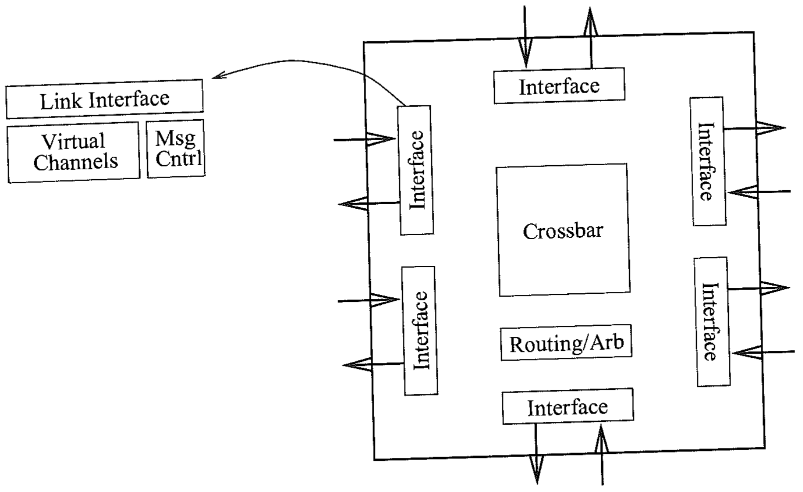


Figure 7.27. Organization of the SGI SPIDER chip.

The 8 bits of virtual channel flow control information are used to identify the virtual channel number of the transmitted micropacket, the address of the virtual channel being credited with additional buffer space, and the amount of credit. An interesting feature of SPIDER links is the transparent negotiation across the link to determine the width of the channel. For example, the router can negotiate to use only the lower 10 bits of the channel. The interface to the chip core remains the same, while the data rate will be a function of the negotiated port width.

The SPIDER chip supports two styles of routing. The first is source routing which is referred to as *vector routing*. This mechanism is utilized for network configuration information and administration. The second style is table-driven routing. The organization of the tables is based on a structured view of the network as a set of metadomains, and local networks within each domain. The topology of the interdomain network and the local network can be different. Messages are first routed to the correct destination domain and then to the correct node within a domain. Routing is organized as shown in Figure 7.29. The destination address is a 9-bit field, comprised of two subfields: a 5-bit metadomain field and a 4-bit local field. The tables map these indices into 4-bit router port addresses, thereby supporting routers with up to 16 ports. Access to the metatable provides a port address to route the message toward the correct domain. If the message is already in the correct domain as determined by comparison with the local meta ID, the local address table is used to determine the output port that will take the message towards the destination node. Not shown in the figure is a mode to force

8	4	4	64
Flow control	Seq #	Ack Seq #	Data
Check bits			Data

Figure 7.28. SPIDER data micropacket format.

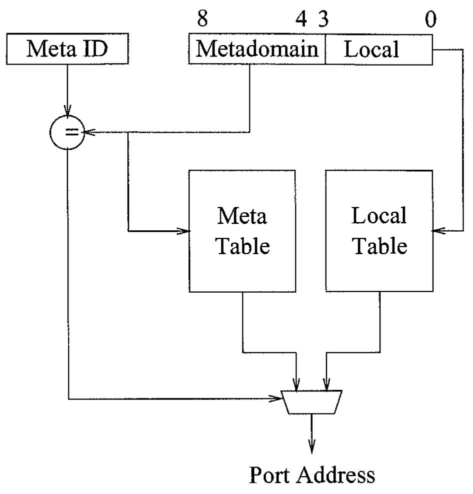


Figure 7.29. SPIDER routing tables.

the message to use the local address. Table-driven routing is oblivious and forces messages to use fixed paths. The table entries are computed to avoid deadlock and can be reinitialized in response to faults. Messages using vector routing can be used to reload the tables.

Normally arbitration for the output of the crossbar is not possible until the routing operation has been completed and has returned a crossbar output port address. In order to overlap arbitration and table lookup, the routing operation described above produces the output port address at the next router. This 4-bit port address (Dir) is passed along with the message. At the adjacent router, the port address is extracted from the header and can be immediately submitted for crossbar output arbitration while table lookup progresses in parallel. This overlap saves one cycle. The header is encoded into the 128 data bits of a micropacket. The overall header organization is shown in Figure 7.30. The number on top of each field indicates the field size in bits. The age field establishes message priorities for arbitration (see below) and the CC (congestion control) field permits the use of multiple virtual channels.

To maximize crossbar bandwidth each input virtual channel buffer is organized as a set of linked lists of messages, one for each output port (five in this chip). This organization prevents messages blocked on an output port from blocking other messages in the same buffer destined for a different output port. With 24 input virtual channels, there are potentially 120 candidates for arbitration. To avoid starvation, messages in the router are aged at a programmable rate and arbitration is priority-driven. The top 16 priority levels of a total of 255 levels are reserved.

41	9	4	8	2
Data	Dest ID	Dir	Age	CC
		Data		

Figure 7.30. SPIDER header micropacket format.

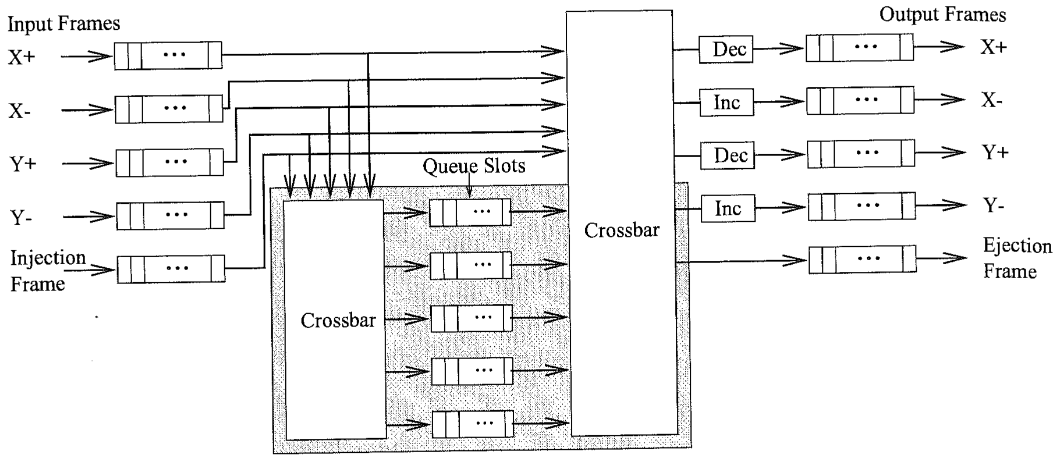


Figure 7.31. Block diagram of the Chaos router. (Dec = Decrementer; Inc = Incrementer.)

Support is provided within the chip to track message statistics such as retransmission rates on links so that problematic links can be identified and may be shut down. Timers support the detection of error conditions and potential deadlock situations, e.g., those due to errors. Timer expiration may result in either micropackets injected into the message stream for message recovery, or in resetting of a router port. A real-time clock is also distributed through the network to support tight synchronization. The pad-to-pad no-load latency through the chip is is estimated to be 40 ns, while the latency from input on a router to the input on the adjacent router is estimated to be 50 ns.

7.2.4 VCT Routers

Routers supporting VCT switching share many of the attributes of wormhole routers. The principal distinguishing feature is the availability of sufficient space for buffering complete message packets. As a result, deadlock freedom is generally achieved through buffer management rather than routing restrictions. The following examples discuss three router architectures currently under development.

The Chaos Router

The Chaos router chip is an example of a router designed for VCT switching for operation in a 2-D mesh. To reduce pin count, each channel is a 16-bit, half-duplex bidirectional channel. The chip was designed for a cycle time of 15 ns. The latency for the common case of cut-through routing (no misrouting) is four cycles from input to output. This performance compares favorably to that found in more compact oblivious routers, and has been achieved by keeping the misrouting and buffering logic off of the critical path. A block diagram of the Chaos router is shown in Figure 7.31.

The router is comprised of the router core and the multiqueue [32]. The core connects *input frames* to *output frames* through a crossbar switch. Each frame can store one 20-flit message packet. Each flit is 16 bits wide and corresponds to the width of the physical channel. The architecture of the core is very similar to that of contemporary oblivious routers, with the additional ability to buffer complete 20-flit packets in each input and output frame. Under low load conditions, the majority of the traffic flows through this core, and therefore considerable attention was paid to the optimization of the performance of the core datapath and control flow. However, when the network starts to become congested, messages may be moved from input frames to the multiqueue, and subsequently from the multiqueue to an output frame. When a packet is stalled in an input frame waiting for an output frame to become free, this packet is moved to the multiqueue to free channel resources to be used by other messages. To prevent deadlock, packets may also be moved from input frames to the multiqueue if the output frame on the same channel has a packet to be transmitted. This ensures that both receiving and transmitting buffers on both sides of a channel are not occupied, ensuring progress (reference the discussion of chaotic routing in Chapter 6) and avoiding deadlocked configurations of messages. Thus, messages may be transmitted to output frames either from an input frame or from the multiqueue, with the latter having priority. If the multiqueue is full and a message must be inserted, a randomly chosen message is misrouted.

Although several messages may be arriving simultaneously, the routing decisions are serialized by traversing the output channels in order. The *action dimension* is the current dimension under consideration (both positive and negative directions are distinguished). Packets in the multiqueue have priority in being routed to the output frame. If a packet in the multiqueue requires the active dimension output and the output frame is empty, the packet is moved to the output frame. If the queue is full and the output frame is empty, a randomly selected packet is misrouted to this output frame. Misrouting is the longest operation and takes five cycles. If the multiqueue operations do not generate movement, then any packet in an input frame requesting this output dimension can be switched through the crossbar. This operation takes a single cycle. The routing delay experienced by a header is two cycles. Finally, if the input frame of the active dimension is full, it is read into the multiqueue. This implements the policy of handling stalled packets and the deadlock avoidance protocol. The preceding steps encompass the process of routing the message. The no-load latency through the router is four cycles: one cycle for the header to arrive on the input, two cycles for the routing decision, and one cycle to be switched through the crossbar to the output frame [32]. If a message is misrouted, the routing decision takes five cycles.

Logically, the multiqueue contains messages partially ordered according to the output channel that the messages are waiting to traverse. The above routing procedure requires complete knowledge of the contents of the multiqueue and relative age of the messages. The current implementation realizes a five-packet multiqueue. This logically corresponds to five queues: one for each output including the output to the processor. Since messages are adaptively routed, a message may wait on one of several output channels, and therefore must logically be entered in more than one queue. When a free output frame is available for transfers from the multiqueue, determination of the oldest message waiting on this output must be readily available. The state of the multiqueue is maintained in a destination scoreboard.

The logical structure of the destination scoreboard is illustrated in Figure 7.32 for a multiqueue with six slots. This scoreboard keeps track of the relationship between the messages in the multiqueue, their FIFO order, and the destination outputs. Each row corresponds to an output frame in a particular dimension or to the local processor. Each column corresponds to one of the five slots in the multiqueue. When a packet enters the tail of the queue, its status enters from the left, i.e., in the first column. In this column, each row entry is set to signify that the corresponding output channel is a candidate output channel for this packet. For example, the packet that is located at the tail of the queue can be transmitted out along the $Y+$ or $X-$ directions, whichever becomes available. As packets exit the multiqueue, the columns in the scoreboard are shifted right. When an output frame is freed, the first message destined for that frame is dequeued from the central queue. Thus, messages may not necessarily be removed from the head of the queue. In this case some column in the middle of the matrix is reset, and all columns are compressed right corresponding to

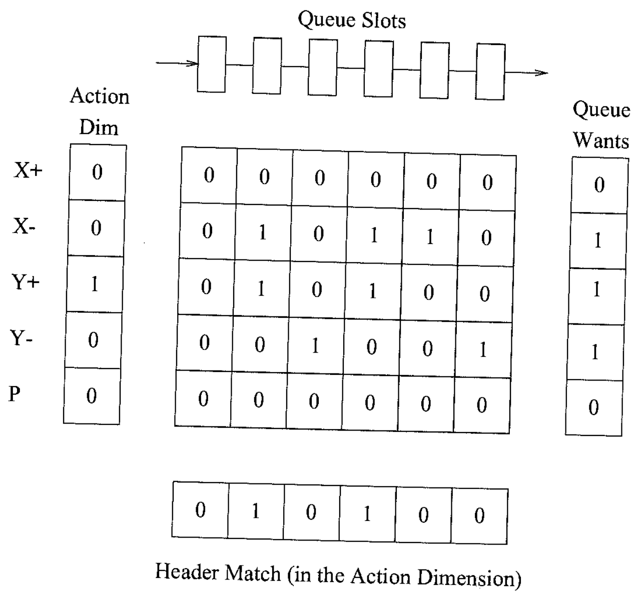


Figure 7.32. Tracking messages in the Chaos multiqueue.

the compression in the queue when a message is removed from the middle. It is apparent that taking the logical OR of row elements indicates if there is a message in the queue for the corresponding output frame. Given a specific dimension (e.g., the action dimension) it is apparent that row and column logical operations will produce a vector indicating messages that are queued for the action dimension. The relative positions of the bits set in this vector determines the relative age of the multiple messages and is used to select the oldest. Thus, all messages are queued in FIFO order, while messages destined for a particular output are considered in the order that they appear in this FIFO buffer.

The maximum routing delay for a message packet is five cycles, and occurs when messages are misrouted. This delay can be experienced for each of the four outputs in a 2-D mesh (the processor channel is excluded). Since routing is serialized by output, each output will be visited at least once every 20 cycles. Thus, messages are 20 flits to maintain maximum throughput. In reality, the bidirectional channels add another factor of 2, therefore in the worst case we have 40 cycles between routing operations on a channel.

Finally, the router includes a number of features in support of fault-tolerant routing. Timers on the channels are used to identify adjacent faulty channels/nodes. Packets bodies are protected by checksums while the header portion is protected by parity checks. A header that fails a parity check at an intermediate node, is ejected and handled in software. A separate single bit signal on each channel is used to propagate a fault detection signal throughout the network. When a router receives this signal, message injections are halted and the network is allowed to "drain" [35]. Messages which do not successfully drain from the network are detected using a timer and ejected to the local processor. When all messages have been removed from the faulty network, the system enters a diagnostic and recovery phase. Recovery is realized by masking out faulty links. This is achieved within the routing decision logic by performing the logical AND of a functional channel

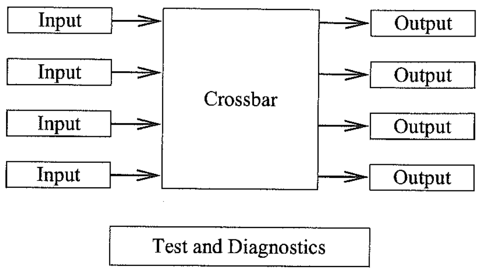


Figure 7.33. The Arctic routing chip.

mask with a vector of candidate output channels computed from the routing header. This will only permit consideration of nonfaulty output channels. If the result indicates that no candidate channels exist (this may occur due to faults), the packet is immediately misrouted.

Currently an active project is underway to implement a version of the Chaos router as a LAN switch. This switch will be constructed using a new version of the Chaos router chip and will utilize 72-byte packets and full-duplex, bidirectional channels with 8-bit wide links. As described in Section 7.2.2, the arbitration between two sides of a half-duplex pipelined channel carries substantial performance penalties. With LAN links being potentially quite long, half-duplex links represented a poor design choice. The projected speed of operation is 180 Mhz using the byte-wide channels for a bidirectional channel bandwidth of 360 Mbytes/s. The LAN switch will be constructed using a network of Chaos router chips. For example, a 16-port switch can be constructed using a 4×4 torus network of Chaos routers. Routing internal to the switch is expected to be normal Chaos routing while switch to switch routing is expected to use source routing and standard FIFO channels with link pipelining, if the links are long enough. Packet-level acknowledgments will be used for flow control across the LAN links. With the design of a special-purpose interface, the goal is to obtain host-to-host latencies down in the 10–20 μs range. This new-generation chip design is being streamlined with clock cycle times approaching 5.6 ns (simulated). This would produce a minimal delay through the router of four cycles or 22.4 ns. Packet formats and channel control have also been smoothed in this next generation chip to streamline the message pipeline implementation.

Arctic Router

The Arctic routing chip is a flexible router architecture that is targeted for use in several types of networks, including the fat tree network used in the *T multiprocessor [269]. The basic organization of the router is illustrated in Figure 7.33, and the current prototype is being used as part of a network with PowerPC compute nodes. Four input *sections* can be switched to any one of four output sections. The router is input-buffered and each input section is comprised of three buffers. Each buffer is capable of storing one maximum sized packet of 96 bytes in support of VCT switching. The crossbar switch connects the 12 inputs to four outputs. Message packets have one of two priority levels. The buffers within an input unit are managed such that the last empty buffer is reserved for a priority packet.

The physical channels are 16-bit full-duplex links operating at 50 MHz, providing a bandwidth of 200 Mbytes/s for each link. The internal router datapaths are 32 bits wide. The Arctic physical channel interface includes a clock and frame signal as well as a flow control signal in the reverse direction. The flow control is used to enable counter increment/decrement in the transmitter to keep track of buffer availability on the other side of the channel. The initial value of this counter is configurable enabling the chip to interface with other routers that support different buffer sizes.

Routing is source-based, and networks can be configured for fat trees, Omega, and grid topologies. In general, two subfields of the header (perhaps single bits) are compared with two reference values. Based on the results of this comparison, 2 bits are selected from the packet header, or some combination of header and constant 0/1 bits are used to produce a 2-bit number to select the output channel. For the fat tree routing implementation, 8 bytes of the message are devoted routing, control, and CRC. The routing header is comprised of two fields with a maximum width of 30 bits: 14 bits for the path up the tree, and 16 bits for the path down the tree to the destination. Since routing is source-based, and multiple alternative paths exist up the tree, the first field is a random string: Portions of this field are used to randomize the choice of the up path. Otherwise a portion of the second field is used to select an appropriate output headed down the tree to the destination. Rather than randomizing all of the bits in the first field, by fixing the values of some of the bits such as the first few bits, messages can be steered through regions of the lower levels of the network. Source based routing can be utilized in a very flexible way to enforce some load balancing within the network. For example, I/O traffic could be steered through distinct parts of a network to minimize interference with data traffic. Messages traversing the Arctic fabric experience a minimum latency of six cycles/hop.

The Arctic chip devotes substantial circuitry to test and diagnostics. CRC checks are performed on each packet, and channel control signals are encoded. Idle patterns are continuously transmitted when channels are not in use to support error detection. On-chip circuitry and external interfaces support static configuration in response to errors: ports can be selectively disabled and flushed and the route tables used for source routing can be recomputed to avoid faulty regions. An interesting feature of the chip is the presence of counters to record message counts for analysis of the overall network behavior.

R2 Router

The R2 router evolved as a router for a second-generation system in support of low-latency high-performance interprocessor communication. The current network prototype provides for interprocessor communication between Hewlett-Packard commercial workstations. The R2 interconnect topology is a 2-D hexagonal interconnect and the current router is designed to support a system of 91 nodes. Each router has seven ports: six to neighboring routers and one to a local network interface (referred to as the fabric interface component or FIC). The topology is wrapped, i.e., routers at the edge of the network have wraparound links to routers at the opposite end of the dimension. The network topology is shown in Figure 7.34. For clarity, only one wrapped link is illustrated. As in multidimensional tori, the network topology provides multiple shortest paths between each pair of routers. However, unlike tori, there are also two *no-farther* paths. When a message traverses a link along a no-farther path, the distance to the destination is neither increased nor decreased. The result is an increase in the number of routing options available at an intermediate router.

The physical channel is organized as a half-duplex channel with 16 bits for data and 9 bits for control. One side of each link is a router port with an arbiter which governs channel mastership. The routers and channels are fully self-timed, eliminating the need for a global clock. There are no local clocks and resynchronization with the local node processor clocks is performed in the network interface. The rate at which the R2 ports can toggle is estimated to be approximately 5 – 10 ns corresponding to an equivalent synchronous transmission rate of 100 – 200 MHz.

Message packets may be of variable length and upto a maximum of 160 bytes. Each packet has a 3-byte header that contains addressing information, and a 16-byte protocol header that contains

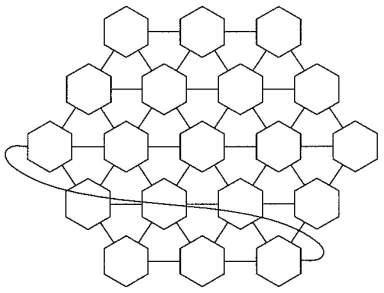


Figure 7.34. Wrapped hexagonal interconnect used in the R2-based network.

information necessary to implement a novel *sender-based protocol* for low-latency messaging. A 4-byte CRC trailer rounds out the message packet format. Two types of messages are supported: normal messages and priority messages. While normal messages may be as large as 64K packets in length, priority messages are limited to a single packet. Typically priority messages are used by the operating system while normal messages are generated by the applications.

A block diagram of the R2 router is illustrated in Figure 7.35. In the figure the ports are drawn as distinct inputs and outputs to emphasize the data path, although they are bidirectional. Each FIFO buffer is capable of storing one maximum-length packet. There are 12 FIFO buffers for normal packets and three FIFO buffers for storing priority packets. Two 7×15 crossbars provide switching between the buffers and the bidirectional ports. Each of the FIFO buffers has a routing controller. Routing decisions use the destination address in the packet header and the local node ID. Buffer management and ordered use of buffers provides deadlock freedom. The implementation is self-timed, and the time to receive a packet header and to compute the address of an output port is estimated to take between 60 and 120 ns.

Routing is controlled at the sender by specifying a number of *adaptive credits*. Each adaptive credit permits the message packet to traverse one link that takes the packet no further from the

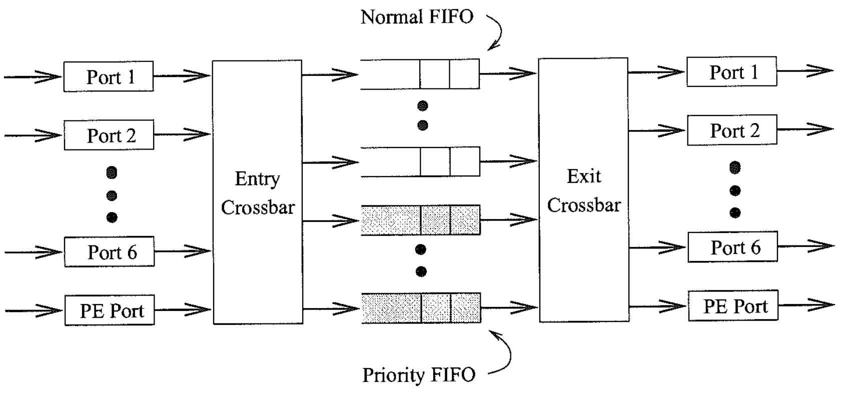


Figure 7.35. Organization of the R2 router.

destination. At any node, there are always two no-farther directions. The adaptive credit field in the header is decremented each time a message takes an adaptive step and further adaptive routing is not permitted once this value is reduced to zero. Shortest paths are given priority in packet routing. Priority packets are only routed along shortest paths. When all outputs for a priority packet are blocked by a normal packet, the priority packet is embedded in the normal packet byte stream being transmitted along one of the shortest paths, analogous to the implementation of virtual channels. An interesting feature of the R2 router is the support for message throttling. Packet headers are monitored and in the presence of congestion, rapid injection of packets from a source can cause negative acknowledgments to be propagated back to the source to throttle injection.

Error handling was considered at the outset during the initial design, and basic mechanisms were provided to detect packets which could not be delivered. Once detected, these packets were ejected at the local node to be handled in software by higher-level protocols. A variety of conditions may cause message exceptions. For example, the message packet headers contain a field that keeps track of the maximum number of hops. This field is decremented at each intermediate router, and when this value becomes negative, the message is ejected. An ejection field in the header is used to mark packets that must be forwarded by the router to the local processor interface. The router also contains watchdog timers that monitor the ports/links. If detection protocols time out, a fault register marks the port/link as faulty. If, the faulty port leads to a packet destination, this status is recorded in the header and the packet is forwarded to another router adjacent to the destination. If the second port/link to the destination is also found to be faulty, the packet is ejected.

7.2.5 Circuit Switching

Circuit switching along with packet switching represent the first switching techniques used in multiprocessor architectures. This section presents one such example and some more recent ideas toward improving the performance of the next generation of circuit-switched networks.

Intel iPSC Direct Connect Module (DCM)

The second generation network from Intel included in the iPSC/2 series machines utilized a router that implements circuit switching. Although most machines currently use some form of cut-through routing, circuit-switched networks do present some advantages. For example, once a path is setup, data can be transmitted at the full physical bandwidth without delays due sharing of links. The system can also operate completely synchronously without the need for flow control buffering of data between adjacent routers. While the previous generation of circuit-switched networks may have been limited by the path lengths, the use of link pipelining offers new opportunities.

The network topology used in the iPSC series machines is that of a binary hypercube. The router is implemented in a module referred to as the *direct connect module* (DCM). The DCM implements dimension-ordered routing and supports up to a maximum of eight dimensions. One router port is reserved for direct connections to external devices. The remaining router ports support topologies of up to 128 nodes. The node architecture and physical channel structure is shown in Figure 7.36. Individual links are bit-serial, full-duplex, and provide a link bandwidth of 2.8 Mbytes/s. Data, control, and status information are multiplexed across the serial data link. The strobe lines supply the clock from the source. The synchronous nature of the channel precludes the need for any handshaking signals between routers.

When a message is ready to be transmitted, a 32-bit word is transferred to the DCM. The least significant byte contains the routing tag — the exclusive-OR of the source and destination addresses. These bits are examined in ascending order by each routing element. A bit value of 1 signifies that

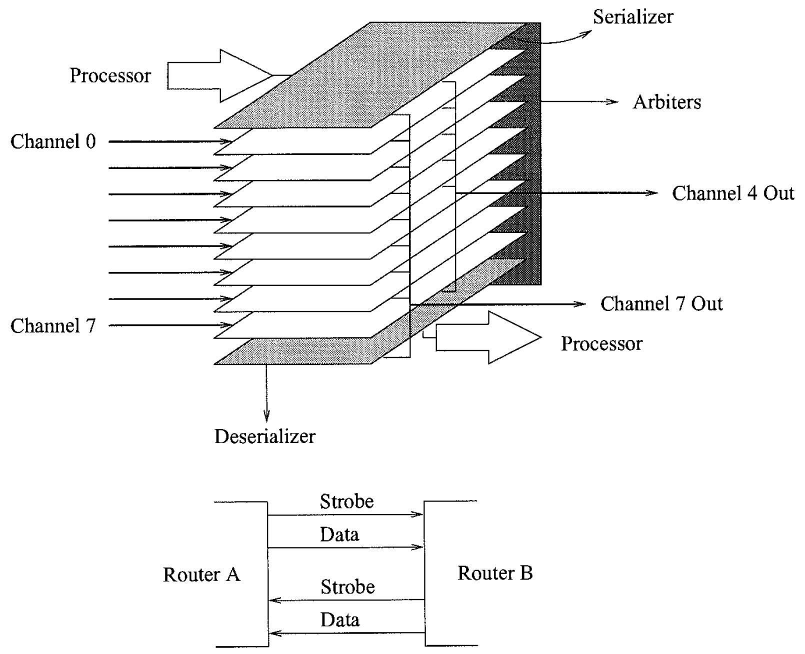


Figure 7.36. The iPSC/2 circuit switched router organization.

the message must traverse the corresponding dimension. The input Serializer generates a request for the channel in the dimension corresponding to the least significant asserted bit in the routing tag. Subsequent routing elements at intermediate nodes receive the header, and make similar requests to output channels. Multiple requests for the same output channel are arbitrated and resolved in a round-robin fashion. Note the connectivity at the outputs in Figure 7.36. An outgoing channel can expect to receive a message only from lower-numbered channels. This significantly reduces the design complexity of the routing elements. Channel status information is continually multiplexed across the data lines in the reverse direction. The status information corresponds to *RDY* or *EOM* (end of message) signals. These signals achieve end-to-end flow control. For example, when the destination DCM receives a routing header, the *RDY* status signal transmitted back to the source DCM initiates the actual transfer of data. The *EOM* status signal appended to the message causes the reserved links to be freed as it traverses the path. The propagation of status signals through the DCM from input to output is achieved by keeping track of the channel grants generated during the setup of the original message path. This end-to-end flow control also permits the propagation of "not ready" status information to the source to enable throttling of message injections.

Optimizing Circuit Switching

Communication requirements heavily depend on the application running on the parallel machine and the mapping scheme used. Many applications exhibit both spatial and temporal communication locality. If the router architecture supports circuit switching, conceivably compile time analysis could permit the generation of instructions to set up and retain a path or circuit that will be heavily used over a certain period of time. It is possible to design a router architecture such that a circuit is set up and left open for future transmission of data items. This technique was proposed in [39] for systolic communication. It has also been proposed in [159] for message passing and extended in [82]. The underlying idea behind preestablished circuits is similar to the use of cache memory in a processor: a set of channels is reserved once and used several times to transmit messages.

In structured message-passing programs, it may be feasible to identify and set up circuits prior to actual use. This set up may be overlapped with useful computation. When data are available, the circuit would have already been established, and permit lower latency message delivery. However, setting up a circuit in advance only eliminates some source software overheads such as buffer allocation, as well as routing time and contention. This may be a relatively small advantage compared to the bandwidth wasted by channels that have been reserved but are not currently used. Circuits could be really useful if they performed like caches, i.e., preestablished circuits actually operate faster. Link-level pipelining can be employed in a rather unique way to ensure this cache-like behavior. We use the term *wave pipelining* [102] to represent data transmission along such high-speed circuits.

It is possible to use wave pipelining across switches and physical channels, enabling very high clock rates. With a preestablished circuit, careful design is required to minimize the skew between wires in a wave-pipelined parallel data path. Synchronizers are required at each delivery channel. Synchronizers may also be required at each switch input to reduce the skew. Clock frequency is limited by delivery bandwidth, by signal skew, and by latch setup time. With a proper router and memory design, this frequency can potentially be higher than that employed in current routers, increasing channel bandwidth and network throughput accordingly. Spice simulations [102] show that up to a factor of four increase in clock speed is feasible. Implementations would be necessary to validate the models. The use of pipelined channels allows the designer to compute clock frequency independently of wire delay, limited by routing delay and switch delay. An example of a router organization that permits the use of higher clock frequencies based on the use of wave pipelining across both switches and channels is shown in Figure 7.37.

Each physical channel in switch S_0 is split into $k + w$ virtual channels. Among them, k channels are the control channels associated with the corresponding physical channels in switches $S_1, \ldots, S_k$. Control channels are only used to set up and tear down physical circuits. These channels have capacity for a single flit and only transmit control flits. Control channels are set up using pipelined circuit switching and handled by the PCS routing control unit. The remaining w virtual channels are used to transmit messages using wormhole switching and require deeper buffers. They are handled by the wormhole routing control unit. The hybrid switching technique implemented by this router architecture was referred to as *wave switching* [102]. With such a router architecture, it is possible to conceive of approaches to optimizing the allocation of link bandwidth in support of real-time communication, priority traffic, or high throughput.

7.2.6 Pipelined Circuit Switching

The implementation of routing protocols based on pipelined circuit switching present different challenges from those presented by wormhole-switched routers, notably in the need to provide support for backtracking and the distinction between control flit traffic and data flit traffic. Ariadne [7] is a router for PCS supporting the MB-m family of fault-tolerant routing protocols. The current implementation is for a k-ary 2-cube with 8-bit-wide, half-duplex physical data channels. There are two virtual data channels and one virtual control channel in each direction across each physical link. The router is fully asynchronous and is comprised of three major components — the control path, the physical link interface, and the data path.

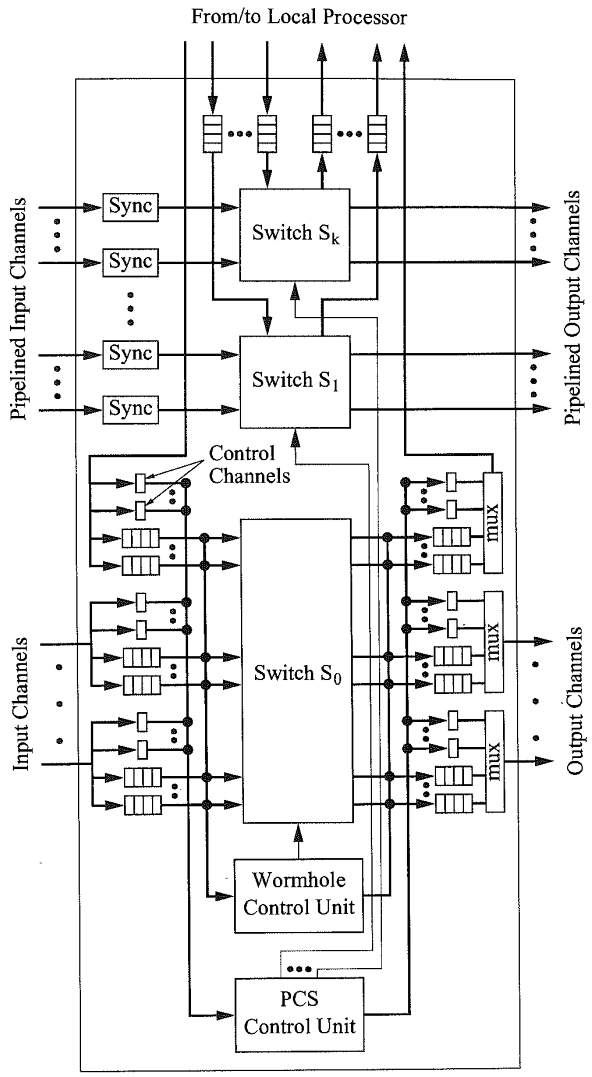


Figure 7.37. Organization of a router employing wave pipelining. (Mux = Multiplexer; PCS = Pipelined circuit switching; Sync = Synchronizer.)

Figure 7.38 illustrates the block diagram of the architecture of the Ariadne chip. Each physical link has an associated link control unit (LCU). These LCUs are split in the diagram with the input half on the left and the output half on the right. Each input LCU feeds a data input buffer unit (DIBU) and a control input buffer unit (CIBU). The buffer units contain FIFO buffers for each virtual channel over the link. The data FIFO buffers feed directly into distinct inputs of the 9 × 9 crossbar.

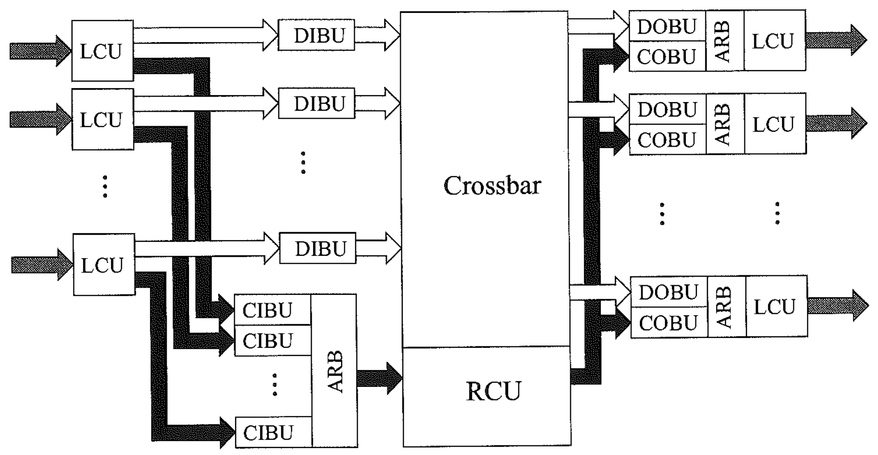


Figure 7.38. Block diagram of Ariadne: a PCS router.

The internal control path is indicated by the solid arrows. Control and data flits arrive over the physical link and are differentiated by the LCU. Control flits are first directed to the CIBUs. The CIBUs send their flits to the router control unit (RCU) via a common bus. The CIBUs arbitrate for use of the bus via a distributed, mutual exclusion mechanism. Once a CIBU has acquired the privilege to use the bus, it presents its routing header and virtual channel address. When the RCU receives a routing header, a routing decision is made, a crossbar mapping is modified, and the header is directed to a control output buffer unit (COBU) via a common output bus.

Figure 7.39 illustrates the signals that comprise each physical link. There are four pairs of request (R) and acknowledge (A) lines. Since the links are half-duplex, nodes must arbitrate for the privilege to send data over the link. One request/acknowledge pair for each side is used in the arbitration protocol (R_{ARBn}, A_{ARBn}). Once a node has the privilege to send, the other request/acknowledge pair is used in the data transfer (R_{DATAn}, A_{DATAn}). An *Accept* line is used to indicate whether or not the flit was accepted by the receiving node or rejected due to input buffer overflow. While there are more pin-efficient methods of implementing these internode handshaking protocols [193], the use of a self-timed design and new switching technique led to the adoption of a more conservative approach.

Virtual channels are granted access to the physical channel using a demand-driven mutual exclusion ring. A signal representing the privilege to drive the physical channel circulates around this ring. Each virtual channel has a station on the ring and will take the privilege if the channel has data to send. In addition, there is also a station on the ring that represents the routing node on the other side of the link (these are half-duplex channels). If the other node has data to send, the privilege will be taken, and the other node will be given control of the link. This scheme multiplexes data in a true demand-driven, rotating fashion among all virtual channels in both directions over the physical link.

While PCS requires one control channel for each virtual data channel, Ariadne combines all control channels into one common virtual control channel. Originally done to reduce circuit area, it also reduces the number of stations on the arbitration rings devoted to control flit FIFO buffers. While it may increase the control flit latency to the RCU, the control flit traffic through the network

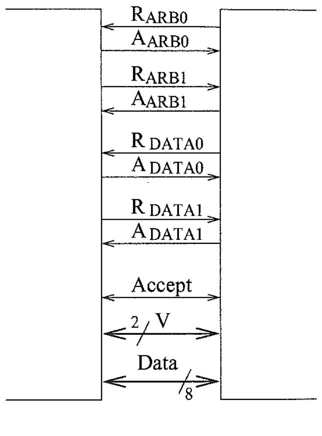


Figure 7.39. Physical link.

is substantially less than the data flit traffic. Each message creates two control flits (the header and acknowledgment flits) and any number of data flits. For most realistic message sizes, it is the data flit population that will be the determinant of message throughput and latency. The larger the message, the smaller the effect of collapsing control channels into a single channel. For example, with 32-flit messages, control flits will represent about 6% of the offered load.

Backtracking protocols must use history information. Rather than carry this information within the routing header, leading to large headers [52, 62], history information is stored locally at a router when necessary. With each virtual data channel we associate a history mask containing a bit for each physical channel. Backtracking protocols use the history mask to record the state of the search at an intermediate node. For example, when a header flit arrives on a virtual channel, say f, the history mask for virtual channel f is initialized to 0, except for the bit corresponding to the physical link of f. The header is routed along some outgoing physical link, say g and the bit for link g is set in the history mask of f. If eventually the header returns to the node backtracking across g, the routing header is prevented from attempting link g again because the bit for g in the history mask of f is set.

The current control flit format is shown in Figure 7.40. All header flits have the Header bit set. Headers will have the Backtrack bit set or cleared depending on whether they are moving forward or backtracking. Acknowledgment flits will have the Header bit cleared and the Backtrack bit set. The next implementation of Ariadne is expected to use 16-bit flits which will allow for larger X and Y offsets as well as an increase in the number of misroutes that are permitted.

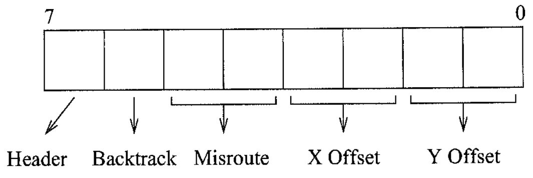


Figure 7.40. Control flit format.

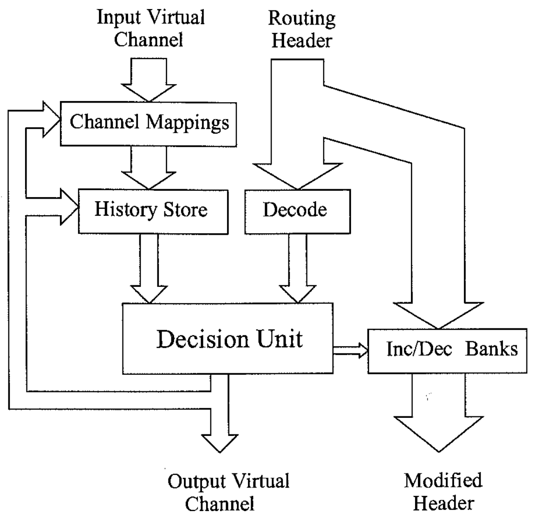


Figure 7.41. Routing control unit.

The routing control unit is more complex than in wormhole-switched routers since history information must be extracted, and the routing protocols are fully adaptive necessitating the selection of one output channel from several candidates. Figure 7.41 shows a block diagram of the RCU. The first step in routing is decoding the routing header and fetching history information. When a forward going routing header enters a node, the history mask is initialized. When a routing header is backtracking, its history mask must be retrieved from the history store. Since the history mask is associated with the virtual channel the header first arrived on, the RCU must first determine the header's point of origination from the crossbar. Once the history information is either initialized or retrieved, a routing decision is made and the values in the outgoing header flit are modified accordingly. The necessary offsets are selected after the routing function has been computed [57]. The selection function favors profitable directions over misrouting.

Data flits are directed to the DIBUs by the LCU. The data input FIFO buffers feed the input ports of a 9×9 crossbar. It is a full virtual-to-virtual channel crossbar — four physical channels with two virtual channels/physical channel and one channel to the processor. The data output FIFO buffers receive their inputs from the output ports of the crossbar. The crossbar is constructed in such a way that the mapped status of every input and output can be read by the RCU. In addition, the RCU may query the crossbar to determine the input that is mapped to an output. Thus, the RCU can obtain the information necessary to direct backtracking headers and acknowledgment flits to their parent node in the path.

Critical sections of the design were simulated using Spice with the level three CMOS transistor models provided by MOSIS for the HP CMOS26B 0.8 μm process. Due to the self-timed nature of the router, the intranode delays incurred by control and data flits is nondeterministic due the arbitration for the RCU and physical links. Since routing headers follow a different path through the router than data flits, the delays are presented in separate tables. The routing header delay is provided in Table 7.5. Adding all of the components, the intranode latency of routing headers ranges from 21 to 48 ns (line 1, Table 7.5). Note that this includes both intranode delay and pad

Table 7.5. Simulated (Spice) timing: Path setup.

Parameter	Time
Control path	13.75 ns + RCU + $2t_{arb}$
RCU (Forward-going header)	13.0 – 16.0 ns
RCU (Acknowledgment)	7.5 ns
RCU (Backtracking header)	16.0 – 19.0 ns

Table 7.6. Simulated (Spice) timing: Data path.

Parameter	Time
Data path	8.5 ns + t_{arb}
Link cycle time (1 active channel)	16.4 ns + t_{synch} + $2t_l$
Link cycle time (>1 active channels)	13.65 ns + $2t_l$
Link idle time for direction change	4.3 ns + t_l

delays as well as channel arbitration delays. Table 7.6 summarizes the major delay components for data flits in Ariadne. In the table, t_{synch} represents the portion of the channel delay that cannot be overlapped with arbitration while t_{arb} represents the worst-case delay in arbitration. Modeled datapath bandwidths are in the vicinity of 50–75 Mbytes/s depending on traffic and message distribution.

7.2.7 Buffered Wormhole Switching

This variant of wormhole switching is employed in the SP-2 interconnection network: a bidirectional multistage interconnection network constructed from 4 $\times$ 4 bidirectional buffered crossbar switches. Two stages of four switches each, are organized into a *frame* that supports up to 16 processors. The architecture of a frame is shown in Figure 7.42. Each crossbar switch has four bidirectional inputs and four bidirectional outputs, and therefore operates as an 8 $\times$ 8 crossbar. In this two-stage organization, except for processors on the same switch, there are four paths between every pair of processors connected to the same frame. The use of bidirectional links avoids the need for costly (i.e., long) wraparound links from the outputs back to the inputs. Multiple frames can be connected as shown in Figure 7.42 to connect larger numbers of processors, either by directly connecting frames (up to 5 frames or 80 processors) or by cascading frames together. In the cascaded organization, switch frames are used in the last stage to provide wraparound links as shown in Figure 7.42. As the systems grow larger, the number of paths between any pair of processors grows. For example, the 128-node configuration provides 16 paths between any pair of frames.

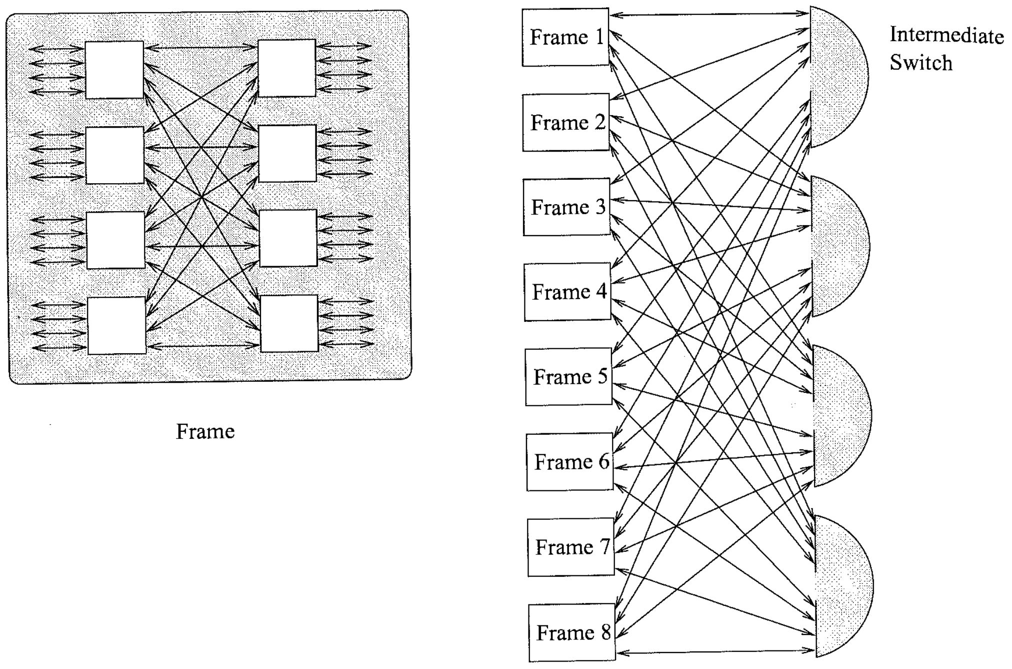


Figure 7.42. System interconnect using an SP-2 frame.

The SP-2 physical channel appears as shown in Figure 7.43. The data channel is 8 bits wide, with a single-bit tag signal and a single-bit token signal in the reverse direction. With the addition of a clock signal, the channel is 11 bits wide. SP-2 flits are matched to the channel width at 8 bits. An active tag signifies valid token on the channel. Tokens are encoded in two cycle frames: a sequence of 0–1 represents no tokens and 1–0 represents two tokens. The token sequence is used to increment a counter in the receiving port. The counter keeps track of the number of flits that can be sent and is decremented each time a flit is transmitted. The other patterns are used for error detection.

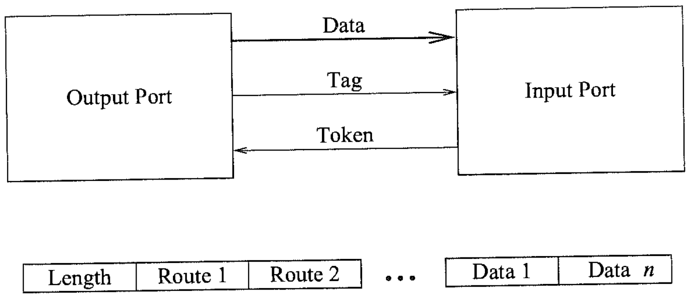


Figure 7.43. SP-2 physical channel and message packet formats.

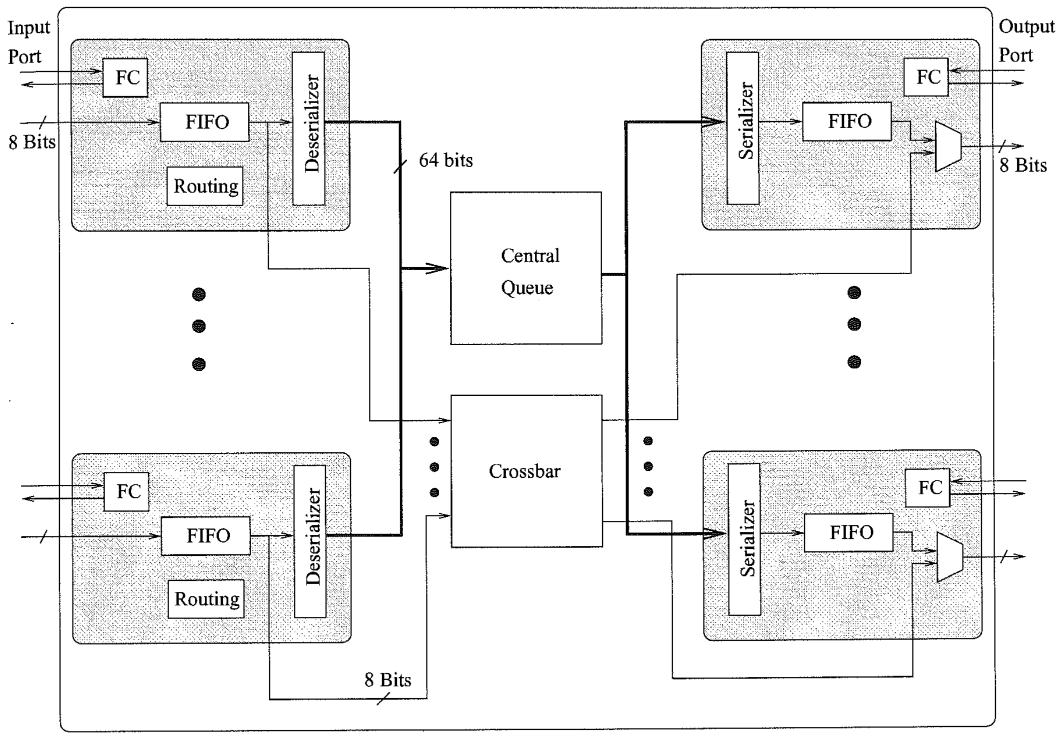


Figure 7.44. Architecture of the SP-2 router. (FC = Flow control.)

Due to propagation delays on the physical link, the input port must be able to buffer a number of flits greater than twice the channel delay: buffers are 31 flits or bytes. The transmitting side of the link consumes a token for each flit transmitted on the channel. The first flit of the message is a byte that contains the message length in flits, thus limiting message sizes to a maximum of 255 flits. Successive flits contain routing information, one routing flit for each frame (two switches) traversed by the message.

Figure 7.44 illustrates the block diagram of the architecture of the SP-2 switch. The behavior of buffered wormhole routing was described in Chapter 2. Messages are routed using source routing. Message flits are received into an input FIFO queue where the first router flit is decoded to request an output port, and CRC checks are performed. Bits 6–4 determine the output port on the first switch in a frame, while bits 2–0 determine the output port of the second switch in a frame. Bit 7 determines which of these two fields is used by the switch — when it is cleared, bits 6–4 are used. When bit 7 is set bits 2–0 are used, and the input port logic discards the routing flit and decrements the length flit in the message. Thus the message becomes smaller as it progresses toward the destination. It takes one cycle to request an output port and receive permission to transmit. If the requested port is available, it is a single cycle through the crossbar to move a flit. If the output port is not free or the request not granted by the time a chunk is received, the chunk must be buffered in the central queue. Such a chunk is referred to as a critical chunk since it is ready for immediate forwarding to an output port. Note that if any message is buffered in the central queue, a output port will not accept a request from the corresponding input port. Subsequent chunks of the message will be buffered in the central queue as space is available. Such chunks are noncritical. As the preceding chunks of a message are transmitted, noncritical chunks become critical chunks, i.e., they are ready for transmission. Storage in the central queue is allocated as follows. One chunk is statically allocated to each output port (to hold the current critical chunk). Remaining chunks are allocated dynamically to incoming messages as they become available and are stored as described in Chapter 2.

Output ports arbitrate among crossbar requests from input ports according to a least-recently-served discipline. Output port requests for the next message chunk from the central queue are also served according to a least-recently-served discipline. Note that the internal data path to the central buffer is 64 bits wide, i.e., 8 flits. Thus, one receiver can be served every cycle, with one chunk being transmitted to the central queue or the output port in one cycle. One transmitter can also be served every cycle. Each chunk is 8 flits, which takes eight cycles to receive. Thus, the bandwidth into the central queue is matched to the bandwidth out of the queue.

The SP-2 switch interface is controlled by a powerful communications coprocessor — a 64-bit Intel i860 CPU with 8 Mbytes of memory. On one side, this processor communicates with an SP-2 switch through a pair of 2-Kbyte FIFO queues. On the other side, the processor interfaces to the RS/6000 CPU Micro Channel bus via a pair of 2 Kbyte FIFO channels. One DMA engine transfers messages to and from the co-processor card and Micro Channel, and a second DMA engine transfers data between the Micro Channel FIFO buffers and the switch interface FIFO buffers. This interface can achieve bandwidths approaching that of the switch links [328].

Like the Cray T3D, the SP-2 employs source routing. The route tables are statically constructed and stored in memory to be used by the messaging software. The construction and use of the routes is performed in a manner that prevents deadlock and promotes balanced use of multiple paths between pairs of processors. In this manner an attempt is made to fully use the available wire bandwidth.

At the time of this writing the next generation of IBM switches were becoming available. The new switches utilize 16-bit flits and the crossbar path is 16 bits wide. Internally the switches operate at 75 MHz as opposed to 40 MHz in the current router. The physical channel structure remains the same, except that differential signaling results in doubling the number of signals per port to 22. The physical channel also operates at 150 MHz. Chunks are now 128 bits wide and the central queue can store 256 chunks for a total of 4 Kbytes. Since the flit size and clock speed has doubled, so have the buffers on the input ports. Flow control and buffer management necessitates 63-flit buffers, or 126 bytes. With the wider internal switch data paths and higher clock rates, the memory-to-memory bandwidth, including Micro Channel transfers, is approaching 100 Mbytes/s for large messages.

7.2.8 Network of Workstations

The first half of the 90s has seen a relentless, and large (close to doubling every 1.5 years) improvement in the price/performance ratio of workstations and personal computers. There is no evidence of this trend fading any time soon. With the concurrent trends in portable parallel programming standards, there has been an explosive growth in research and development efforts seeking to harness these commodity parts to create commercial off-the-shelf multiprocessors. A major architectural challenge in configuring such systems has been in providing the low latency and high throughput interprocessor communication traditionally afforded by multiprocessor backplanes and interconnects in a cluster of workstations/PCs. We are now seeing a migration of this interconnect technology into the cluster computing arena. This section discusses two examples of such networks.

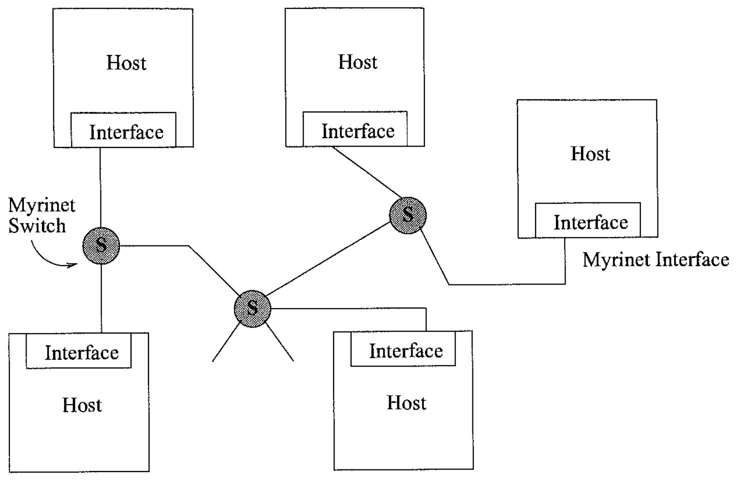


Figure 7.45. An example of a network configured with Myrinet.

Myrinet

Myrinet is a switching fabric that grew out of two previous research efforts, the CalTech Mosaic Project [314] and the USC/ISI ATOMIC LAN [107] project. The distinguishing features of Myrinet includes the support of high performance wormhole switching in arbitrary LAN topologies. A Myrinet network comprises of host interfaces and switches. The switches may be connected to other host interfaces and other switches as illustrated in Figure 7.45, leading to arbitrary topologies.

Each Myrinet physical link is a full-duplex link with 9-bit-wide channels operating at 80 MHz. The 9-bit character may be a data byte or one of several control characters used to implement the physical channel flow control protocol. Each byte transmission is individually acknowledged. However, due to the length of the cable up to 23 characters may be in transit in both directions at any given time. As in the SP-2, these delays are accommodated with sufficient buffering at the receiver. The *STOP* and *GO* control characters are generated at the receiver and multiplexed along the reverse channel (data may be flowing along this channel) to implement flow control. This slack buffer [314] is organized as shown in Figure 7.46c. As the buffer fills up from the bottom, the *STOP* control character is transmitted to the source when the buffer crosses the *STOP* line. There is enough remaining buffer space for all of the flits in transit as well as the flits that will be injected before the *STOP* character arrives at the source. Once the buffer has drained sufficiently (i.e., crosses the *GO* line) a *GO* character is transmitted and the sender resumes data flow. The placement of the *STOP* and *GO* points within the buffer are designed to prevent constant oscillation between *STOP* and *GO* states, and reduce flow control traffic. A separate control character (*GAP*) is used to signify the end-of-message. To enable detection of faulty links or deadlocked messages, non-*IDLE* characters must be periodically transmitted across each channel. Longer timeout intervals are use to detect potentially deadlocked packets. For example, this may occur over a nonfaulty link due to bit errors in the header. In this case, after 50 ms the blocked part of the packet is dropped, and a forward reset signal is generated to the receiver and the link buffers reset.

Myrinet employs wormhole switching with source routing. The message packets may be of variable length and are structured as shown in Figure 7.46c. The first few flits of the message header

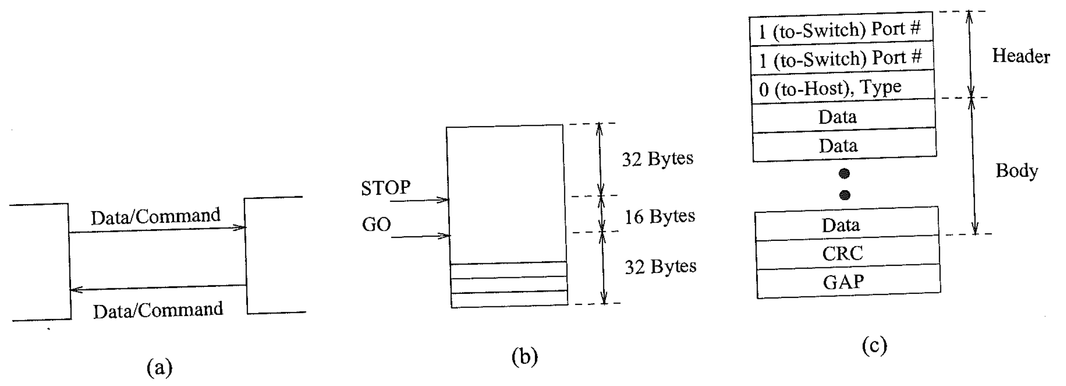


Figure 7.46. (a) The Myrinet physical link. (b) Organization of the receiver buffer. (c) Myrinet message packet format.

contain the address of the switch ports on intermediate switches. In a manner similar to the SP-2 switches, each flit addresses one of the output ports in a switch. Each switch strips off the leading flit as the message arrives and uses the contents to address an output port. Conflicts for the same output port are resolved using a recirculating token arbitration mechanism to ensure fairness. The remaining flits of the message are transmitted though the selected output port. Once the message reaches the destination, the Myrinet interface examines the leading flit of the message. This flit contains information about the contents of the packet, e.g. control information, data encoding, etc. For example, this encoding permits the message to be interpreted as an Internet Protocol (IP) packet. The most significant bit determines if the flit is to be interpreted by the host interface or a switch. The header flits are followed by a variable number of data flits, an 8-bit CRC, and the *GAP* character signifying the end of the message.

The switches themselves are based on two chips: a pipelined crossbar chip and the interface chip that implements the link-level protocols. The maximum routing latency through an eight-port switch is 550 ns [30]. At the time of this writing, four-port and eight-port switches are available with 16- and 32-port switches under development. The Myrinet host interface resembles a full-fledged processor with memory and is shown in Figure 7.47. Executing in this interface is the Myrinet control program (MCP). The interface also supports DMA access to the host memory. Given this basic infrastructure, and the programmability of the interface it is possible to conceive of a variety of implementations for the messaging layer and indeed, several have been attempted in addition to the Myrinet implementation. A generic implementation involves a host process communicating with the interface through a pair of command and acknowledgment queues. Commands to the interface cause data to be accessed from regions in memory, transferred to the interface memory, packetized, and transmitted out the physical channel. Acknowledgment of completion is by a message in the acknowledgment queue, or optionally the generation of an interrupt. Message reception follows a similar sequence of events where the header is interpreted, data transferred to locations in host memory, and an acknowledgment (or interrupt) generated. Typically a set of unswappable pages are allocated in host memory to serve as the destination and source of the interface DMA engine. In general, the Myrinet interface is capable of conducting transfers between the interface and both host and kernel memory. The MCP contains information mapping network addresses to routes, and performs the header generation. In any Myrinet network, one of the interfaces serves as the *mapper*: it is responsible for sending mapping packets to other interfaces. This map of the network is

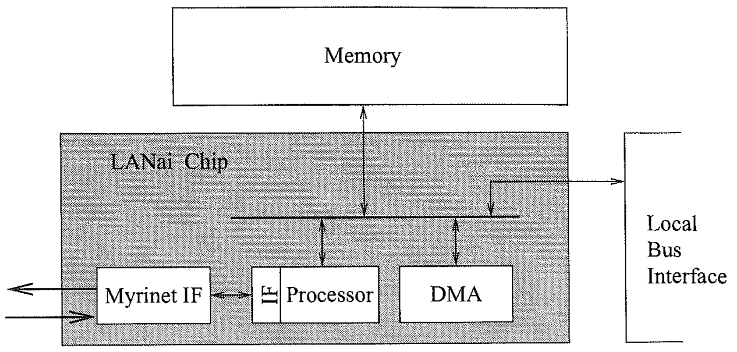


Figure 7.47. Architecture of the Myrinet interface. (DMA = Direct memory access; IF = Interface.)

distributed to all of the other interfaces. Now all routes can be created locally by each interface. The network maps are computed in a manner that is guaranteed to provide only deadlock-free routes. The attractive feature of the Myrinet network is that should the mapper interface be removed or faulty links separate the network, a new mapper is selected automatically. Periodic remapping of the network is performed to detect faulty links and switches, and distribute this information throughout the network.

More recently, Myrinet designs have evolved to a three-chip set. The single-chip crossbar is used for system area networks (SAN) where the distances are less than 3 meters. For longer distances, custom VLSI chips convert to formats suitable for longer distances of up to 8 meters typically used in LANs. A third chip provides a 32-bit FIFO interface for configurations as classical multicomputer nodes. Switches can be configured with combinations of SAN and LAN ports producing configurations with varying bisection bandwidth and power requirements. The building blocks are very flexible and permit a variety of multiprocessor configurations as well as opportunities for subsequently growing an existing network. For example, a Myrinet switch configured as a LAN switch has a cut-through latency of 300 ns while a SAN switch has a cut-through latency of 100 ns. Thus, small tightly coupled multiprocessor systems can be configured with commercial processors to be used a parallel engines. Alternatively, low latency communication can be achieved within existing workstation and PC clusters that may distributed over a larger area.

ServerNet

ServerNet [15, 156, 157, 158] is a SAN that provides the interconnection fabric for supporting interprocessor, processor-I/O and I/O-I/O communication. The goal is to provide a flexible interconnect fabric that can reliably provide scalable bandwidth. ServerNet is built on a network of switches that employ wormhole switching. Figure 7.48 illustrates the structure of the physical channel and the organization of the router. Like Myrinet, ServerNet is based on point-to-point switched networks to produce low latency communication. The structure and design of ServerNet was heavily influenced by the need to integrate I/O transfers and interprocessor communication transfers within the same network fabric.

The structure and operation of the physical channel was described in Example 7.3. The choice of 8-bit channels and command encoding was motivated by the corresponding reduction in pin-out and complexity of control logic for fault detection. The link speed of 50 MHz presents a compromise

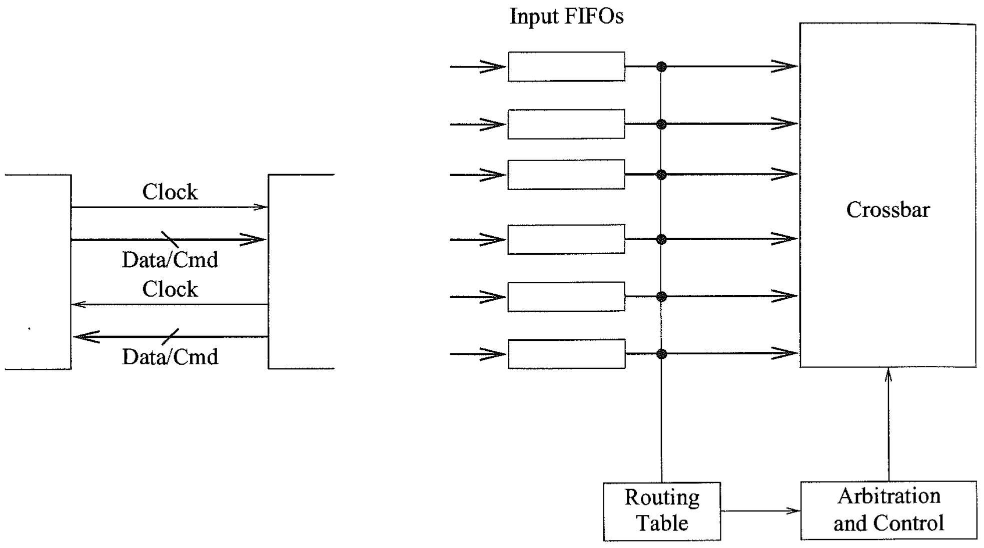


Figure 7.48. The ServerNet channel and router organization. (CMD = Command.)

between cost and the performance of modern peripheral devices. If increased bandwidth is required between end points, additional links can be added to the network. The router is a six-port, wormhole-switched router as shown in Figure 7.48. Virtual channels are not used. Since source clocking is employed to transmit flits, a synchronizing FIFO is used to receive flits at the sender's clock and output flits on the receivers clock to the input buffers. The flit buffers are 64 bytes deep. As described in Example 7.3, flow control for buffer management is implemented via commands in the reverse direction. A 1,024-entry routing table uses the destination address in the header to determine the output port. The packet now enters arbitration for the output port. Each input possesses an accumulator that determines the priority of the input channel during arbitration. The channel that gains the output channel has the associated accumulator value reduced by an amount equal to the sum of the accumulators of the remaining channels that are seeking access. The channels losing arbitration increment the value of their accumulator. This scheme avoids starvation. Deadlock freedom is achieved by design in computing the values of routing table entries. The latency through the router is on the order of 300 ns and the measured latency for a zero length packet through one switch is on the order of 3 μs.

Message headers consist of an 8-byte header which contains 20-bit source and destination IDs, message length, and additional control and transaction type information. The message header is followed by an optional 4-byte ServerNet address, variable-sized data field, and a trailing CRC. The ServerNet address is used to address a location in the destination node's address space. For interprocessor communication, the host interface will map this address to a local physical address. In this manner processor nodes can provide controlled and protected access to other nodes. All messages generate acknowledgments. Timers are used to enforce timeouts on acknowledgments. There are four message types: read request (17 bytes), read response (13 + N bytes), write request (17 + N bytes) and write response (13 bytes). The value of N represents the data size. The maximum data size is 64 bytes.

The six-port architecture can be used to create many different topologies including the traditional mesh, tree, and hypercube topologies. ServerNet is structured to use two disjoint networks: the X network and the Y network. Duplication in conjunction with acknowledgments for each packet, and error checking by means of CRC are used to realize reliable operation.

7.3 Engineering Issues

The discussions of the network optimizations and router architectures have necessarily included engineering issues and trade-offs. In this section we speculate on important trends and their impact on future networks. The design of a scalable, reliable computing system is fundamentally about the optimization of metrics such as cost, performance, and reliability in the presence of physical constraints. The emerging advances in packaging technologies, e.g., low cost MCMs, 3-D chip stacks, etc. will lead to major changes in the relationship between these physical constraints, and as a result significantly impact the cost, performance, and reliability of the next generation of interconnection networks. An accurate analysis of the effect of packaging technology and options early in the system design process will enable the implementation of systems that effectively utilize available packaging technology.

Provided with a high-level design, the fundamental question relates to where should we place die, multichip module, board, and subsystem boundaries. Die boundaries may be determined by the use of commercial parts. MCM boundaries may be determined by cost, performance, or thermal density. These partitioning decisions are critical. To make the most effective use of packaging technology we must be able to predict the impact of packaging technology on a particular design. The type of trade-offs that can be made at the packaging level include the following.

- High packaging densities (80–90%) can be realized with flip chip bonding and MCMs. This process is also more expensive and must be weighed against the reliability of a larger number of I/Os. If the implementation strategy is one of an active backplane, high-density multipath networks become viable, particularly in embedded multiprocessor applications.

- High wiring densities (50 μm pitch vs. 500 μm pitch) can be achieved in multichip packaged substrates. This has a direct, positive effect on the achievable bisection bandwidth of the network but is achieved at a higher cost and possibly entails greater power densities.

- Area array bonding can provide higher chip pin-out changing the fundamental trade-offs of what goes on- and off-chip. Off-chip speed penalties may no longer be as severe depending on the package substrate materials, e.g., organic polymer dielectric layers and copper interconnects.

- Due to the nonlinear cost/yield relationships, monolithic designs may be more cost-effectively implemented on multiple smaller die. Thus larger switches and increased buffering may become viable architectural options.

- Global interconnects on MCM substrates are not as lossy as intrachip interconnects. As a result, comparable or better performance can be obtained via the use of MCM interconnects. Furthermore, large reductions in interchip delays are possible due to lower parasitics involved in the MCM die attach technologies, hence redefining the partitioning of designs across multiple die. Several smaller die may be a cost-effective option to a larger monolithic die. This off-chip penalty has historically motivated several generations of computer architecture designs.

The above trade-offs must be made once node design and interconnect options have been proposed. These packaging technologies are making the use of multiprocessor architectures in embedded sensor processing applications quite feasible in the foreseeable future. However, doing so will require the optimization of the topologies for these new constraints.

A second issue is embedded in the advent of cluster-based computing with workstations and PCs. We are seeing the migration of the expertise gained from the design of several generations of high speed multiprocessor interconnects, to the design of low-latency, robust cluster interconnects. Among the many architectural issues being resolved in this context is the point at which the network is integrated into the local node (e.g., memory, as an I/O device, etc.), software/hardware interface to network bandwidth, and the design of message layers that realize low-latency communication when constrained to use commodity parts. Some of the software issues are discussed in Chapter 8.

7.4 Commented References

Dally [71] originally studied the design of k-ary n-cube networks under the assumption of VLSI implementations wherein network implementations were wire-limited. Wiring constraints were captured in the form of bisection width constraints. Networks were compared by fixing the available bisection width to that of a binary hypercube with bit-wide, full-duplex, data channels. Under different wire delay models the analysis concluded that low-dimensional networks provided a better fit between form and function, resulting in better overall latency and throughput characteristics. Subsequently, Abraham and Padmanabhan [1] studied the problem under the assumption of fixed pin-out constraints. In this environment, the wider channel widths of low-dimensional networks were offset by the greater distances. Thus, their analysis tended to favor higher-dimensional networks. Bolding and Konstantinidou [33] observed that the maximum available pin-out is not independent of the bisection width. Thus, they proposed using the product of the bisection width and pin-out as a measure of the cost of the network. Thus comparisons could be made between two (quantifiable) equal-cost networks. Agarwal's [3] analysis included the effect of switch delays, which was missing from prior analysis. In this model, the analysis favored slightly higher-dimensional networks than Dally's analysis [71]. One of the principal inhibitors of the use of higher-dimensional networks has been the wire delay due to the long wires that result when these networks are laid out in the plane. Scott and Goodman [310] studied the ability to pipeline bits on the wire, effectively using these long propagation delays as storage to create pipelined wires. As result, their analysis too favored higher-dimensional networks. These techniques are reflected in the switching fabric that is produced in current generation networks. Finally, the continuous evolution of high-density packaging with area-bonded die is resulting in the capability for producing chips with very high number of I/O connections. This relaxation of the pin constraint will no doubt favor lower-dimensional networks due to the constraints on bisection width.

The first routers were packet-switched and circuit-switched, following from previous experiences with multicomputer communications in the LAN environment. While packet-switched networks enabled deadlock-free routing via structured memory allocation, message latencies in tightly coupled multiprocessors were very high due to the linear relationship between interprocessor distance and latency. The advent of cut-through routing significantly reduced this latency, but the need to buffer packets at the nodes required the involvement of the node processors: we can expect this trend to be quite inefficient in the tightly coupled multiprocessor world. The advent of small-buffer cut-through or wormhole switching made it feasible to construct single-chip, compact, fast routers that did not require the services of intermediate nodes. A generation of router chips were based on this approach [77, 79, 255]. As densities continued to increase, it was possible to include enough buffer space on the single chip to support VCT switching [41, 83, 188]. With better understanding of the advantages as well as limitations of cut-through switching, efforts started focusing on optimizations to further improve the performance or reliability leading to new router architectures [7, 316, 327].

Finally, we are seeing a convergence of techniques for routing in LANs and multiprocessor backplanes. As clusters of workstations become economically viable, focus has shifted to the support of efficient messaging mechanisms and hardware support. The new generation of router architectures reflects this trend [30, 156]. We can expect this trend to continue, further blurring the distinctions between clusters of workstations and tightly coupled multiprocessor backplanes.

EXERCISES

7.1 Rewrite the expression for the no-load message latency in a k-ary n-ary cube under the following assumptions. Rather than a router architecture that is buffered at the input and output, assume that the router has input buffering only. Thus, in a single cycle, a flit can progress from an input buffer on one router to an input buffer of another router, if the corresponding output link at the first router is free. Furthermore, assume that the routing delay and the switch delay scale logarithmically with the network dimension.

Solution We have the equation for the no-load latency given by

$$t_{wormhole} = n\frac{k}{4}\left[r + s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right] + \max\left[s, 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left\lceil \frac{L}{W} \right\rceil \tag{7.25}$$

Each stage of the message pipeline from the source to the destination now consists of transmission across the router and the link. Therefore, the coefficient of the second term becomes the sum of the wire delay and switch delay. The new expression is

$$t_{wormhole} = n\frac{k}{4}\left[r + s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right] + \left[s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left\lceil \frac{L}{W} \right\rceil \tag{7.26}$$

This equation can be rewritten as

$$t_{wormhole} = rn\frac{k}{4} + \left[s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left(n\frac{k}{4} + \left\lceil \frac{L}{W} \right\rceil\right) \tag{7.27}$$

The first term is the component due to routing delays (in the absence of contention). The coefficient of the second term corresponds to the clock cycle time of the network.

7.2 Individual analysis of the networks has been based on their use of a single resource, e.g., wiring density or pin-out. Actually both node degree and bisection are related. A better measure of network cost/performance appears to be the product of node degree and bisection width proposed in [33]. Determine the relationship between channel widths of two networks of equal cost using this performance metric. Compare a binary hypercube and an equivalent-sized 2-D mesh using this metric.

Solution In general, consider two topologies of dimension r and s with the same number of nodes. We know that the product of their bisection bandwidth and pin count are equal. The width of each dimension in the r-dimensional topology is given by $N^{\frac{1}{r}}$. This formula is derived from the relation $N = k_m^r$. The pin-out is given by (including only data pins) $2rw_r$ where w_r is the channel width. We can now write the products of the bisection bandwidth and pin-out as

$$(2w_r N^{1-\frac{1}{r}})(2rw_r) = (2w_s N^{1-\frac{1}{s}})(2sw_s)$$

From the above expression we have

$$\frac{w_r}{w_s} = \sqrt{\frac{sN^{1-\frac{1}{s}}}{rN^{1-\frac{1}{r}}}}$$

Substituting the correct number of dimensions for a binary hypercube and a 2-D mesh, we have

$$\frac{w_{hyp}}{w_{mesh}} = \sqrt{\frac{2N^{1-\frac{1}{2}}}{rN^{1-\frac{1}{r}}}}$$

which simplifies to

$$\frac{w_{hyp}}{w_{mesh}} = \sqrt{\frac{4\sqrt{N}}{N\log_2 N}}$$

7.3 Early in this chapter, an expression was provided for the relative length of the longest wire in a k-ary n-cube when embedded in two dimensions. Repeat this analysis when a k-ary n-cube is embedded in three dimensions.

Solution Consider the base case of of a 3-D cube with each dimension interleaved as shown in Figure 7.2. A k-ary n-cube is embedded in three dimensions by embedding $\frac{n}{3}$ dimensions of the network into each of the three physical dimensions. Let us assume that the number of dimensions is a multiple of 3. Each additional dimension beyond the first three increases the number of nodes in the network by a factor of k. Embedding in three dimensions provides an increase in the number nodes in each physical dimension by a factor of $k^{\frac{1}{3}}$. If we ignore wire width and wiring density effects, the length of the longest wire in each physical dimension is also increased by a factor of $k^{\frac{1}{3}}$. Therefore for n dimensions the longest wire will grow by a factor of $k^{\frac{n-3}{3}} = k^{\frac{n}{3}-1}$. Note that similar arguments lead to the general case of the wire length of embeddings in d dimensions being $k^{\frac{n}{d}-1}$.

PROBLEMS

7.1 Cascading is employed to configure a 512-node IBM SP-2 system. How many disjoint paths exist between any pair of processors?

7.2 Flow control operations across a link take a finite amount of time. A message may be in transit during this period. In pipelined links, several messages may be in transit. The receive buffer size must be large enough to store the phits in progress over a link. Given the maximum number of phits in transit on a physical channel, write an expression for the minimum receive buffer size in phits to prevent any loss of data.

7.3 Consider an m-dimensional and n-dimensional tori of equal cost, where cost is given by the product of the bisection width and pin-out that utilizes this bisection width. Write analytical expressions for the relationships between the channel widths of the two networks and the bisection bandwidths of the two networks.

7.4 Consider embedding a k-ary n-cube in three dimensions rather than two dimensions. Rewrite the expression for average message latency by modifying the expressions that capture the effect of wire length. Compare this to the expression derived in this chapter.

7.5 We are interested in determining the maximum message injection rate for a given bisection bandwidth. Assume we have a 16×16 torus with 1-flit-wide, half-duplex channels. Message destinations are uniformly distributed. Find the maximum message injection rate in flits/node/cycle.

Hint Every message generated by a node is equally likely to cross the bisection, or be addressed to a node in the same half of the network.

7.6 A $4 \times 4 \times 8$ Cray T3D system experiences a single node failure. There are no spare nodes. All of the routing tables are updated to use the longer path in the dimension. Now we are concerned about deadlock. Is deadlock possible? Justify your answer with a proof or an example.

7.7 Source routing as described in the Cray T3D can cause imbalance in the use of virtual channels. For an eight-node ring, compute the degree of imbalance across each physical link. The degree of imbalance across a physical link is defined as the ratio of (1) the difference in the number of source-destination pairs that use virtual channel 0 and virtual channel 1 across the link, and (2) the total number of source-destination pairs that use the link. The choice of the dateline link does not matter.

7.8 What is the bisection bandwidth in bytes of the following machine configurations?

1. A $4 \times 8 \times 4$ Cray T3D
2. An 8×14 Intel Paragon
3. A 64-node IBM SP-2

7.9 Using the following values for the constants in the parameterized cost model shown in Table 7.4, compute the routing latency and the maximum flit bandwidth available to a message.

Constant	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
Value (ns)	0.4	0.6	2.2	2.7	0.6	0.6	1.4	0.6	1.24	0.6