INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road. Ann Arbor. MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9434067

A flexible oblivious router architecture

Park, Joonho, Ph.D.

State University of New York at Binghamton, 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A FLEXIBLE OBLIVIOUS ROUTER ARCHITECTURE

BY

JOONHO PARK

B.S., State University of New York at Binghamton, 1986
M.S., The Pennsylvania State University, 1988

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Advanced Technology with
Specialization in Computer Engineering
in the Graduate School of the
State University of New York
at Binghamton

May 1994

© Copyright by Joonho Park, 1994
All Rights Reserved

Accepted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Advanced Technology with
Specialization in Computer Engineering
in the Graduate School of the
State University of New York
at Binghamton

Prof. Stamatis Vassiliadis Electrical Engineering (Co-advisor)	[illegible]	Date	4/25/94
Prof. José G. Delgado-Frias Electrical Engineering (Committee Chairman and Co-advisor)	[illegible]	Date	5/4/94
Prof. Craig A. Bergman Electrical Engineering (Member)	[illegible]	Date	5/3/94
Prof. Fernando Guzmán Mathematical Science (Member)	Fernando Guzmán	Date	5/4/94
Dr. Joseph F. Skovira IBM Glendale Lab. (External Examiner)	[illegible]	Date	4/25/94

ABSTRACT

A FLEXIBLE OBLIVIOUS ROUTER ARCHITECTURE

In this study, a flexible oblivious router architecture and evaluation of a Dynamically Allocated Multi-Queue (DAMQ) buffer are presented. The architecture is suitable for current technologies and is intended for multiprocessor and massively parallel systems. In the first part of our study, we explore the possibility of proposing a general purpose router architecture. Our study shows that the interconnection networks classified as trees, cubes, meshes and multistage interconnection networks require only a few instructions for executing their routing algorithms. Our investigation strongly suggests that a common router architecture can satisfy at least 40 network topologies with the introduction of a few, very simple to implement instructions. Furthermore, the proposed architecture provides a programming capability that allows other oblivious routing algorithms not considered in our investigation to be accommodated. Our overall conclusion is that general purpose cost effective routers can potentially be designed that perform equally well as customized routing logic, suggesting the possibility of a common router for multiple interconnection networks.

The second part of this study concentrates on a buffer management scheme of a router and its evaluation. We have proposed a new way to implement a Dynamically Allocated Multi-Queue (DAMQ) buffer with a scheme called "self-compacting buffer". This technique is efficient in that the amount of hardware required to manage the buffers is significantly less than previously built routers while it offers high performance by exploiting more channel bandwidth than a First-In First-Out (FIFO) buffer. We also report extensive simulation results comparing the performance of a self-compacting buffer against the ideal buffer management scheme. The comparison extends previous work by considering a much broader range of network topologies, including several examples of k-ary n-cubes and delta networks. In addition, we introduce a single router simulation method that can quickly and accurately approximate the performance of an entire network. The single router simulation is much smaller in its size and faster by an order of magnitude than the full scale simulation; however it predicts performance with remarkable accuracy.

Dedicated to my wife,

Jamie

ACKNOWLEDGMENTS

The completion of this dissertation would not have been possible without the help of many people. I wish to thank my advisor, Professor Stamatis Vassiliadis, for his constant advice, ideas and support. He gave encouragement and guidance for me to successfully complete this work. He also provided me with an opportunity to go one semester, full time study on campus which was extremely helpful to focus on the research. It is hard to describe my appreciation to him in words. I would also like to express my sincere gratitude to my advisor and chairman of the thesis committee, Professor José Delgado-Frias, for his long support, enthusiasm, so many helpful discussions and comments throughout my graduate study. His confidence in me kept me going and made it possible to finish the hard work. My sincere thanks also go to Dr. Brian W. O'krafka for his technical support and encouragement to complete two of my papers. He always shared his busy moments with me to discuss papers and suggested ways to improve the quality of the papers. I want to thank my thesis committee members, Professor Craig A. Bergman, Professor Fernando Guzman and Dr. Joseph F. Skovira for their valuable comments, corrections and support. I especially want to thank Professor Craig A. Bergman for his extra help that improved my dissertation technically as well as grammatically.

I am grateful to many people in IBM Endicott. I owe a lot to Gil Martino who did proof-reading of my dissertation. He patiently corrected my dissertation and gave me valuable comments that made the dissertation more readable and professional. I want to thank my managers, Hanif Dandia and Pete S. Morelli, for supporting me through the Graduate Work Study Program. I would also like to thank my team members, Frank V. Paxhia, Mike A. Leska and Keith J. Kobel for tolerating my busy schedule.

There are so many friends who supported me. I wish to thank Professor Jong Kim for his consistent support with his brilliancy. I also want to thank Professor Joonwon Lee for his encouragement. My thanks also go to so many friends in Binghamton for keeping me alive with coffee breaks, beers and jokes.

I thank my parents and in-laws for their love and confidence in me. Their love and prayers gave me the strength to withstand and overcome difficult times.

Last, but not least, I am grateful to my wife, Jamie, for her love, endless support, patience and encouragement. She has suffered for five years to support my study. She deserves special thanks, that will last forever. And my daughter, Minji, I thank her for being there. Her smile and welcome at the door are the best gifts I receive every day. Finally, I thank my Lord for giving me such happiness and an opportunity to achieve one of the great goals in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Interconnection networks play an important role in the design of parallel systems; they influence the performance of such systems. A key component of an interconnection network is a router which executes different switching technologies, flow controls and routing algorithms [1]. The router can be categorized into two major types depending on the network topology it supports. The first type of router relates to the networks with a fixed topology. In this case, the router performs only one routing algorithm which depends mainly on the chosen topology. Examples within this category include routers such as the Torus routing chip [2], the router supported in the iPSC/2 [3] and Cosmic Cube router [4]. The second type of router relates to the networks with physically reconfigurable topologies. In such systems, the router is able to execute multiple routing algorithms or an algorithm that can support multiple topologies. The Intel/CMU's iWARP[5] and INMOS transputer [6] are examples of this type of router. In this scheme, there are two approaches in routing, namely source routing and lookup table routing [7]. In source routing, the source node determines the routing paths on the underlying network topology. The packet used to communicate between processors has to carry the complete routing information in the header. In lookup table routing, each router has an entry in a table that indicates which output channel has to be used to reach the destination node.

If a router is built using a network with a fixed topology, a different router has to be developed for different network topologies. The implication here is that the router has no flexibility and thus it can not be used as an "off the shelf" component for the design of parallel systems. The second type of router resolves the previously mentioned problem since theses routers can be used as "off the shelf" components to satisfy various network requirements, but introduce some new problems. In particular, in the source routing scheme, the network bandwidth can be wasted when all the complete routing information is included in the header packet. The lookup table can be viewed as flexible, and a potential of hand shaking with necessary computations can be envisioned with the processing element. However, because the routing chip performs no useful operations besides accessing memory locations, the execution of the algorithm can be slow when arbitrary networks are implemented and the look-up table is kept to a small size, especially when the destination address spaces are not easily partitioned into contiguous ranges.

In the first part of our study, we propose a flexible router architecture that can support fixed and/or reconfigurable network topologies. Our work is based on the investigation of 40 network topologies representing five families of networks. Our proposal can be viewed as a general purpose router architecture. In this study, the term architecture denotes the attributes of a system as seen by the programmer, i.e. conceptual structure and functional behavior, and it is distinct from the organization of the dataflow and physical implementation of the machine. Furthermore, it can support interval labeling routing algorithms [6] providing high performance with limited look-up table sizes in the look-up table scheme. Additionally, while maintaining flexibility, the number of instructions to be implemented are limited to a minimal set so that the router can be easily implemented.

While the first part of the study focuses on flexible router architecture for fixed and/or reconfigurable networks, the second part of the study deals with an important router function; the buffer management scheme. We present the design and evaluation of a new buffer management scheme for a router called "self-compacting buffers".

The "self-compacting buffer" technique is an efficient way to build a dynamically allocated multiqueue (DAMQ) buffer, which is one member of a family of buffer architectures defined by Tamir and Frazier [8] (Figure 1.1). In a FIFO buffer, packets are routed through a single read and single write port in first-in, first-out manner (Figure 1.1(a)). With FIFO buffer, packets in an input port may get blocked if they are not the first packet to be routed. The blocking problem can be resolved by providing separate FIFO queues for each output port at every input port. Figure 1.1(b) shows one method of this approach where 4 (4 by 1) crossbar switches are used. This scheme is called *statically allocated, fully connected* (SAFC) buffer. The problem with SAFC buffer is that it requires expensive hardware resources. Furthermore, the utilization of buffer space at the input ports are not as good as FIFO buffers. One approach to reduce the amount of hardware in SAFC buffer is using just one (4 by 4) crossbar switch (Figure 1.1(c)). This scheme is called *statically allocated multi-queue* (SAMQ) buffer. This approach eliminates some of the hardware overhead of SAFC buffer but the buffer is still inefficiently utilized because it is statically allocated. A better way of using the buffer is to dynamically allocate buffer space in a SAMQ buffer. Figure 1.1(d) shows this scheme which is called *dynamically allocated multi-queue* (DAMQ). It is reported that the

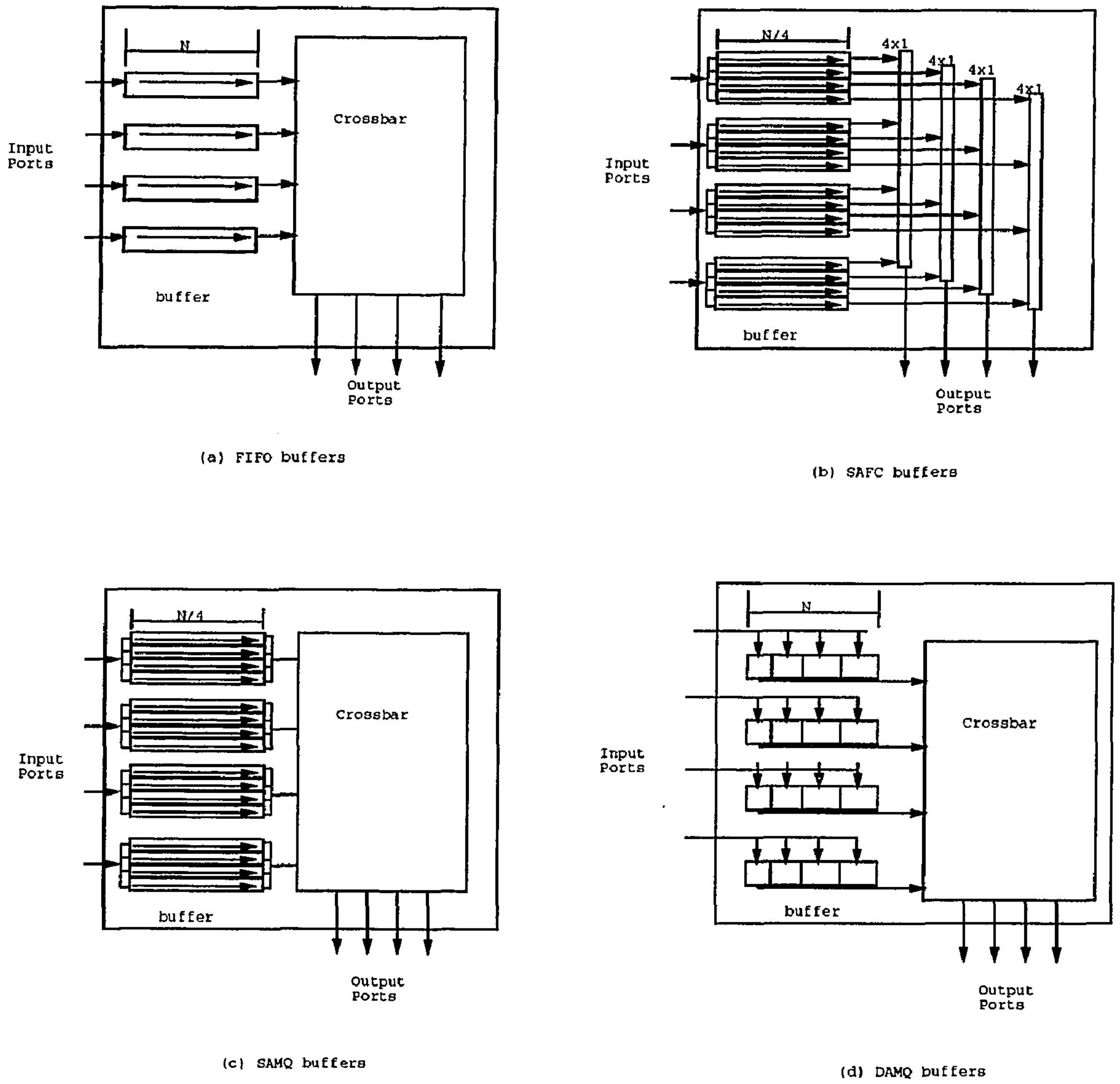


Figure 1.1: Alternative Designs of Buffers

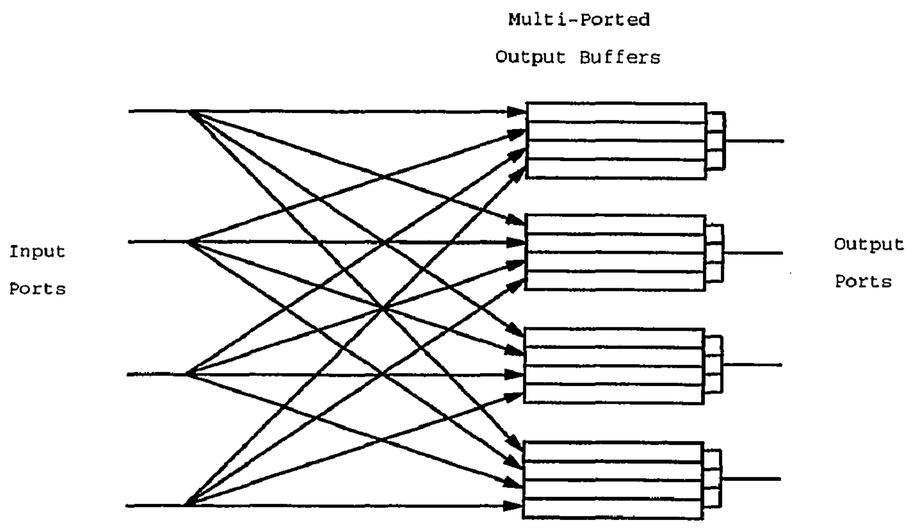


Figure 1.2: Schematic Diagram of Ideal Buffer.

DAMQ buffer achieves the best performance among four buffer types [8]. Finally, it is useful to consider the *ideal buffer* (Figure 1.2), which provides a useful reference point for evaluating the other buffer types. In the ideal buffer, a multi-ported buffer is associated with each output port. If multiple packets request a given output port in the same cycle, the output port can accept all packets simultaneously. This implies that there is no blocking at input ports, unlike the other switch organizations mentioned above. Because of its high complexity, an ideal buffer would be very difficult to implement in practice.

The "self-compacting buffer" technique described in this study applies to packet-switched networks. There are three common flow control schemes namely: store-and-forward, virtual cut-through, and wormhole. In store-and-forward networks, complete packets are received and stored before they are transferred to the next switch. Store-and-forward networks can support high bandwidth, but the network latency is proportional to the product of packet size and the number of hops in the network [7]. The virtual cut-through flow control scheme [9] eliminates this drawback by buffering entire packets only when they can not advance to the next node. As soon as a desired output channel is available, a packet is forwarded (cut-through) without waiting for complete reception of the whole packet. Like store-and-forward switches, virtual cut-through switches require a minimum buffer size of one packet. The wormhole flow control scheme was introduced to eliminate this requirement [10]. Wormhole routing is similar to the cut-through approach but permits packets to remain among multiple switches that are blocked.

This study considers two aspects of DAMQ buffer design. The first is a new way to implement DAMQ buffers with virtual cut-through flow control. The second is an investigation of DAMQ buffer performance for a broad class of multiprocessor networks. We focus on DAMQ buffers because they offer the best performance of the various switch architectures examined by Tamir and Frazier [8]; we focus on virtual cut-through because it is commonly used and offers better performance than alternative flow control schemes. Although virtual cut-through requires larger buffers than wormhole routing, current integrated circuit densities support large buffers at reasonable cost. We extend published performance studies in two ways. First, we consider a much broader class of multiprocessor networks, including k-ary n-cubes and Delta networks. Second, we introduce the *single router simulation* technique, which permits the performance of large networks to be accurately approximated with much less simulation time.

The main contributions of this work are:

- The design of an efficient way to build DAMQ buffers using self-compacting buffer techniques. This design offers comparable performance to a previously published design (the UCLA *ComCobb* chip [8]) at lower hardware cost.

- Extensive simulation results comparing the performance of a self-compacting DAMQ buffer against ideal and FIFO buffer. The comparison extends previous work by considering a much broader range of network topologies, including several examples of k-ary n-cubes and delta networks.

- Introduction of the "single router" simulation method. This technique uses the simulation of a single router element to approximate the performance of an entire network. It drastically reduces simulation time and the complexity of the simulator program itself, with little effect on accuracy. Single router simulation is applicable to buffered networks in which the channel utilizations and routing probabilities are identical (or almost identical) from router-to-router.

The routing algorithm handler (mentioned in the first part of the study and in [11, 12]) and the self-compacting buffer [13] are two important components that manage the internal activity of the router. Besides those two router managing entities, each router requires external interfacing to its neighboring router. The input/output(I/O) port controllers receive/transmit the packets between routers. Among several primary functions, the communication protocol is the basic function of the I/O port controllers. The I/O port controllers are also responsible for carrying signals that are required for a router that supports multiple switching techniques [13]. In addition, the I/O port controller has the capability to support a variable packet length and a variable number of header packets. The support of variable number of header packets is crucial when the routers are used in different sizes of networks because the size of routing information in the header packet may vary. In the third part of our study, we present an efficient architecture of an I/O port controller. Specifically, the I/O port controller includes the following features:

- The ability to support variable packet lengths.
- The ability to support a variable number of header packets.
- Provide a signal propagation capability to support circuit and packet switching.
- A communication protocol to receive and transmit packets.
- Cooperation with the routing algorithm handler and the packet flow controller (DAMQ buffer) to initiate internal activities of the router.

1.1 Dissertation Outline

Chapter 2 provides survey work in areas related to routers. It describes how routers are related to network topologies, routing algorithms and switching techniques with examples of previously designed routers.

Chapter 3 describes a flexible oblivious router architecture. It explains the difference between oblivious and adaptive routing. It introduces the primary components of a router with detailed descriptions. Then, it presents a general instruction set required to execute routing algorithms and a control instruction set to coordinate components of routers. And it describes internals of router architecture in detail.

Chapter 4 is devoted to a new organization for DAMQ with self-compacting buffers and its evaluation. It provides a detailed description of how a self-compacting buffer operates. It compares timing and hardware requirements to existing designs. Then, it shows performance of DAMQ on k-ary n-cubes and delta networks based on simulation results. In addition, it introduces a method of single router simulation which predicts performance of k-ary n-cubes and delta networks with accuracy and a short run time.

Chapter 5 describes the I/O port controller architecture. It presents an efficient communication protocol and ways of supporting multiple switching techniques, variable packet lengths and a variable number of header packets.

Finally, chapter 6 includes a summary and list of contributions of this study as well as directions for future study.

CHAPTER 2

LITERATURE SURVEY

Parallel processors are emerging as the promising technology to achieve high computational power. A parallel system consists of nodes and an interconnection network. The nodes are processing elements which carry out tasks such as number crunching jobs, graphic processing, and vector processing. In order for the nodes to cooperatively work on the jobs, they need to communicate with other nodes for synchronization, exchanging partial results or system control [14, 15, 16]. The exact reasons and types of communication may vary depending on the applications.

The interconnection network as shown in Figure 2.1 is the means by which the nodes communicate with each other. The interconnection network is often the critical component of a large parallel computer because performance of the parallel system is very sensitive to network throughput and latency, and because the network accounts for a large fraction of the cost [17]. Hence, developing high performance interconnection networks has been the subject of extensive research.

A key component of the interconnection network is the router (also known as a switching element). The router is equipped with all the intelligence that is necessary to control network traffic. Thus, developing high performance interconnection networks directly relates to developing high performance routers. Some of the major issues that influence the design of the router include network topol-

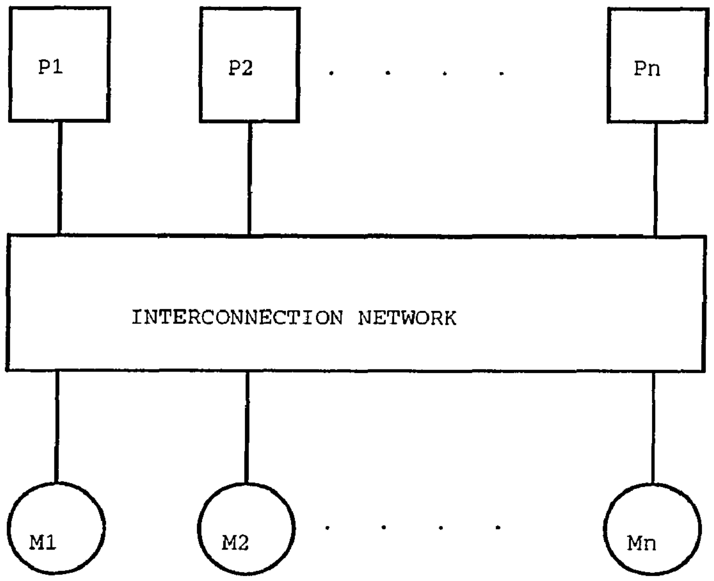


Figure 2.1: Model of Concurrent Processing Systems

ogy, switching techniques, routing and flow control [18, 7]. The following sections present survey results in the area of router design.

2.1 Network Topology

The network topology can be depicted by a graph in which nodes represent switching points and edges represent communication links [19]. Different types of topologies are summarized in Figure 2.2. The design of a router is closely related to the network topology because the routing algorithm that has to be executed by the router depends on the network topology. The following is a description of topologies in a tightly coupled multiprocessor system.

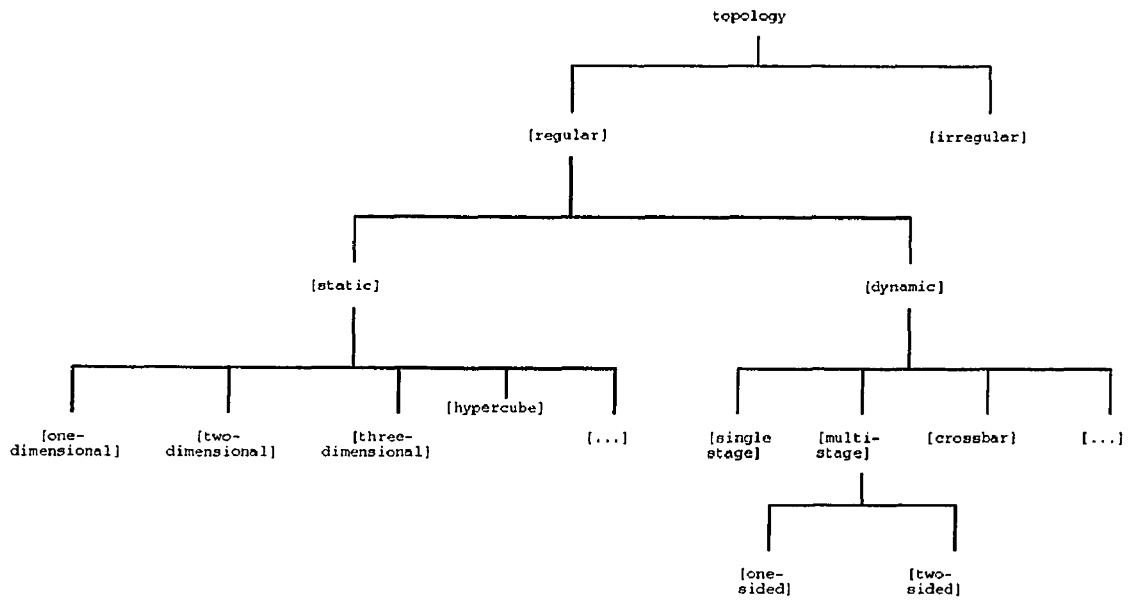


Figure 2.2: Topology of Interconnection Networks

2.1.1 Static Interconnection Network

In the static interconnection network, communication links are passive and cannot be reconfigured for direct connections to other nodes [20]. If every node is connected to every other node, it is called a completely connected network. The completely connected network offers a simple routing algorithm by eliminating intermediate nodes and achieves high network performance. However, the cost of constructing such a network is far greater than the benefits it can offer. Because of this, the completely connected networks are used in multicomputer systems with a relatively small number of nodes that require high throughput and low latency. In terms of cost and performance, other network topologies, such as mesh and hypercube network topologies, have received more attention. These networks provide high network bandwidth with low system cost and require a simple routing algorithm.

Another network that belongs to the static network is ring topology. The ring structure can be constructed by adding one more link to a linear structure shown in Figure 2.3 (a). The ring structure (Figure 2.3(c)) reduces the maximum distance of the linear structure by half. It also adds more network fault tolerance by providing an alternative path to reach a node. Another widely used static network topology is a tree. It is used in an application such as the parallel search algorithm [21]. The Binary Tree [22], Fat Tree [23], Flip Tree [24], Hypertree [25] and KYKLOS networks [26] fall under tree topology. Besides the above mentioned networks, there are other popular networks including Systolic Array, k-ary n-cube, Chordal Ring, Cube-Connected-Cycle, etc.. Figure 2.3 shows several examples of static interconnection networks.

2.1.2 Dynamic Interconnection Network

The dynamic network can be constructed from one or more stages of interconnection networks that employ n x n switching elements. An example of a single stage interconnection network is the shuffle-exchange network. A Multistage Interconnection Network (MIN) is by far the most widely used dynamic network. A MIN has many different topologies. The MIN topologies include the Omega network [27], the Delta network [28], the Baseline network [29] and the BBN's Butterfly network [16]. Most MINs employ a 2 x 2 or a 4 x 4 switching element to route packets. The MIN can be either one-sided or two-sided. The one-sided networks have input-output ports on the same side. The two-sided MIN can be

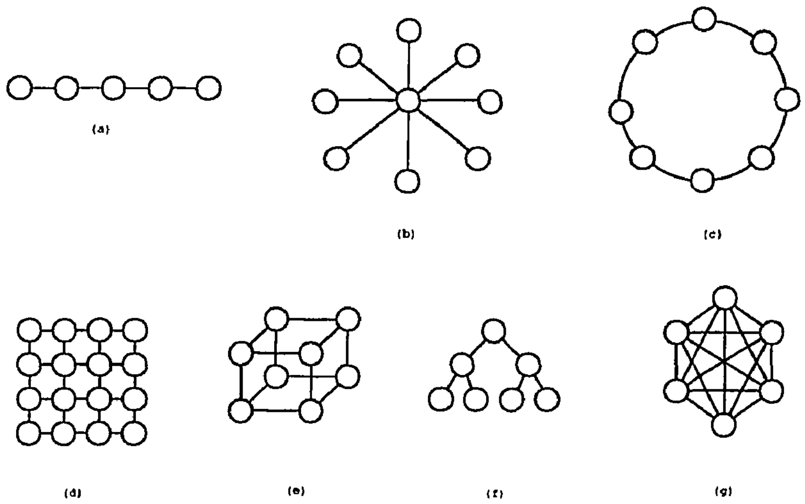
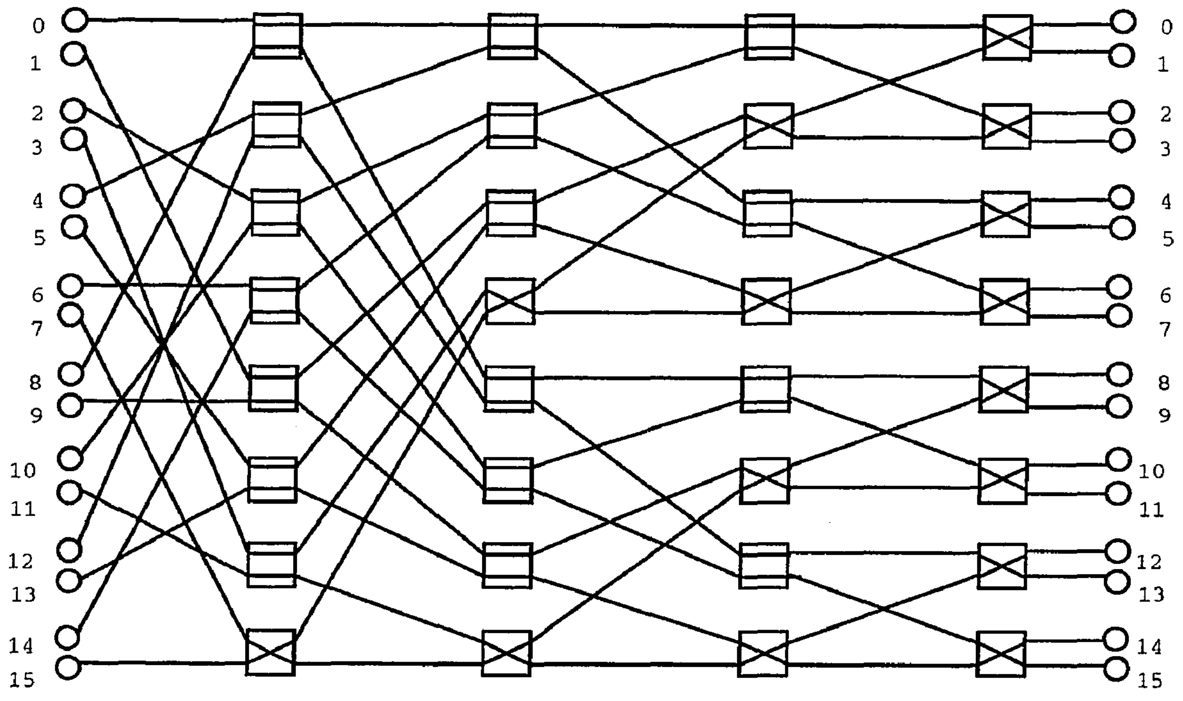


Figure 2.3: Examples of Static Interconnection Networks: (a) Linear (b) Star (c) Ring (d) Mesh (e) Hypercube (f) Binary Tree (g) Completely Connected

divided into three classes: blocking, rearrangeable, and non-blocking. In blocking networks, simultaneous requests to the same connection result in conflict. Only one request is granted at a time and the rest of them are either discarded or buffered. A network is called rearrangeable nonblocking if it can perform all possible connections between inputs and outputs by rearranging its existing connections so that a connection path for a new input-output pair can always be established [19]. In the non-blocking network, all possible connections can be made without blocking.

The important characteristics of the MIN is its flexibility and cost effective communication links between processors. The MIN's flexibility was exploited by Lin and Wu on the baseline network. For example, the baseline network can be configured into a ring structure if all processors, p_i, establish connections to p_{i+1}. Figure 2.4 shows the result of ring structure created from a 16 x 16 baseline network



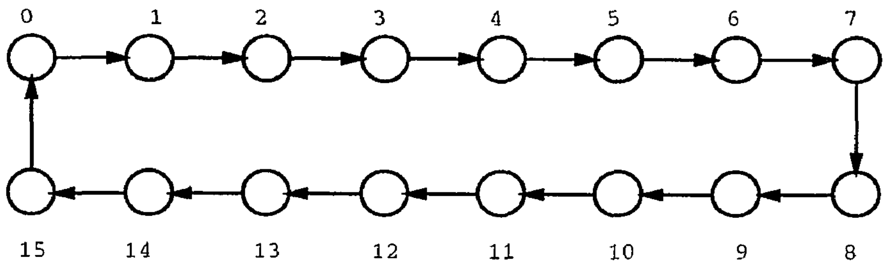


Figure 2.4: Configuration of a Ring Structure on Baseline Network

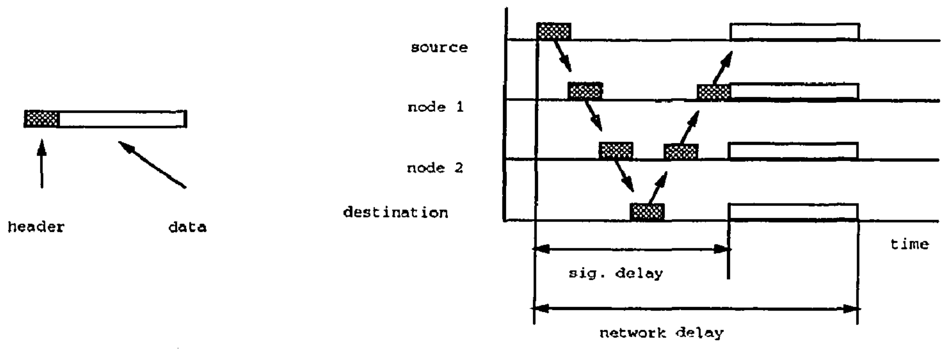
along with physical paths at each switching elements. As it is shown, there are no conflicts at the switching elements and it can effectively function as a ring structure. In [30], Lin and Wu proposed algorithms that can configure the baseline into tree, mesh, ring and/or a mixture of those with variable size networks by partitioning networks.

2.2 Switching Techniques

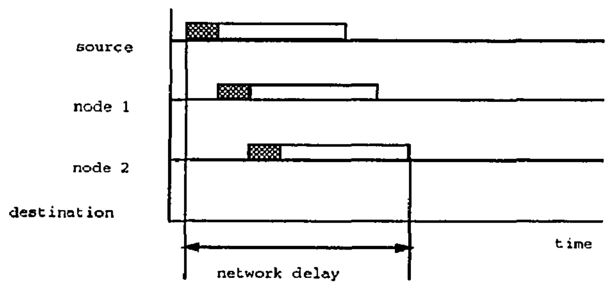
The circuit switching and packet switching are two major switching techniques. In circuit switching, a physical path is actually established between a source and a destination node before transmission of data. In packet switching, data is put in a packet and routed through the interconnection network without establishing a physical connection [20]. The circuit switching is suitable for transmission of bulk data and the packet switching for short messages.

Packet switching can be further classified into three switching techniques called store-and-forward, virtual cut-through and wormhole routing. Figure 2.5 (modified from [9]) shows data transmission of different switching techniques. The store-and-forward switching technique was used in early computer networks. In the store-and-forward, the complete packet is received at the intermediate node before it is forwarded to the next node. This switching strategy was adopted in the research prototype Cosmic Cube and several first-generation commercial multicomputers including the iPSC-1, Ncube 1, Ametek 14 and FPS T-series [7].

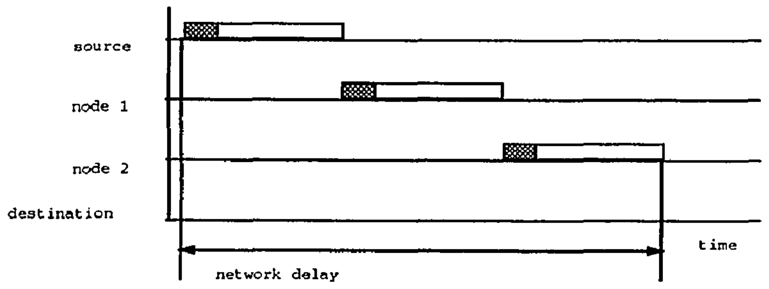
The drawback of the store-and-forward switching was that the network latency is proportional to the packet length. To improve the network performance, Kermani and Kleinrock proposed the virtual cut-through switching technique [9].



(a) circuit switching



(b) wormhole routing



(c) store-and-forward switching

Figure 2.5: Operation of Three Different Switching Systems

In virtual cut-through, the packets are not put into the buffer all the time. Rather, the packets are forwarded immediately to the next node if it is possible, therefore reducing the time wasted in waiting for the arrival of a complete packet. The packets are buffered only if the next node is not capable of receiving packets or if the required output channel is not available. The ComCobb chip developed at UCLA [31] and the Harts at the University of Michigan [32] adopted virtual cut-through as their switching technique.

While virtual cut-through offers better performance than store-and-forward, packets still have to be buffered if they cannot advance forward. Wormhole switching adopts a new technique that eliminates the need for buffering. In wormhole routing, a packet is broken down to a number of flits. This is the smallest block of data for which flow control is maintained. A flit is one or more of phits, which are the size of the physical channel. Figure 2.6 (taken from [1]) shows the relationship between the packet, flits and phits. The header, composed of one or more flits, leads its way through the network and remaining flits follow in a pipelined fashion, thus the name wormhole routing, as shown in Figure 2.7 (taken from [7]). When the header is blocked in a router, the header and following flits are not stored into a buffer. Instead, all flits remain in the network until the blocked condition is cleared. Because of its unique switching technique, wormhole routing has advantages of: (1) reducing the communication latency, compared with store-and-forward, and (2) requiring only a few flit buffers per channel instead of packet buffers at each node like store-and-forward and virtual cut-through require [33]. Numerous systems including Symult 2010, Ncube-2 and iWarp have adopted wormhole routing. By keeping all flits in the network when the header is blocked, wormhole routing simplifies the

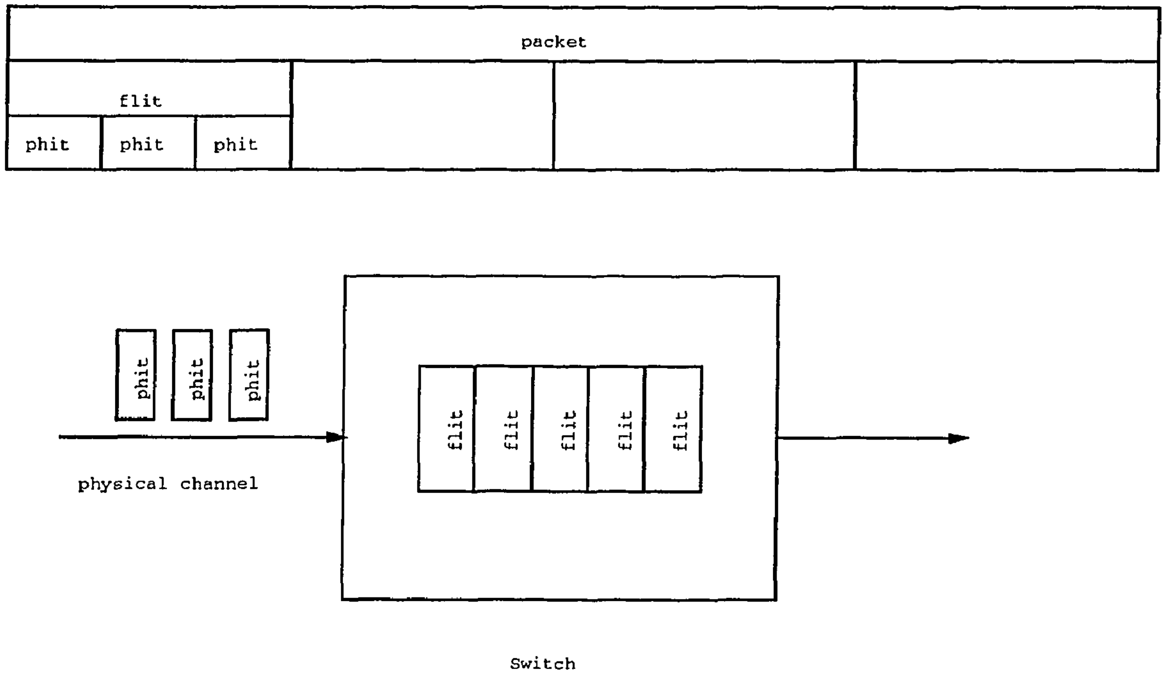


Figure 2.6: Packet-Switching Terminology

buffer control and greatly reduces the overhead spent in buffer control. However, channels may not be shared efficiently as the network traffic gets heavier and the number of flits goes higher.

2.3 Flow Control

Buffers and channels are resources of a switch. Flow control refers to the resource management policy of a switch which it controls (allocation/deallocation of buffers and channels for packet reception/transmission). Since resources are limited, it is possible that resource conflicts occur. The first instance of such a case is when a packet cannot proceed because the receiving switch does not have any

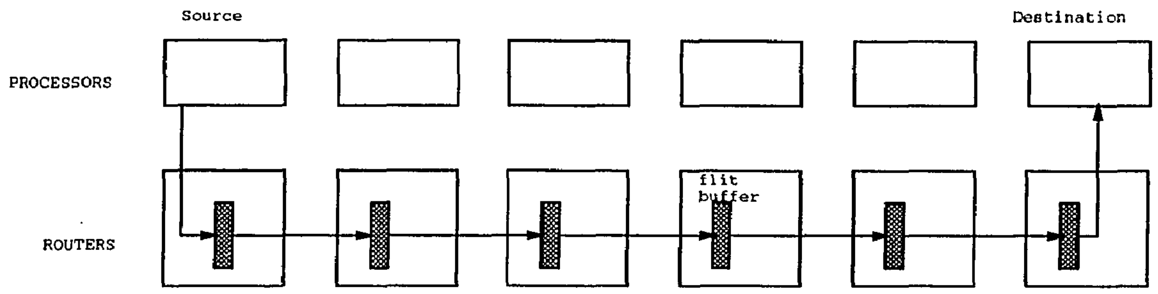


Figure 2.7: Wormhole Routing

free buffer space. In this case, the packet can be thrown away, blocked in place, buffered or rerouted depending on the resource management policy. The second instance of a resource conflict is when two packets are ready to be transmitted through the same output channel at the same time. In this case, there should be a priority policy to resolve the conflict. Well known policies are round robin, first in first out and fixed priority.

The architecture of the buffer and channel greatly influences flow control. Tamir and Frazier investigated four different styles of buffer organization known as First In First Out (FIFO), Statically Allocated Fully Connected (SAFC), Statically Allocated Multi-Queue (SAMQ) and Dynamically Allocated Multi-Queue (DAMQ) buffers [34]. Figure 1.1 in chapter 1 showed those four buffer organizations. In the FIFO buffer, the first packet in the buffer has the top priority for connection to the output port when it is available. The inefficiency of FIFO buffers is that the rest of the packets in the buffer may not be transmitted even if their output ports are ready, until they become the first packet in the buffer.

The SAFC buffer was designed to avoid such starvation. In the SAFC, the separate FIFO buffers are provided for each output port so that if the output port is available, the packet from the associated buffer can be transmitted independent of availability of other output ports and its priority against the priority of packets in other FIFO buffers. Although the SAFC improves performance of the buffer, it also has a few problems. First, it requires an n x n crossbar switch to support n input/output ports. Second, n separate FIFO buffers are also required. Such hardware overhead is not a practical use of chip area in VLSI implementations. In addition, the utilization of buffer space in the SAFC is lower than that of the FIFO. Another problem of supporting the SAFC buffer is the complexity of delivering information. When a router is blocked, its input port has to notify the output port of buffer filled information so that the output does not transfer more packets until the input port of the neighboring router gets free buffer space. Having n separate input FIFO buffers, the amount of information that has to be sent to the neighboring output port is n times more with n x n routers. Furthermore, the output port has to have knowledge of which neighboring input port the packet is destined. Thus, it needs to pre-route the header packet to the input port for determination of appropriate input port and receive the result back. The pre-routing adds more complexity to the design of the SAFC buffer.

The SAMQ buffer is similar to SAFC buffer except that it uses an n x n crossbar switch instead of using n of (n x 1) crossbar switches as was the case with the SAFC buffer. Thus, the SAMQ buffer reduces hardware complexity compared to the SAFC buffer. However, the SAMQ buffer has the same problem that the SAFC had for having the separate FIFO buffers including the pre-routing problem because the SAMQ also uses separate FIFO buffers for each output port like the

SAFC buffer. By using one n x n crossbar, the SAMQ has a lower rate at which the input ports are read out than the SAFC buffer.

The DAMQ buffer also provides separate FIFO buffers for each output port. However, unlike the SAFC and SAMQ buffers, it has physically one input port but the input port is logically divided into multiple FIFO buffers. This was done by using the linked list concept implemented by hardware with pointer registers. In the DAMQ buffer, the size of each FIFO buffer in an input port is flexible. The DAMQ eliminates the pre-routing problem of the SAFC and SAMQ buffer as well as other problems which occur because of having multiple FIFO buffers with physically one input port. The DAMQ buffer achieves higher buffer utilization rate than the SAFC and the SAMQ buffers because of flexible FIFO buffer sizes. In the SAFC and the SAMQ buffer, the size of each FIFO buffer within an input port is (n/t) bytes for a buffer with t I/O ports and n total bytes per input port. If there is packet traffic where the packet coming into an input port is destined to an output port, only (n/t) bytes are utilized with the SAFC and the SAMQ buffers. Under the same condition, the DAMQ buffer can utilize all n bytes for incoming packets because the size of each FIFO buffer is flexible.

2.4 Virtual Channel

Dally developed the concept of virtual channel and presented virtual channel flow control in [17]. This research was motivated by the fact that the throughput of interconnection networks is limited to a fraction (typically 20% - 50%) of the network's capacity because of coupled resource allocation [35]. With virtual channels, one physical channel is divided into multiple virtual channels. Figure 2.8

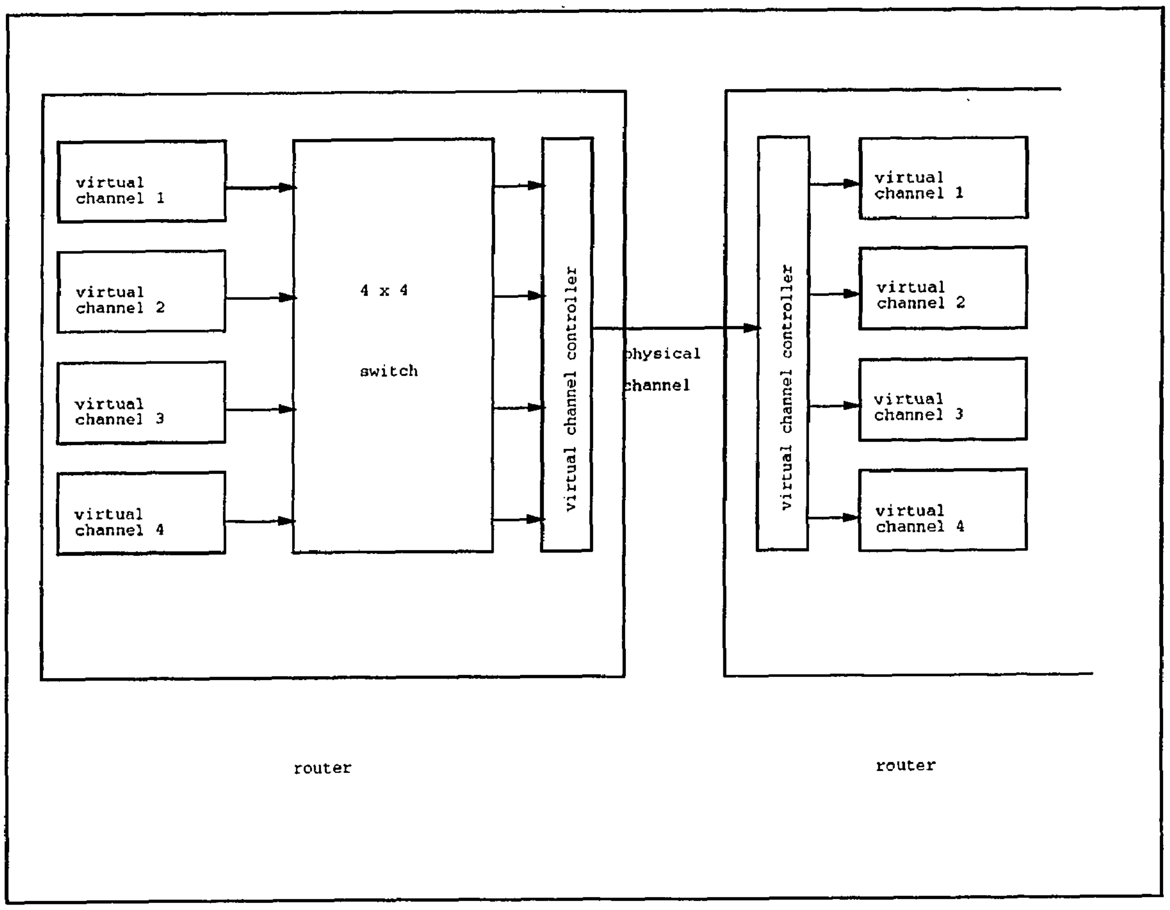


Figure 2.8: Four Virtual Channels Through a Physical Channel

shows the four virtual channels sharing a physical channel. Virtual channels are useful in three ways. First, by increasing the degree of connectivity in the network, they facilitate the mapping onto a particular physical topology of an application in which processes communicate according to another logical topology. Second, even when the application and the architecture have the same topology, extra connections may still be needed to route around congested or faulty nodes. Third, virtual channels provide the ability to deliver a guaranteed communication bandwidth to certain classes of packets [7]. For example, it is important that some bandwidth be reserved to support system related functions, such as debugging, monitoring, and system diagnosis.

2.5 Deadlock

Deadlock in the interconnection network occurs when no packets can advance toward their destination because the buffers in the nodes are full [10]. Figure 2.9 shows an example of deadlock. In Figure 2.9, the buffers in all nodes are full and no packets can advance toward their destinations. When the deadlock occurs in the network, communication within the deadlocked nodes is not possible and some mechanism to resolve the deadlock situation has to be applied. One way of resolving the deadlock situation is by preemption. Preempted packets are either thrown away or rerouted. Either way, preempted packets have to be reliably delivered to the destination and this requires complex circuitry. Such extra hardware will increase the network latency. Because of this, the resolving deadlock by preemption is not a common practice. In general, deadlocks are avoided by the routing algorithm. In this way, there is no need for extra hardware for resolving the deadlock situation.

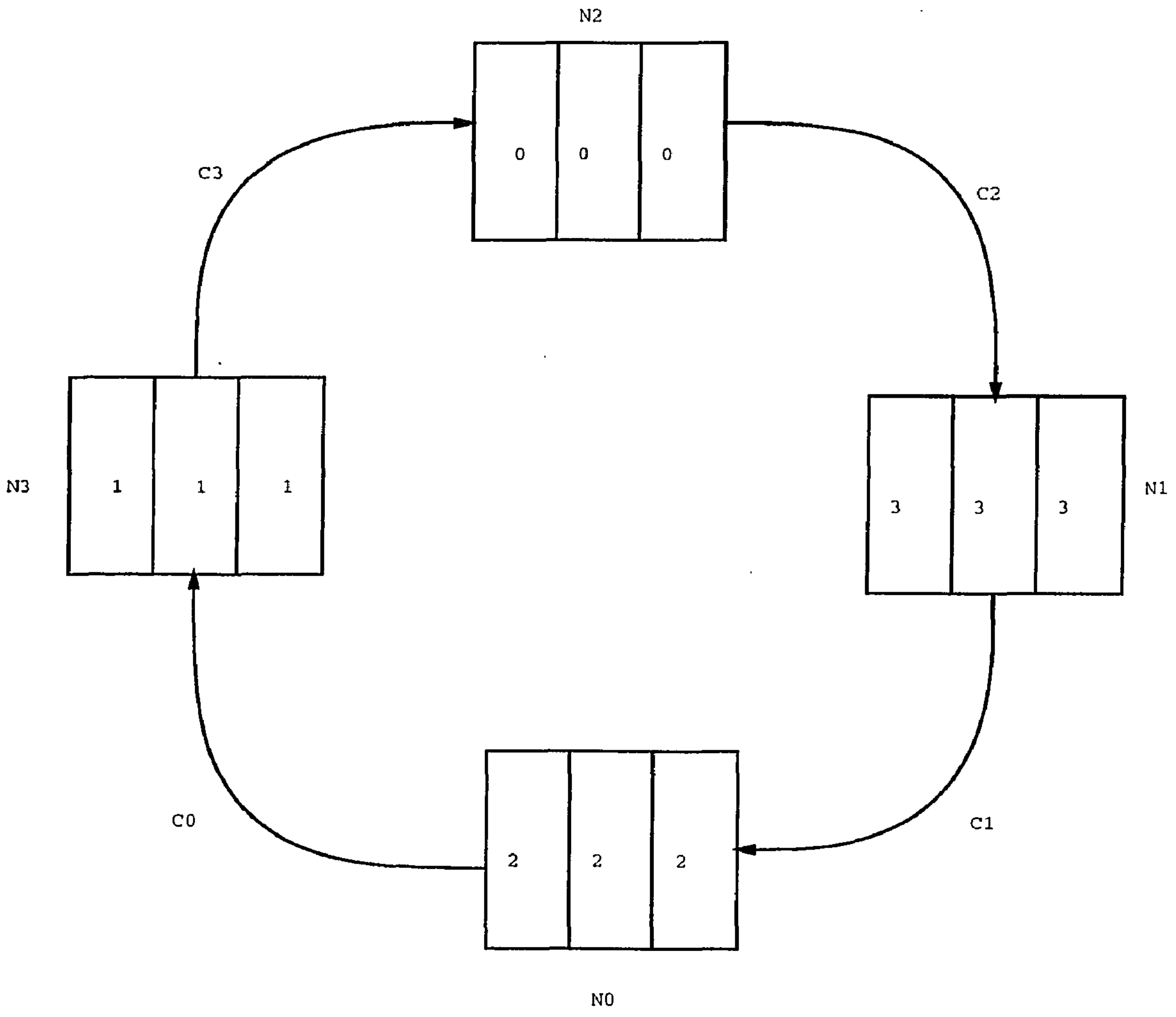


Figure 2.9: Deadlock in a 4-cycle

2.6 Routing Algorithm

Routing determines an output channel (path) that leads to the destination of the packet. A good routing algorithm should be easily implementable in hardware. Routing algorithms can be classified as deterministic or adaptive. In deterministic routing, routing decisions are made based only on the address of source and destination information. Deterministic routing is also known as oblivious routing. In adaptive routing, routing decisions are made on information that includes address of source and destination, network traffic load and faulty channels.

There are several well known deterministic routing algorithms. One of them is digit routing used in the delta network. In digit routing, the routing decision is made simply by looking at a portion of routing information in the packet. Most networks that belong to Multistage Interconnection Networks use digit routing. Figure 2.10 shows an example of digit routing in a baseline network.

Dimension-ordered routing is used in n-cube and mesh networks. The e-cube routing is one branch of dimension-ordered routing and is used for the binary hypercube [36]. In e-cube routing with N nodes, each node in the cube is represented by n digits ($n = log_2 N$). Each digit corresponds to one output channel. When a source node, s, needs to send a packet to a destination node, d, the routing tag, r, is computed by $r = s \oplus d$ where $\oplus$ is XOR. Then, the packet is routed through i-th output channel where i is the position of the first occurrence of one starting from the left most position of r. If there is no such occurrence, the packet reached its destination. After computing i, the i-th bit in r is zeroed before it gets forwarded. The e-cube routing is deadlock-free routing because it forces monotonic

Routing information (r) of Omega Network = destination

Thus, for source node of 3 and destination of 6,

routing information, r = 110

a node in i-th stage uses i-th digit from most significant bit of r.

If i-th digit of r is 0, then the packet is routed through upper link,
else it is routed through lower link.

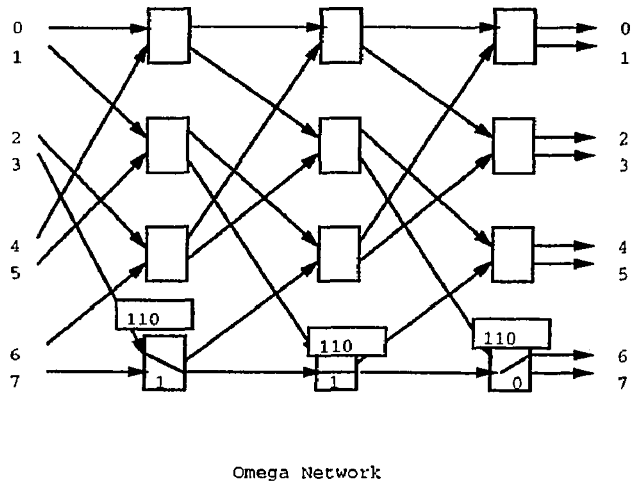


Figure 2.10: An Example of Digit Routing

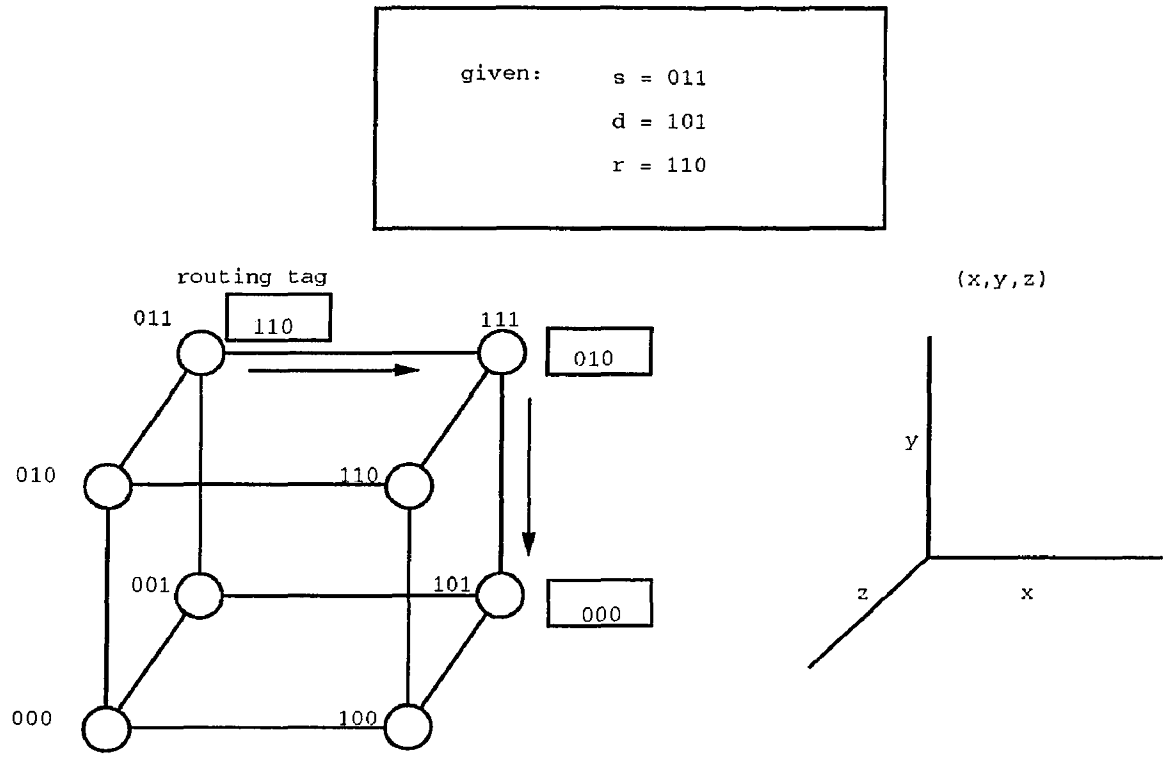


Figure 2.11: An Example of e-cube Routing in Hypercube

order on the dimensions traversed. A routing example on a hypercube is shown in Figure 2.11.

Another branch of dimension-ordered routing is 2D mesh routing. In a 2D mesh, each node is represented by its position (x, y) in X and then the Y dimension. A packet is routed in the X dimension first and then the Y dimension (alternatively the Y dimension first and the X dimension next). For a given source node, s, and a destination node, d, there are two approaches to represent the routing tag, $r = (x, y)$. The first approach is to use the destination address directly as the routing tag. In this case, the value x of the routing tag is compared to the value x of the current node. If they are equal, then the packet is routed to the Y dimension. Otherwise, the packet is forwarded in the same direction from which it came. The same is done in the Y direction until the y value of the routing tag matches the y value of the current node, which is the destination node. In the second approach, the routing tag carries relative distance from the source to the destination node of X and Y dimension. In this case, the value x of the routing tag is examined to see if it is equal to 0. If it is, the packet is routed to the Y dimension. Otherwise, the value x is decremented by one and the packet is forwarded in the same direction. The same process is repeated in the Y dimension until the value y becomes 0. Then, the packet has arrived at the destination node. Like the e-cube routing, the 2D mesh routing is also deadlock-free by keeping monotonic order in its routing.

The k-ary n-cube routing is a generalization of the e-cube routing algorithm. Dally and Seitz investigated deadlock-free routing algorithms, using virtual channels, for k-ary n-cubes as well as the Cube-Connected-Cycle and the Shuffle-Exchange Network [10]. Each node of a k-ary n-cube is identified by an n-digit radix k number. Each channel between two nodes is divided into an upper and lower virtual channel. The packet is routed on the high channel if the ith digit of the destination address is greater than the ith digit of the present node's address. Otherwise, the message is routed on the low channel. The k-ary n-cube with virtual channel is a deadlock-free routing algorithm because the packet is routed in order of descending subscripts.

There are numerous types of networks that belong to tree networks; thus, the routing algorithm for a tree network is another important class of deterministic routing algorithms. The tree networks include Binary Tree [22], Fat Tree [23], Flip

Tree [24], Hypertree [25] and KYKLOS network [26]. Among those, the routing algorithm for the Binary Tree is common to most of the tree networks. The basic concept of Binary Tree's routing algorithm is simple. The packet is routed up to the parent node if the destination node does not belong to the subtree of the present node. If the destination node is one of the nodes in the subtree, the packet is routed to the right child if it belongs to the right subtree of the present node. Otherwise, the packet is routed to the left child. It can be formalized as:

Let $i = (x_{p-1}, ..., x_1 x_0)$ be a node that has a packet for node $j = (y_{q-1}, ..., y_1 y_0)$, where

$$i = \sum_{h=0}^{p-1} x_h 2^h$$

and similarly for j. Then, j is in the left (right) subtree of i if and only if the following three conditions are all satisfied.

1) $p < q$,

2) $(x_{p-1}, ..., x_1 x_0) = (y_{q-1}, ..., y_{q-p})$,

3) $y_{q-p-1} = 0$ $(y_{q-p-1} = 1)$. [22]

The comparison of $(x_{p-1}, ..., x_1 x_0)$ and $(y_{q-1}, ..., y_{q-p})$ requires a shifter and a comparator. Horowitz and Zorat proposed an address numbering scheme called the even-odd numbering scheme that eliminates shifting of the address. Under the even-odd scheme, the second condition above can be replaced with $(x_{p-1}, ..., x_1 x_0)$ $= (y_{q-1}, ..., y_1 y_0)$. Figure 2.12 shows an example of even-odd numbering scheme. In [11], various routing algorithms including cube, mesh, ring, tree and MIN networks are summarized.

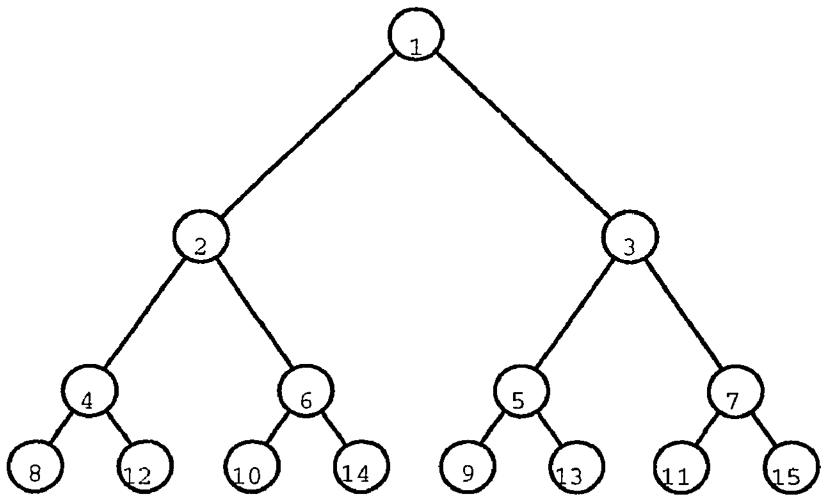
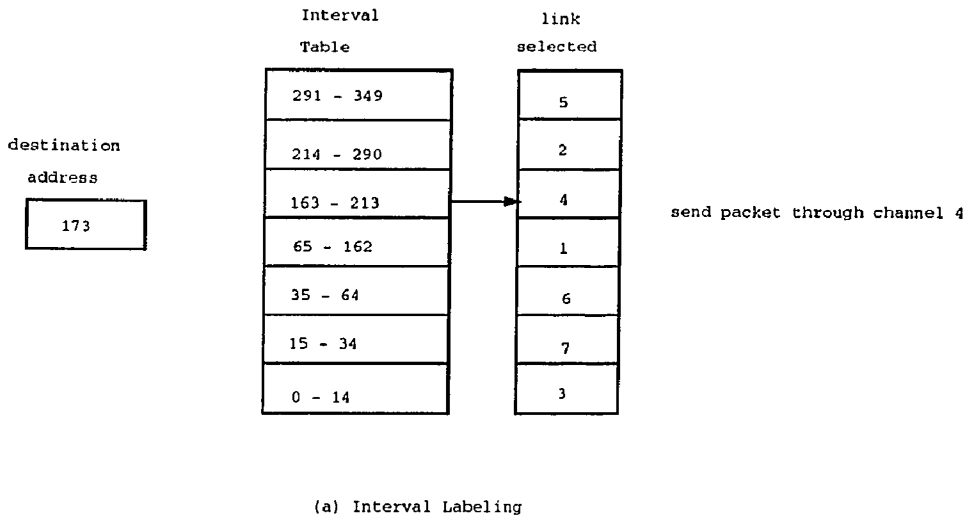


Figure 2.12: A Binary Tree with Even-odd Numbering

While most of the deterministic routing algorithms are simpler to implement than adaptive routing algorithms, they do not have the capability of dealing with dynamic conditions of a network such as faulty channels and/or traffic congestion. Incorporating some degree of intelligence to deal with dynamic network conditions, the adaptive routing algorithms can perform better in terms of throughput, latency and reliability of packet delivery. However, the adaptive routing often requires additional channels to avoid deadlock. Adaptive routing can be categorized into minimal adaptive routing and non-minimal adaptive routing. In minimal adaptive routing, packets are delivered to destinations through the shortest path. Supporting minimal adaptive routing requires additional channels and the number of additional channels increases rapidly as the size of the network grows larger [37]. In non-minimal routing, packets are routed through longer paths if the shortest path is not available. Non-minimal, deadlock-free, adaptive routing requires fewer additional channels than minimal adaptive routing, as shown in non-minimal adaptive routing on the k-ary n-cube proposed by Dally and Akoi [38].

The deterministic and adaptive routing algorithms discussed above support a fixed network topology. However, it is possible that networks are reconfigured. Currently available products that make it possible are Intel/CMU's iWarp cells or the Transputer IMS T9000 family. In this case, the router should have the capability to support multiple network topologies. In general, source routing and table-lookup routing schemes are used.

In the source routing scheme, the sending node prepares a header that has specific routing paths that lead to the destination. The header in the source routing is lengthy because it has complete routing information within it. Such an approach increases the network traffic by having longer messages. A more practical routing scheme for reconfigurable networks is the use of table-lookup. The basic concept of table-lookup routing is to have an entry in a table for each node address. When a packet comes into a router, the packet's destination address is used as an index to find an entry in the table where the output channel number used is specified. However, keeping an entry for each node increases the size of the table as the number of nodes in the network increases and, for large numbers of nodes, the cost of the table may not be practical. One method that can keep the size of the table small is to assign node addresses in such a way that a range of addresses uses a particular output channel. As long as the ranges of each output channel are mutually exclusive, this method will work and can reduce the table size. This method is called interval routing and Figure 2.13 shows an example of interval labeling and routing. In the example of interval labeling, there are seven different



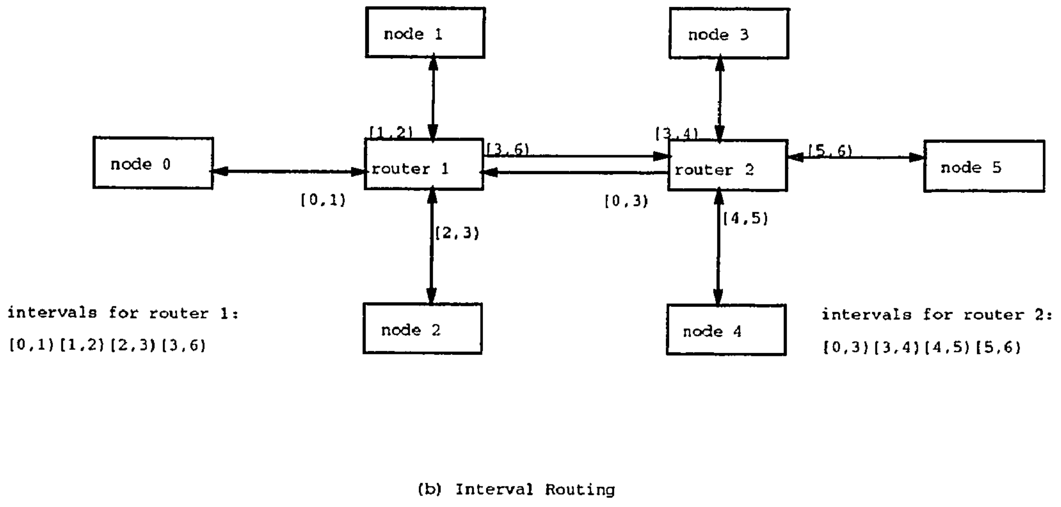


Figure 2.13: An Example of Interval Labeling and Routing

address ranges which span from address 0 to 349. There may be 350 nodes or less, depending on the network topology and the labeling scheme used. In this example, the destination address 174 falls into the range which is associated with output channel number 4 and the packet is routed through output channel 4. In the interval routing example, it shows how six nodes are labeled and how each router makes up an address range for each output channel.

2.7 Crossbar Switch

The crossbar switch is a key component of a switching element. It provides arbitrary connections between n input ports and n output ports. A multicomputer of n processing nodes can be built with a single n x n crossbar switch. Theoretically, throughput is maximized and latency is minimized if the entire network is one large crossbar switch[39]. However, the complexity of hardware increases rapidly as the number of processing nodes grows. In the early days, 2 x 2 or 4 x 4 crossbar switches were the most widely implemented sizes. Recently, with advances in VLSI technology, larger crossbar switches are becoming practical as can be seen in the Inmos T9000, where 32 x 32 crossbar switches were used, and Choi [40] shows a practical implementation of a 256 x 256 crossbar switch. The crossbar switch is composed of buses that connect I/O ports and an arbiter. Figure 2.14 shows a schematic diagram of a crossbar switch. An arbiter manages allocation/deallocation of bus connections between I/O ports. Arbitration of crossbar switches is different depending on the buffer management policy. Two basic arbitration schemes are the n-user 1-server arbiter and the m-user b-server arbiter. In a conventional n x n FIFO switch, the arbiter receives n requests for use of one bus and grants only

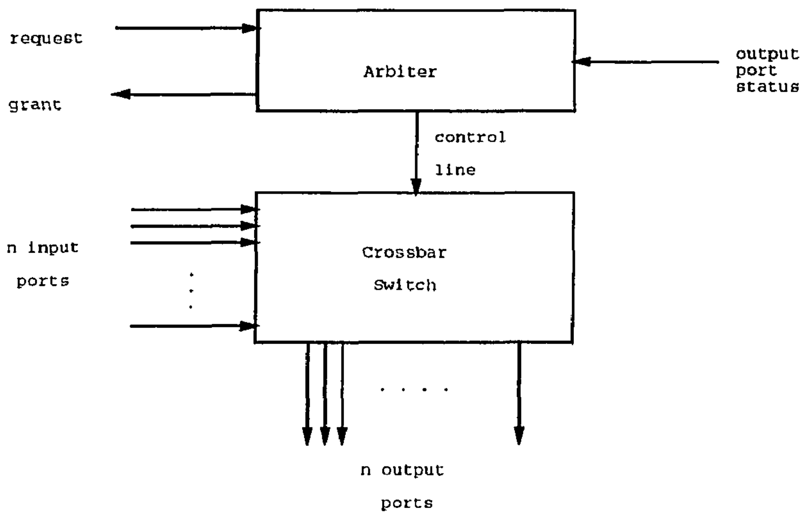


Figure 2.14: Schematic Diagram of A Crossbar Switch

one request. This type of arbiter is called n-user 1-server arbiter and has been well studied [41]. The arbitration in a switch with a multi-queue buffer is more complicated than a conventional n x n FIFO. In the switch where n input ports are connected to m output ports through b buses, the input port has to acquire two resources to be able to transmit: a bus connection from the input port to a crossbar and an output port connection through the crossbar. Since each input port requests one particular output port and an output port can grant only one request at a time, this can be implemented by an m user 1-server arbiter. All the output ports for which there are requests have to contend for the available the b buses. This arbitration can be done by an m-user b-server arbiter as described in [42].

2.8 Summary

In this survey chapter, major issues that are related to routers were presented. First, the network topology was surveyed. Among the network topologies, the regular topology was discussed, which was divided into two types: static and dynamic. The static network topology was classified according to its dimensions required for the layout. The important static network topologies included were mesh, hypercube and tree networks. The dynamic network topology had three classes: single stage, MIN and crossbar. The MIN is the most widely used network because of its flexibility and cost-effectiveness. Configuring a ring network based on the baseline network was given as an example of MIN's flexibility. Then, four types of switching techniques were surveyed: circuit, store-and-forward, virtual cut-through and wormhole. The circuit switching is suitable for bulk data transfer. The rest of the techniques are more appropriate for small-size data transfers. The store-and-forward technique was used in the early multicomputers. This technique had long network latency because it receives a complete packet at each node before it forwards the packet to the next node. The virtual cut-through technique was introduced in early 1980. In virtual cut-through, a packet was allowed to go to the next node before the whole packet was received. This technique greatly reduced the network latency. Both the store-and-forward and virtual cut-through require large buffer space for storing blocked packets. The wormhole routing technique was introduced in late 1980 and requires only small amounts of buffers with simple routing logic. The wormhole routing technique reduced network latency with low hardware overhead through its simple design and control technique, and became one of the most popular routing techniques. Flow control schemes were surveyed following the switching techniques. Well known FIFO buffer management schemes have an advantage in terms of simplicity of hardware, but letting only the packet at the top of the queue be transferred decreases the network bandwidth. To overcome this difficulty, three other buffer management polices, namely SAFC, SAMQ and DAMQ, were used. In these schemes, any packet in the buffer is given a chance for transmission. Among those schemes, the DAMQ is known as the best policy in terms of cost/performance trade-off. The virtual channel was surveyed next. The virtual channel increased the overall performance of networks by dividing a physical channel into multiple logical channels. In addition, the virtual channel was necessary in most of k-ary n-cube networks for constructing deadlock-free routing algorithms. Following the virtual channel, deadlock was briefly surveyed where it was pointed out that deadlock is commonly avoided by the routing algorithm. Routing was the next topic. Two classes of routing algorithms, deterministic and adaptive, were explained and examples of deterministic routing algorithms were examined. Furthermore, routing algorithms for reconfigurable networks were presented with an example of interval labeling and routing. Finally, issues around the crossbar switch were surveyed as the last topic of this chapter. As a summary of this chapter, Table 2.1 shows routers that were used in multicomputer systems along with their key design parameters.

Table 2.1: Routers and Types

System	*Switching Technique*	*Flow Control*	*Virtual Channel*	*Routing Algorithm*	Ref
iPSC-1	store-and-forward	FIFO	No	determin.	[43]
iPSC-2	circuit	-	No	determin.	[43]
Torus	wormhole	-	Yes	determin.	[10]
Chaos	wormhole	-	No	adaptive	[44]
Harts	virt. cut-thro.	FIFO	No	determin.	[45]
Ncube-1	store-and-forward	FIFO	No	determin.	[7]
Ncube-2	wormhole	-	No	determin.	[7]
Cosmic Cube	wormhole	-	No	determin.	[4]
Ametek 2010	wormhole	-	No	determin.	[46]
iWarp	wormhole	-	No	source	[47]
J-Machine	wormhole	-	Yes	determin.	[48]
CommCobb	virt. cut-thro.	DAMQ	No	table	[8]
BBN Butterfly	store-and-forward	FIFO	No	determin.	[16]
Touchs. Delta	wormhole	FIFO	No	determin.	[49]
Inmos T9000	wormhole	-	Yes	interval	[6]

CHAPTER 3

FLEXIBLE OBLIVIOUS ROUTER ARCHITECTURE

In this chapter we present a router architecture that accommodates a family of oblivious routing algorithms. The architecture is suitable for current technologies and is intended for multiprocessor and massively parallel systems. In the proposed architecture, we suggest that general purpose routers can be designed that accommodate a variety of multiprocessor interconnection networks. In particular, the routing algorithms of the interconnection structures that can be classified as trees, cubes, meshes and multistage interconnection networks can be accommodated with a flexible, simple to implement architecture. Our investigation strongly suggests that a common design can satisfy at least 40 network topologies with the introduction of few, very simple to implement instructions. The overall conclusion is that general purpose cost effective routers can potentially be designed that perform equally well as customized routing logic suggesting the possibility of a common router for multiple interconnection networks. Furthermore, the proposed architecture provides programming capabilities that allow other oblivious routing algorithms not considered in our investigation to be accommodated.

The remainder of the discussion is organized as follows. In section two, the background and the direction of our investigation is briefly explained. In section three, the flexible router architecture is introduced with examples of routing programs using the proposed router architecture. Section four reports various program characteristics. And, section five concludes the chapter with some remarks.

3.1 Routing Algorithms

In our investigation, we have considered oblivious routing algorithms for the determination of a general purpose router architecture. An oblivious routing scheme always produces the same communication path given the same source and destination address. The architecture is developed around topologies that have been extensively used in the design of parallel systems. The routing algorithms we have considered have been divided into five families denoted as: tree, cube, mesh, multistage interconnection networks and others (i.e. networks that did not fall into any of preceding four types of network topologies). We considered for direct implementation of the functions required by the routing algorithms of interconnection networks classified as trees, 6 classified as cube network, 5 as mesh, 10 as multistage interconnection networks and 8 that were not classified with the other network families. The major consideration of the investigation was to have simple to implement instructions and keep the number of instructions in the instruction set as small as practicable. Furthermore, we were interested in providing an architecture that allows parallelism in its implementation. For the 40 interconnection networks we considered, when possible, we considered optimal routing algorithms. Consequently we chose a set of instructions that will perform the algorithmic requirements. The entire study is rather lengthy to report here, and the interested reader is referred to [11] for a detailed discussion.

Identifying the algorithmic requirements is part of developing an architecture.

This architecture must support other functions necessary for hand shaking with other units. We support these capabilities with additional functions and protocols described in detail later.

3.2 Flexible Router Architecture

Generally speaking, the router consists of three major parts: n-input controllers, an n-input by n-output switching mechanism and n-output controllers. The input controller receives packets, performs a routing algorithm based on the routing information in the packet and determines the output controller through which packets are forwarded to the neighboring router. The n-input by n-output switch connects n-input controllers with n-output controllers. The output controller sends packets to the paired input controller of neighboring router. Figure 3.1 shows the logical structure of a router.

For a router to be flexible, the router should be able to execute routing algorithms of multiple network topologies. Therefore, a flexible router should provide means to implement all the required functions and/or instructions for various routing algorithms. All routing algorithms are executed by the input controller. Thus, the same n x n switch and output controllers can be used regardless of the input controller's flexibility. Developing a flexible router means providing capabilities for the input controller to execute multiple routing algorithms.

The logical function of the input controller can be broken into three major blocks: input port, routing algorithm handler, and packet flow controller as shown in Figure 3.2. The input port (incorporated in the port controller) is responsible for carrying out communication protocol for the reception of packets from the output

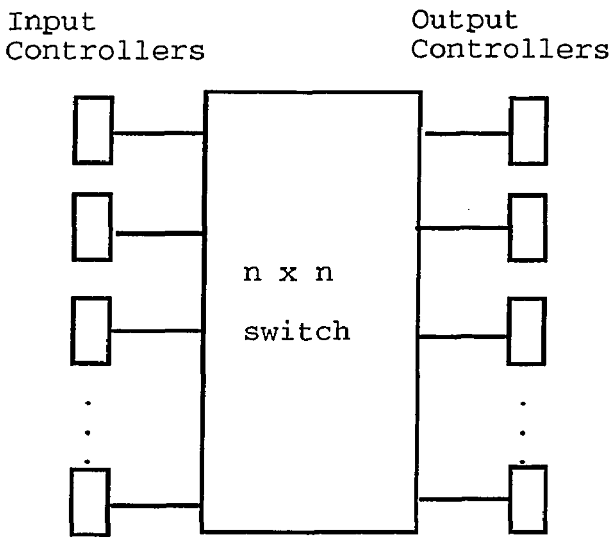


Figure 3.1: Logical Structure of a Router

controller. The input port extracts the header portion of the packet and transfers it to the routing algorithm handler. The input port also receives/transfers the data bytes to the packet flow controller. Upon receiving header information, the routing algorithm handler executes the routing program on the header information and sends the result (output controller number) to both the n x n switch arbiter and also to the packet flow controller. The packet flow controller stores the data bytes sent from the input port into its buffer and waits until the routing algorithm handler determines to which output controller the data bytes should be routed. Depending on the method of assigning buffers of the packet flow controller to the data bytes, there are three well known flow control schemes, namely store-and-forward, virtual cut-through [9] and wormhole [7]. Once the packet flow controller is notified of the output controller through which the data bytes should be forwarded, it waits

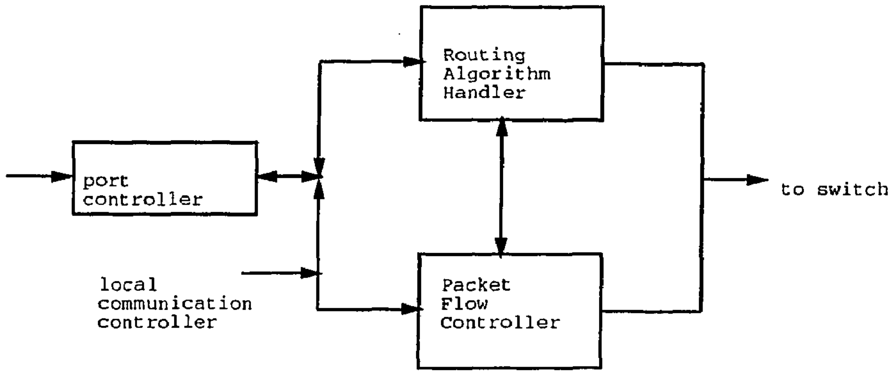


Figure 3.2: Logical Structure of an Input Controller

for the connection to the switch. When it receives the acknowledgment from the arbiter of the n x n switch, it transmits the data bytes to the output controller.

A necessary condition for being a flexible router is to have a routing algorithm handler that can execute multiple routing algorithms. In the next section, we propose a novel routing algorithm handler architecture that provides support for all the required instructions and manipulations of data to manage multiple routing algorithms. The architecture of the port/packet flow controller is reported elsewhere [13],[70]. When necessary, the functions and protocols of the port controller are discussed.

3.2.1 Routing Algorithm Handler Architecture

To directly support the algorithms, we have identified 12 simple to implement general purpose instructions shown in Table 3.1 that satisfy the requirements. The particular instructions required by the routing algorithm families of interconnection networks are reported in Table 3.2. In this table, the OUT, CMP and BC instruc-

ALU Instructions		Format	Operations
ADD	R1, R2, R3	RR	R3 = R1 + R2
SUB	R1, R2, R3	RR	R3 = R1 - R2
CMP	R1, R2	RR	Compare R2 to R1 (sets cond. code)
AND	R1, R2, R3	RR	R3 = R1 AND R2
XOR	R1, R2, R3	RR	R3 = R1 XOR R2
PLO	R1, R2	RR	R2 = pos. of leading one bit in R1
shift instructions			
SHR	R1, R2, R3	RR	R3 = Shift Right R1 by (R2)
SHL	R1, R2, R3	RR	R3 = Shift Left R1 by (R2)
data transfer inst.			
MOV	R1, R2	RR	R1 = R2
control instructions			
BC	address	MI	Branch on Condition
OUT	channel no.	I/R	End of program
coommnunication inst.			
MSG	R1,R2,R3	RR	Send message to local processor

R# represents a register and its number.

(R2) means the contents of register R2.

Table 3.1: General Instruction Set

tions are not listed because they are used in all networks. A detailed description of all the algorithms, the types of interconnection networks and the routing programs can be found in [11].

General Instruction Set

The general instruction set is used to execute routing algorithms. All operands are either stored in the registers or are made available within the immediate field of the instruction. Most of the arithmetic and logical instructions have three operand fields to improve the register pressure and help reduce the number

Network name	Instructions required	reference
Binary Tree	AND PLO XOR	[22]
Fat Tree	AND PLO XOR	[23]
Flip Tree	AND PLO XOR MOV	[24]
Bin. Tree with a Full Ring	AND PLO XOR MOV SUB SHIFT ADD	[25]
Bin. Tree with a Half Ring	AND PLO XOR SUB SHIFT	[25]
Hierachical Mesh	AND SUB ADD	[52]
Hypertree	AND PLO XOR ADD SHIFT SUB	[25]
Diamond Network	AND PLO XOR	[53]
KYKLOS Structure	AND MOV PLO SUB SHIFT XOR	[26]
Tree of Meshes	AND XOR PLO SUB	[54]
Quad Tree	AND XOR PLO ADD SHIFT	[69]
Hypercube	AND PLO SHIFT XOR	[55]
Folded Hypercube	AND PLO SHIFT ADD	[66]
Banyan Hypercube	AND XOR PLO	[56]
Spanning Multiaccess Channel	AND	[57]
Base-m n-Cube	AND SHIFT	[58]
Cube-Connected Cycles	PLO	[59]
Mesh Array		[59]
Torus Network	AND SUB	[2]
K-ary N-cube	AND SUB	[51]
Hexagonal Mesh	AND SUB	[45]
GNNM Hypercube	AND SUB	[60]
Omega Network	AND SHIFT	[28]
Delta Network	AND SHIFT	[28]
Baseline Network	AND SHIFT	[29]
Benes Network	AND SHIFT	[61]
Shuffle Exchange Network	AND SHIFT XOR	[10]
Augmented Data Manipulator Net	AND SHIFT	[62]
Generalized Cube Network	AND SHIFT	[28]
Extra Stage Network	AND SHIFT	[63]
Rectangulart SW Banyan Network	AND SHIFT	[64]
Gamma Network	AND SHIFT	[65]
Ring Network	AND SUB	[11]
Completely Conn. Net.		[67]
Pyramid Network	AND ADD	[28]
Chordal Ring Network	SHIFT	[50]
Crossbar		[68]
Cube-Conn Cycles w/ virt. chan.	AND SHIFT XOR MOV ADD	[10]
K-ary n-cube /w virtual channel	AND SHIFT XOR MOV ADD	[10]
Shuffle Exchange Net. w/v.c.	AND SHIFT XOR MOV ADD	[10]

Table 3.2: Instructions Required for Interconnection Networks

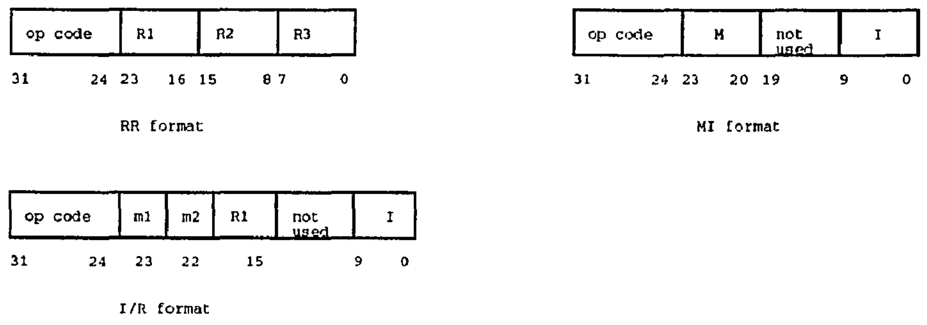


Figure 3.3: Three Instruction Formats

of data transfers between registers. All instructions have equal length which is assumed to be 32 bits long. Each instruction has one of three formats: register to register (RR), immediate or register (I/R), or immediate with mask (MI). In the RR format, all operands are in registers. In the MI format, the I is the absolute address and M is the mask value. In the I/R format, there are two mode bits, m1 and m2. The m1 indicates if the R1 is used or the I is used for the OUT instruction only. The m2 is used for the instruction ECP and it indicates if it has the I as its operand or no operand. Figure 3.3 shows the three instruction formats with the assigned bit positions.

The description and some more information regarding the instructions can be found in Table 3.1. The specific definition of the instructions is reported in [11]. The mnemonics and the functions of most instructions are self explanatory except the instruction PLO (find Position of Leading One bit). The PLO instruction is used frequently in the routing programs of the tree and cube networks. As the name implies, it finds the position of leading one bit starting from the most significant bit position down to the least significant position and leaves the result of positional value in the target register. As an example, the instruction:

PLO R1,R2 (R1 = 00100000 and R1 is an 8 bit register labeled from 0 to 7)

will have the result (R2 = 00000101) because the leading one bit was in the 5th bit in R1.

Addressing: All instructions in a program (except BC) imply a sequential access of the program. The branching instruction is the only one that may use the address of memory to determine the instruction to be executed. For the simplicity of the architecture, there is only one type of addressing and that is absolute addressing. The absolute address is the address assigned to a memory location. An absolute address does not require any transformation of addresses when accessing memory.

Instructions and condition code: The condition code is set only by the CMP (compare) instruction and tested by the BC (branch on condition) instruction. Overflow conditions are ignored and not recorded anywhere. No other flag bits, such as "result equal to zero", are set as a result of arithmetic/logical, shift or the PLO instructions.

Control Instruction Set

The instruction set we have described in the previous section can be used to determine the behavior of the routing algorithm handler. No additional instructions are required for the design of a router. The router can be initialized with proper settings of its memory and states (to be discussed). The control instruction set is introduced primarily to perform functions such as initialization, cooperative operations with a local processor, potential operation mode that allows adaptive routing

Instructions		Format	Operation
LPG	R1,R2,R3	RR	Load program
LSR	R1,R2	RR	Load status register
LR	R1,R2	RR	Load general register
ECP	address	I/R	End of control program

Table 3.3: Control Instruction Set

which requires complex functions, etc.. We introduce the control instruction set to increase the flexibility of the router architecture. The control instruction set comprises 4 instructions as shown in Table 3.3. The Load ProGram (LPG) instruction initiates the transfer of the routing program from the local communication controller to the memory of the routing algorithm handler. LPG has three operands. The first operand, (R1), contains the address of where the routing program is stored in the local communication controller. The second operand, (R2), has the address of where the routing program should be loaded in the routing algorithm handler. The third operand, (R3), is the counter which specifies the size of the routing program to be loaded. The program loading operation is performed until the counter, (R3), is equal to zero. The Load Status Register (LSR) loads the new content into the status register. The Load Register (LR) puts the new value into one of the general registers of the routing algorithm handler. The End Control Program (ECP) instruction terminates the control program and returns the privileged state of the routing algorithm handler to the normal state and it may or may not set the instruction address depending on the value of the m2 field in the instruction. The description of how the control instruction is executed can be found in section 3.3.

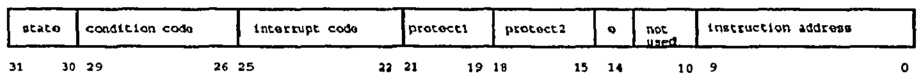


Figure 3.4: Status Register Format

Status Register

The status register in the routing algorithm handler keeps the information required for the execution of the active program and its implementation is always required. It includes the instruction address, the condition code, the interrupt code, the protects, execute bit(e) and the state of the routing algorithm handler. Figure 3.4 shows the bits assigned for each of the fields. The content of the status register is set by the control instruction LSR.

States: There are two states in the routing algorithm handler: the normal state and the privileged state. Each of these states has two modes, namely operating mode and idle mode. The control instructions are executed in the privileged state and the general instructions in the normal state. The interrupt is only executed when the routing algorithm handler is in the idle mode. After the completion of each routing program, the routing algorithm handler enters the idle mode. The mode of the routing algorithm handler changes from the stopped mode to the operating mode when the input port transfers header information to the registers of the routing algorithm handler and causes an end of the header packet interrupt. The state of the routing algorithm handler changes from the normal state to the privileged state when the local communication controller causes an execute control program interrupt. The MSG instruction can also change from the normal to privileged state.

Condition Code: The condition code is set by the result of the CMP (compare) instruction and recorded in the condition code field in the status register. The meanings of each bit in the field are:

condition code	
0	operands are equal
1	first operand is low
2	first operand is high
3	undefined

Interrupt Code: There are five types of interrupts in the routing algorithm handler. The interrupt code in the status register records types of interrupts as follows:

0000	hardware failure
0001	input port
0010	local communication controller
0011	program check
0100	Instruction not implemented

Protect Fields: As was shown in the instruction format, a routing algorithm handler can have up to 256 registers. In this architecture, we do not implement all registers. To avoid unnecessary hardware, once the number of registers in the routing algorithm handler for a specific technology is decided, the protect1 field is used to indicate the number of implemented registers. The protect1 field has 3 bits and it can represent numbers ranging from 0 to 7. If a register number in the instruction is given as $r_7 r_6 ... r_0$, a value i of the protect1 field represents that for all r_j, where $j \geq i$, are zeros. If not, the program check interrupt will occur. The protect2 field is used in a similar way, it determines the actual addressing space that a program can use and it sets the flag bit on if the address exceeds the address range implemented.

Execute Bit(e): Indicates if the execution occurs from the local memory of the routing algorithm handler or an external device.

Instruction Address: The instruction address field contains the address of the next instruction for either control instructions or general instructions. When the state bit in the status register indicates that the routing algorithm handler is in the normal state, the instruction address represents the next instruction address for general instructions. Otherwise, it represents the next instruction address for the control program.

Address Generation and Data Format: Execution of instructions by the routing algorithm handler involves generating addresses of instructions and operands. When an instruction is fetched from the location designated by the current status register, the instruction address is increased equally after execution of each instruction. For the branching instruction, the address of the next instruction is either the address of the next instruction in the sequence or the address specified in the I field in the instruction depending on the branching decision made in the branch instruction. All instructions treat data as only one type, two's complement numbers. In the two's complement numbers, the most significant bit is used as the sign bit indicator. The logical structure of the routing algorithm handler is shown in Figure 3.5.

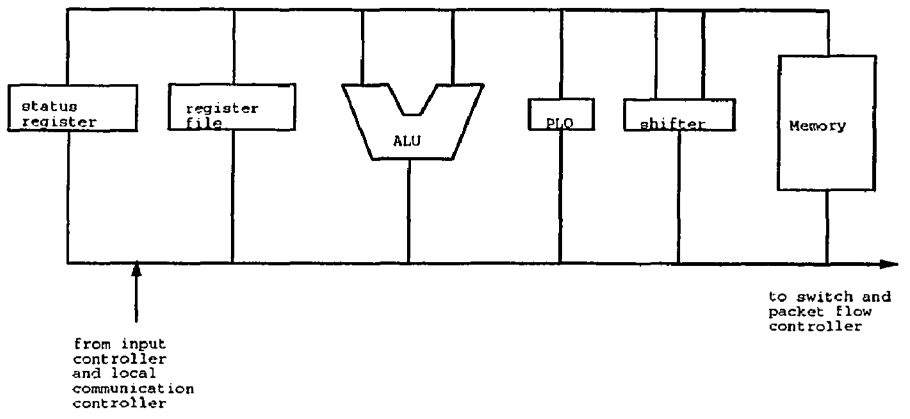

status register
register file
ALU
PLO
shifter
Memory
to switch and packet flow controller
from input controller and local communication controller

Figure 3.5: Conceptual Structure of Routing Algorithm Handler

3.2.2 Storage

Registers: There are up to 256 registers in total. For simplicity in program writing, we consider two types of registers. The first type of registers is the general registers. These registers store values or results of computations. The second type of registers is the constant registers. The constant registers hold constant values used in the routing program.

Memory: The routing program is loaded into memory and executed from the memory. The routing algorithm handler does not allow operands to be stored in memory. The word length of the memory is 32 bits.

Interrupts

The interrupt facility allows the routing algorithm handler to react to the hardware failures in the router, monitor the program execution status, initiate the routing program stored in the memory, and also communicate with the local processor through the local communication controller described later. We describe the following types of interrupts.

Interrupt from Input Port: When the input port receives a packet, it stores the header part of the packet into the predetermined register(s) in the routing algorithms handler. Then, it causes an interrupt to the routing algorithm handler that begins executing the routing program on the newly arrived header information. If the routing algorithm handler is in the normal state and the interrupt occurs, then the routing program executes. Otherwise, the interrupt will remain pending until the state changes to normal.

Interrupt from Local Communication Controller: The local communication controller communicates between the local processor and the input controllers. The local processor sends data, the routing program and the control program to the local communication controller. It also instructs the local communication controller to notify the input controller that the control instructions should be executed. The local communication controller does this operation by causing an interrupt to the input controller. When the interrupt occurs from the local communication controller, the input controller changes its state to the privileged state and executes control instructions stored in the local communication controller.

Interrupt from Program: An interrupt of a program check occurs when the register number exceeds the allowed number or an invalid instruction is detected. The routing algorithm handler sends the program check message to the local processor and halts the program.

Interrupt from Hardware: The bus error or hardware failure from any component can cause an interrupt. The routing algorithm handler reports the error to the local processor and halts the program.

3.2.3 The Local Communication Controller Mechanism

In this section, we describe the concept and the conceptual structure of the local communication controller. The facility need not be implemented if the control instruction set is not considered for implementation. This facility can be either implemented in hardware or in software. If the local communication controller is implemented, it is responsible for the communication between the local processor and the input controllers. The local processor sends programs and data to the input controllers via the local communication controller and vice versa. The communication between the local processor and the local communication controller is carried out through messages. The interrupt mechanism is used for the communication between the local processor and the input controllers. The local communication controller reserves spaces for the status register, the general registers, the constant registers, routing programs and control programs. Those spaces, except the control instruction space, are replicas of storages in each of the input controllers. Figure 3.6 shows reserved spaces in the local communication controllers. Even though the physical structure of the controller may not be necessary, its implementation may be highly desirable for performance reasons.

Communication between local processor and local communication controller:

When the processor needs to send data or routing programs to one or all of the input controllers, it first sends messages that contain data for the local communication controller. The data/instructions are stored in the reserved spaces. Fur-

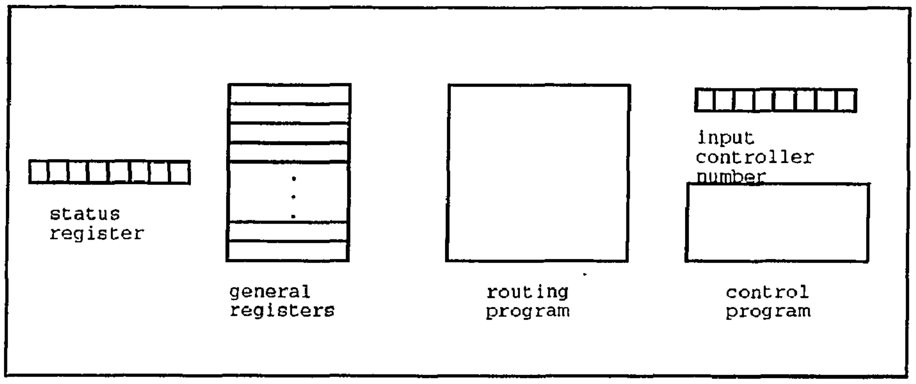


Figure 3.6: Reserved Spaces in Local Communication Controller

thermore, the input controller number and the control instructions are transferred to the communication controller. If it is desired that the routing algorithm handler will execute the routing program from the communication controller, then the status register reflects this correspondence. Otherwise, the control program has to be loaded into the routing algorithm handler. The control program is loaded at the beginning of the address space reserved for the control instructions. The last address of the control program is used by the local communication controller to modify the addressing of the routing program, if necessary, by a constant offset so that no conflict of the addresses occurs.

Communication between local communication controller and input controllers:

In all cases, the local communication controller causes an interrupt to notify the input controller, which receives data and/or the routing program, that the data/routing program has arrived. The interrupted input controller then puts its state to the privileged state and executes the control instructions stored in the local communication controller or executes the control instructions in its memory depending on the status bit reflecting where the control instructions reside. As indicated earlier, the starting address of the control instructions is always the beginning address space for both techniques. By executing control instructions, the input controller may load data into the status register and general registers. The routing programs are also loaded into their memory by control instructions.

3.3 The Operating Environment

Once the type of interconnection is decided for a parallel system, the routing program for the routing algorithm handler can be developed in either the host computer or the local processor. The executable program is downloaded to the routing algorithm handler in each input controller.

Routing Algorithm Handler Initialization and Downloading of Data: The registers of the routing algorithm may hold information that is used repeatedly in the routing program. An example of such information is the address of the source node where the router is attached. This information is needed frequently in many of the routing programs and it is known in advance, thus it does not need to be computed in the routing program. The other data needed to be downloaded are the content of the initial status registers in each the routing algorithm handlers. The value for the protect fields will be assigned appropriately depending on the number of registers to be supported. The initial instruction address for the routing program should be determined and set accordingly. The control program for initialization also has to be present in the host computer or the local processor. The control program needed for the initialization will be the sequence of LPG, LR, LSR and ECP if it is assumed that the control instruction set is implemented. Once all the necessary data are ready, the processor transfers the data to the local communication controller, assumed for simplicity here to be implemented in hardware. After sending all the data, the processor sends another message that indicates the end_of_data. Upon receiving the end_of_data message, the local communication controller causes an interrupt to the routing algorithm handler in the input controller. Then, the routing algorithm handler executes the control program loaded in the local communication controller and initializes the status register, general registers, and its memory. The last instruction in the control program is the ECP instruction. It will put the state of the routing algorithm handler into the normal state and stopped mode. And it waits for header information to begin operating. Multiple input controllers using this example scheme can be initialized with the same environment by changing the input controller number and repeating the process. A parallel initialization is possible with proper hardware support and parallel loading of the program to all input controllers.

Execution of the Routing Program: Once the routing algorithm handler is initialized, it will be in the normal state and the stopped mode. The mode changes from the stopped to operating state when the input port causes an interrupt that notifies the end of the header information transfer to the input controller. Then, the routing algorithm handler executes the routing program on the new header information. The last instruction in the routing program is the OUT instruction. It sends the result of the routing program to the n x n switch as well as to the packet flow controller. Then, it changes the operating mode to the stopped mode.

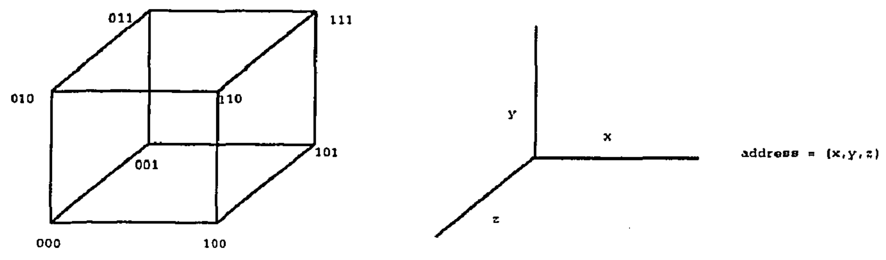


Figure 3.7: Hypercube Network

3.4 Routing Program Examples

In this section, we show examples of routing programs written using the routing algorithm handler instructions. In particular, the first example, describes the operation of a routing algorithm entirely supported by the router. In the second example, we demonstrate the flexibility of the proposed architecture by showing an example which requires both the processor and the router, operating in synergy. In the third example, we make an example of how to support reconfigurable topologies using the Inmos [6] table lookup scheme, for the interval labeling method with the proposed architecture.

Example 1: (Routing in Hypercube) Figure 3.7 shows a hypercube network. For the given pair of source, $s_{n-1}s_{n-2}...s_0$, and destination address, $d_{n-1}d_{n-2}...d_0$, the routing tag, $r_{n-1}r_{n-2}...r_0$, is computed as:

$$r_{n-1}r_{n-2}...r_0 = (s_{n-1}s_{n-2}...s_0 \; XOR \; d_{n-1}d_{n-2}...d_0)$$

Then, using this routing tag, each router performs the following algorithm:

if ($r_{n-1}r_{n-2}...r_0$ = all zeros) then forward the message to the local processor

else find the position, i, of the leading one bit in ($r_{n-1}r_{n-2}...r_0$)

set $r_i = 0$

send message through ith direction

The routing program developed in the routing algorithm handler that performs the hypercube routing algorithm outlined above is shown below. R1, C1 and C2 are general registers containing the routing tag(R1) and the constant values(C1 and C2). The values of C1 and C2 are set when the routing algorithm handler is initialized by the local processor. The value of R1 is loaded by the input port each time the input port transfers new header information. The output channels are numbered as following:

output channel in x direction = 1

output channel in y direction = 2

output channel in z direction = 3

output channel to the local processor = 4

Hypercube Routing Program

R1: Routing tag

C1: = 0

C2: = 1

```
1.              CMP   R1,C1
2.              BC    B'1000', processor    (branch if equal)
3.              PLO   R1,R2
4.              SHL   C2,R2,R3
5.              XOR   R3,R1,R1
6.              OUT   R2
7.  processor:  OUT   4
```

Register R1 has the routing tag and the CMP instruction at line 1 compares C1 to the routing tag. At line 2, the BC instruction tests if the routing tag contained all zeros. If it is true, the program branches to the location labeled processor and executes the OUT instruction where operand 4 indicates that the message should be forwarded to the local processor. If the test result at line 2 is not true, the program continues to search for the direction the message should go. The PLO instruction at line 3 does this job by finding the position of the leading one bit in R1 and storing the result in R2. Thus, the value, i, of R2 represents the direction of the next node. The SHL instruction at line 4 shifts register C2(=1), i bit positions to the left and stores the result in R3. The value in R3 is used to zero out the ith bit of the routing tag and it is done by XORing R1 and R3. Finally, the program sends the result of the routing program by OUT R2.

Example 2: (Synergetic Operation) As mentioned before, the instructions provided for the routing algorithm handler were selected carefully after investigating numerous network topologies. Yet, there are special cases where the routing algorithm should be executed in the local processor. This case may occur for example if the routing algorithm requires instructions not provided within the routing algorithm handler or if the size of the routing program is too big to be stored in the memory of the routing algorithm handler. The routing algorithm handler provides the MSG instruction through which it can delegate some or all of the routing program to the local processor. The MSG instruction has three operands, which are all registers. The first operand indicates the parameter to be sent to the local processor. One example of the parameter is the address of the destination node. The second and the third operands are pointers to the beginning and end of the sequence of registers that contain the address of memory locations in the local processor where the desired routing program is stored. When the MSG instruction is executed in the routing algorithm handler, it causes an interrupt to the port controller. Then, the port controller builds a message where it contains the parameter and the address of the memory location in the processor, along with the input controller number which has the routing algorithm handler executing the MSG instruction. After building the message, the port controller sends it to the local processor. The local processor will execute the desired program for the routing algorithm handler. Then, it sends the result to the port controller. The result includes the input controller number, the new content of the status register and the control program. Included in the status register is the new instruction address which the routing algorithm handler will fetch to execute the next instruction when control returns to it. The control program has two control instructions, LSR(load status register) and ECP(end control program). At this point, the port controller causes an interrupt to the routing algorithm handler which is in the input controller specified in the message from the processor. The routing algorithm handler then executes the control program stored in the port controller and has the new content of the status register. The next instruction address of the routing algorithm handler is the location specified in the instruction address. Figure 3.8 shows the sequence of communications that occur by the MSG instruction.

Example 3: (Routing of 4x3 mesh with interval labeling) The routing algorithm with interval labeling belongs to the table-lookup routing scheme. It is an efficient routing algorithm that reduces the table size [6, 71]. The architecture we propose can also support this scheme. We show a routing program example for

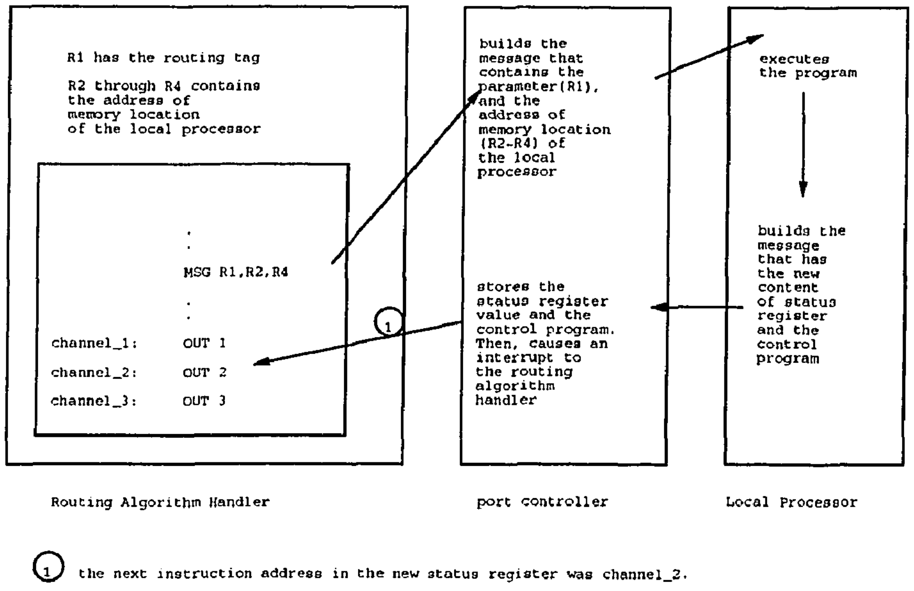

R1 has the routing tag
R2 through R4 contains the address of memory location of the local processor
MSG R1,R2,R4
channel_1: OUT 1
channel_2: OUT 2
channel_3: OUT 3
builds the message that contains the parameter(R1), and the address of memory location (R2-R4) of the local processor
stores the status register value and the control program. Then, causes an interrupt to the routing algorithm handler
executes the program
builds the message that has the new content of status register and the control program
Routing Algorithm Handler
port controller
Local Processor
1 the next instruction address in the new status register was channel_2.

Figure 3.8: Sequence of Communication Caused by the MSG Instruction

a 4x3 mesh interconnection network reported in [71] using the proposed routing algorithm handler. As shown in Figure 3.9, each node of the 4x3 mesh 2D is assigned a label $\phi(x, y)$. In particular, the example routing program will show how the routing is done for node (1,1). Let d be the label of the destination address in a packet. Each routing table requires only four entries. One for each outgoing channel. For example, the routing table at node (1,1) will contain the following information. For $d \geq 7$, the packet will be routed using the +Y channel. For $5 \leq d < 7$, $1 < d \leq 3$, and $d \leq 1$, the packet will be routed through channels -X, +X, and -Y, respectively. Following are the routing programs using the routing algorithm handler. It is assumed that 1,2,3,4 and 5 represent channel numbers of -X,+X,-Y,+Y and local processors respectively.

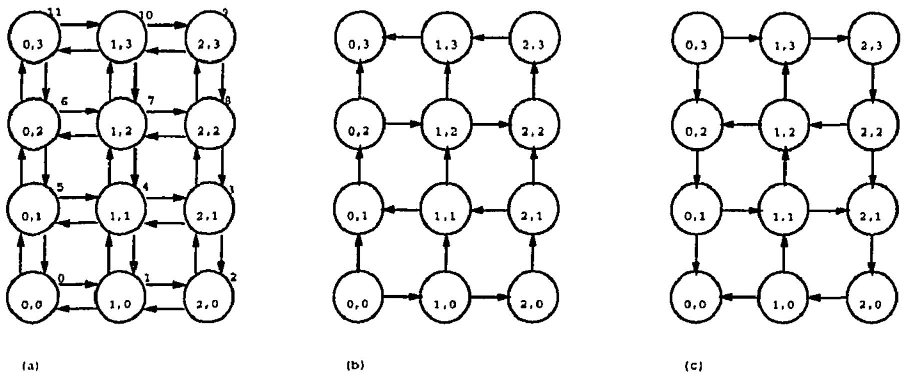


Figure 3.9: The Labeling of a 4x3 Mesh: (a) Physical Network; (b) High-Channel Network; (c) Low-Channel Network.

R1: Routing tag (d)

R2: Label of the current node

C1: = 0

C2: = 5

C3: = 0

```
1.            CMP   R1,R2
2.            BC    B'1000',processor    (branch if equal)
3.            CMP   R1,C3
4.            BC    B'1010',+Y           (branch if greater than or equal)
5.            CMP   R1,C2
6.            BC    B'1010',-X           (branch if greater than or equal)
7.            CMP   R1,C1
8.            BC    B'1100',-Y           (branch if less than or equal)
9.            OUT   2
10. +Y:       OUT   4
11. -X:       OUT   1
12. -Y:       OUT   3
13. processor: OUT  5
```

3.5 Program Characteristics

In this section, we discuss some of the program characteristics for the various routing programs we considered. This program has been reported elsewhere [11]. As they are shown in Figure 3.10, the CMP and the BC instructions are used in all routing programs because every routing program has to check to see whether or not the packets arrived at their final destination nodes. The OUT instruction is also used in every routing program since it sends the result of the routing decision made to the n x n switch as well as to the packet flow controller. Many routing algorithms need operations on the selected fields of the given header information. The AND instruction is used to mask out the unnecessary field of data. The PLO instruction is used in the routing program of all tree networks and some of the cube networks. The shift (SHR and SHL) instructions are used to align the data for comparisons and used in the routing programs of most of the multistage interconnection networks. The ADD and SUB instructions were used to increment or decrement values. The length of routing programs for the tree networks, the cube networks, the mesh networks, the multistage interconnection networks and the networks not classified as preceding families are shown in Figure 3.11 through 3.15. The actual length of routing programs may vary depending on the size of the network.

3.6 Conclusion

In this chapter, we have presented a router architecture that accommodates a family of oblivious routing algorithms. The architecture is suitable for current

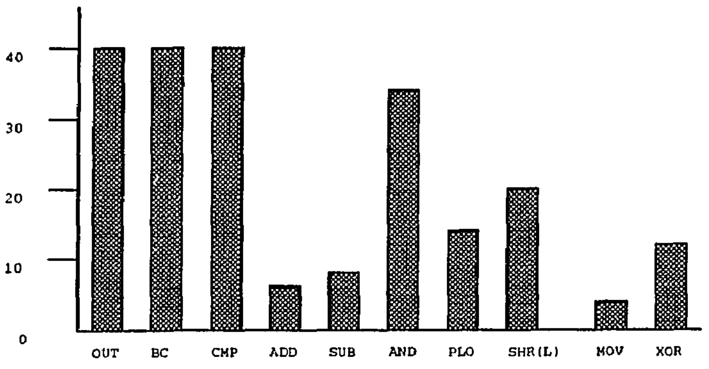


Figure 3.10: Frequency of Instruction Usage in the 40 Networks Shown in Table 3.1

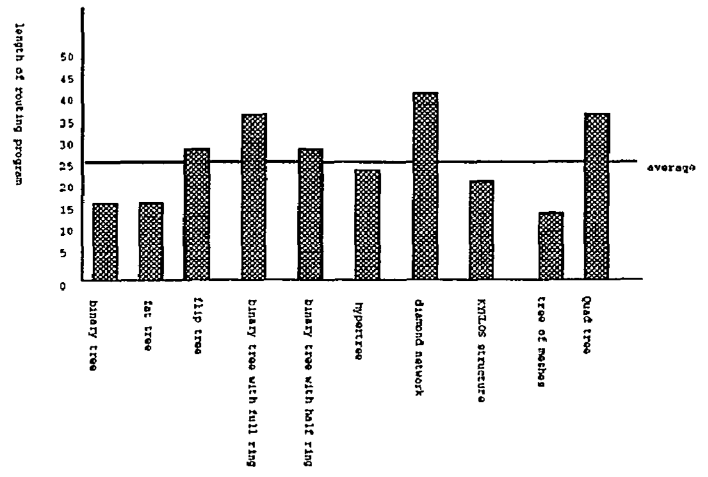


Figure 3.11: Routing Program Length for Tree Networks

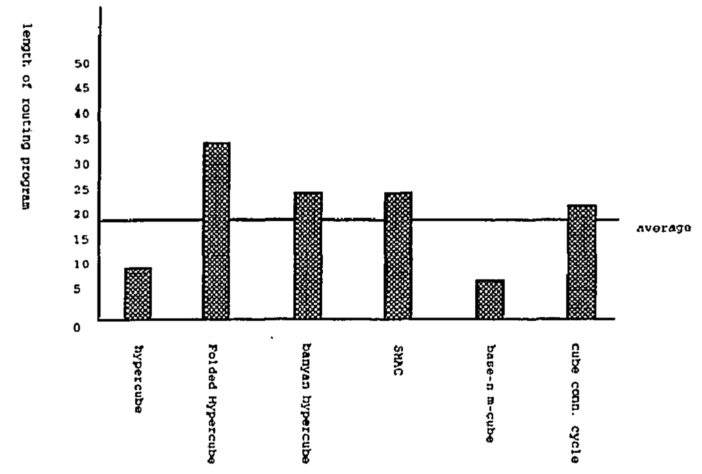


Figure 3.12: Routing Program Length for Cube Networks

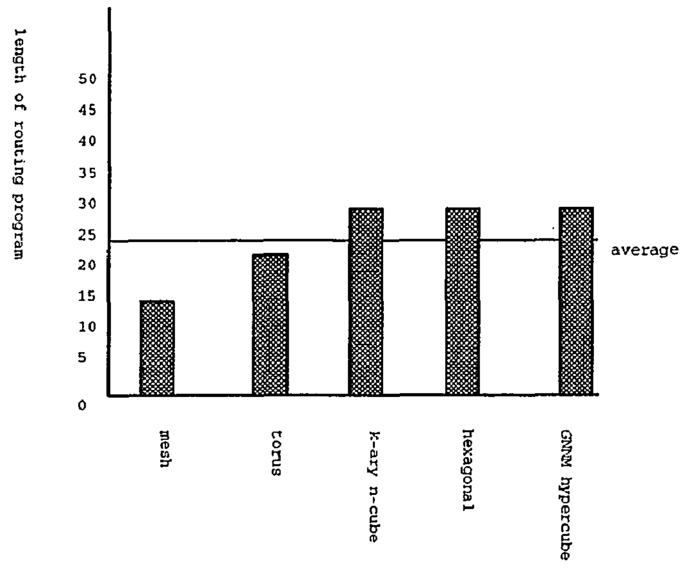


Figure 3.13: Routing Program Length for Mesh Networks

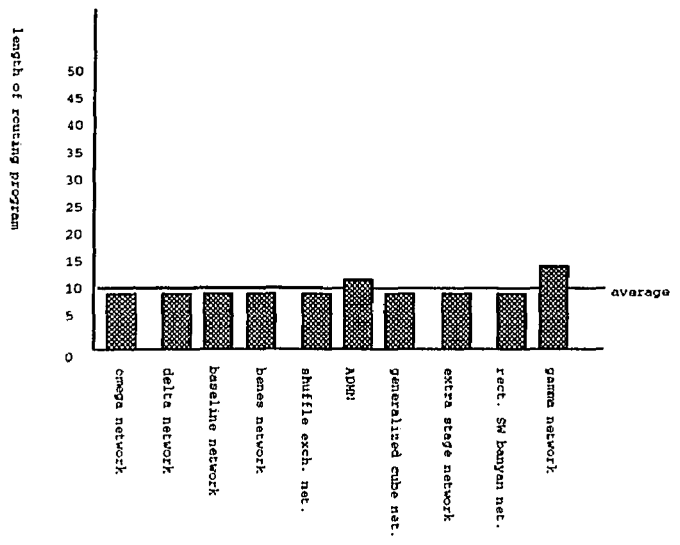


Figure 3.14: Routing Program Length for Multistage Interconnection Networks

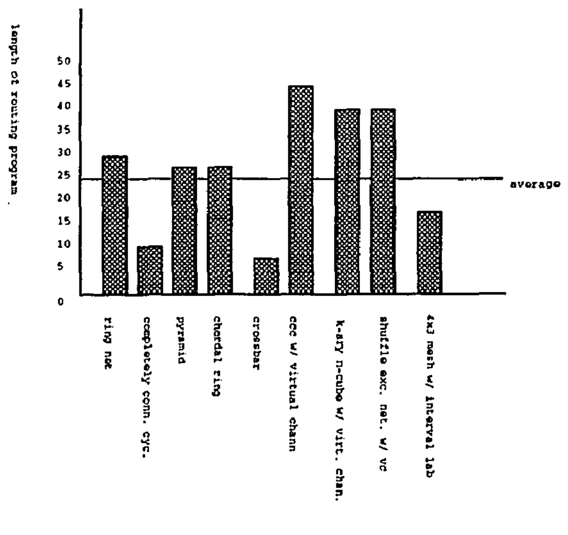


Figure 3.15: Routing Program Length for Other Networks

technologies and it is intended for multiprocessor and massively parallel systems. For this investigation, we studied the routing algorithms of over 40 interconnection networks. We have identified the common functions and the instruction set that satisfies the requirements for executing all the routing algorithms. Since the architecture provides programming capabilities, it allows other oblivious routing algorithms not considered in our investigation to be accommodated as well. Furthermore, the architecture can handle various types of header packet formats that are necessary to support different sizes of the interconnection networks. In addition, the fact that the architecture is programmable makes it easy to modify the routing algorithm if there are any errors in the routing algorithm or a better algorithm is developed later. Because the architecture supports a wide range of interconnection networks, it can be mass-produced and has the potential of being an "off the shelf" product. The overall conclusion is that general purpose cost effective routers can be designed suggesting the possibility of a common router for multiple interconnection networks.

CHAPTER 4

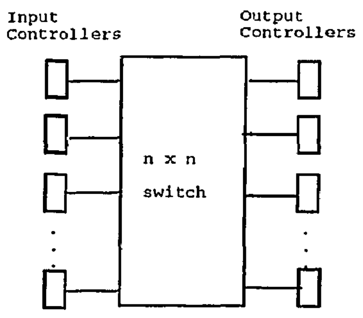
DESIGN AND EVALUATION OF A DAMQ MULTIPROCESSOR NETWORK ROUTER WITH SELF-COMPACTING BUFFERS

A broad class of multiprocessor interconnection networks can be constructed out of buffered routers that transfer packets from a set of inputs to a set of outputs. This study describes a new approach to implementing a high performance router using a technique called "self-compacting buffers". This technique is efficient in that the amount of hardware required to manage the buffers is relatively small; it offers high performance since it is an implementation of a Dynamically Allocated Multi-Queue (DAMQ). Multi-queues exploit more channel bandwidth than FIFO queues by allowing unblocked packets to bypass blocked packets. The first part of this chapter provides a detailed description of the self-compacting buffer architecture and compares it against a competing DAMQ buffer design. The self-compacting buffer technique offers comparable performance requiring less hardware. The second part presents extensive simulation results comparing the performance of a self-compacting buffer against an ideal buffer. The comparison extends previous work by considering a much broader range of network topologies, including several examples of k-ary n-cubes and delta networks. In all cases, the self-compacting buffer has performance comparable to an ideal buffer up to 80 % of the saturation bandwidth of the ideal buffer. In addition, simulation data show how the performance of an entire network can be quickly and accurately approximated by simulating just a single router.

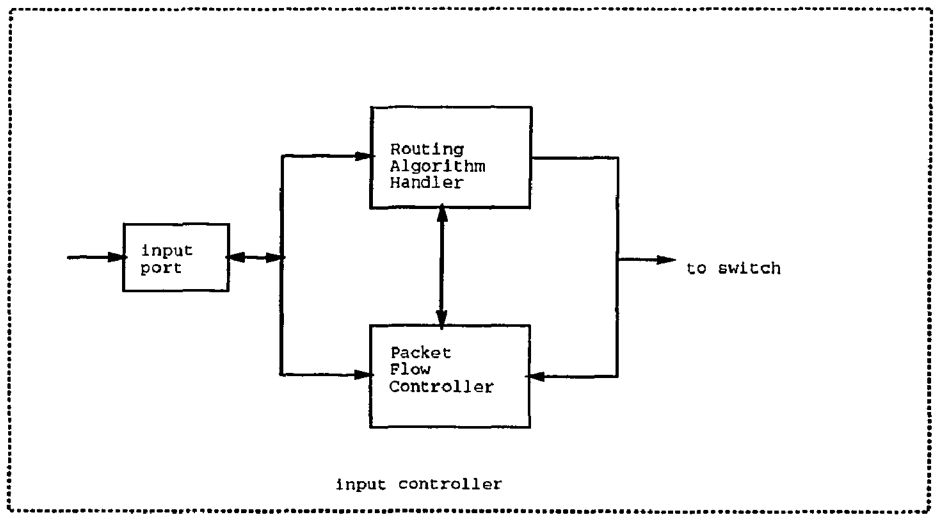
The remainder of the chapter is organized as follows. Section 2 presents the design of a DAMQ buffer using the self-compacting buffer technique. It contains a comparison of the new design with another DAMQ design published by Tamir and Frazier [8]. The performance of DAMQ buffers in k-ary n-cubes and Delta networks is examined in Section 3, assuming a uniform random workload and various packet sizes. The final section summarizes the main conclusions of this work.

4.1 Implementing DAMQ Buffers with Self-Compacting Buffers

Logically, the router can be viewed as being composed of the input port controllers, the (n by n) switch and the output port controllers (Figure 4.1). The input port controller receives incoming packets, performs the routing algorithm for the packet and determines the appropriate output channel number. The (n by n) switch establishes a path from n input controllers to n output controller, and the output controller sends the packet to a neighboring node. Figure 4.2 shows an example of a block diagram for an input port controller. The function of the input port controller can be viewed from three perspectives. First, the input port controller is responsible for receiving the packet and distributing the header part of the packet to the routing algorithm handler and to the packet flow controller. Second, determine the output channel number based on the header information which is received from the input port controller. This task is carried out by the routing algorithm handler. Third, allocate and deallocate the buffer space for incoming and outgoing packets. In this section, we present a packet flow controller



(a)



(b)

Figure 4.1: (a) Logical Blocks of a Router. (b) Logical Blocks of a Input Controller.

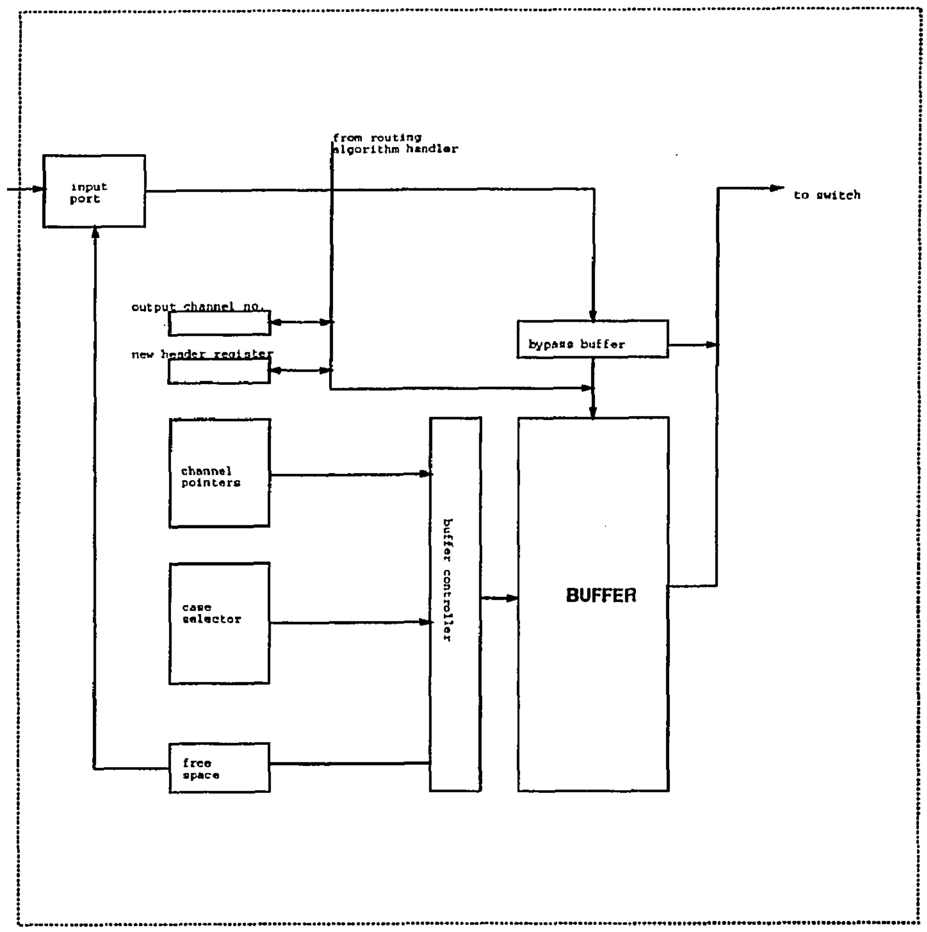


Figure 4.2: Logical Structure of Packet Flow Controller

architecture that implements the DAMQ buffer with a self-compacting buffer.

4.1.1 Self-Compacting Buffers

The packet flow controller consists of a buffer, buffer controller, channel pointers, case selector, a new header register, the output channel number register, a free space register and a bypass buffer. A detailed description of the packet flow controller components follows.

Buffer Management Scheme: For the proposed self compacting buffer scheme, it is assumed that the buffer is divided dynamically into regions with every region containing the data associated with a single output channel. This scheme supports the DAMQ buffer management method introduced in [8]. The self compacting buffer scheme has the following properties:

Property 1: If two channels are denoted as i, k where $i < k$, then the dynamically allocated region for channel i and k always resides in a space addressed by addresses A_i and A_k respectively where $A_i < A_k$.

Property 2: There is no reserved space dedicated for a channel i. If no data are currently requiring the output channel i, then there is no region reserved for channel i.

Property 3: Within the space for each channel, the data are stored in a FIFO manner so that the buffers preserve the order data arrived. When the new packet for channel i has to be written into the buffer, the packet is inserted at the bottom of the region for channel i. When the packet is read out, the packet is read from the top of the region for channel i.

Property 4: For every output channel i, there is an integer number, δ_i, denoting the number of entries present in the region ($\delta_i = 0$ indicates that there is no space reserved at this time for output channel i).

The properties of buffer organization suggests that when an insertion/deletion in the buffer occurs via a write/read operation, there should be a mechanism to access arbitrarily the region that is associated with a channel. In particular, if the insertion of the packet requires space somewhere in the middle of the buffer, the required space must be created by moving all the data which resides below the insertion address. Furthermore, the reading from the top of the region for output channel data may create empty spaces in the middle of the buffer. The data below the read address needs to be shifted up to fill the empty spaces. In the section to follow, we discuss in detail a high performance self compacting capability. The buffer space maintained under the self compacting buffer scheme is shown in Figure 4.3.

Buffer Organization: The buffer consists of n storage locations. The addresses of the storage locations are from 0 to $n-1$. Each storage location can load and store data. For a storage location i, the following actions can occur.

- shift up: storage location i can transfer its content to storage location $i-1$,
- shift down: storage location i can push down its content to storage location $i+1$,
- no action: storage location holds data.

Each storage location has a tag and a data field associated with it as shown in Figure 4.4. The tag field specifies the types of actions of a storage location. The data field simply stores the data. The "u" (shifting up), "d" (shifting down) and

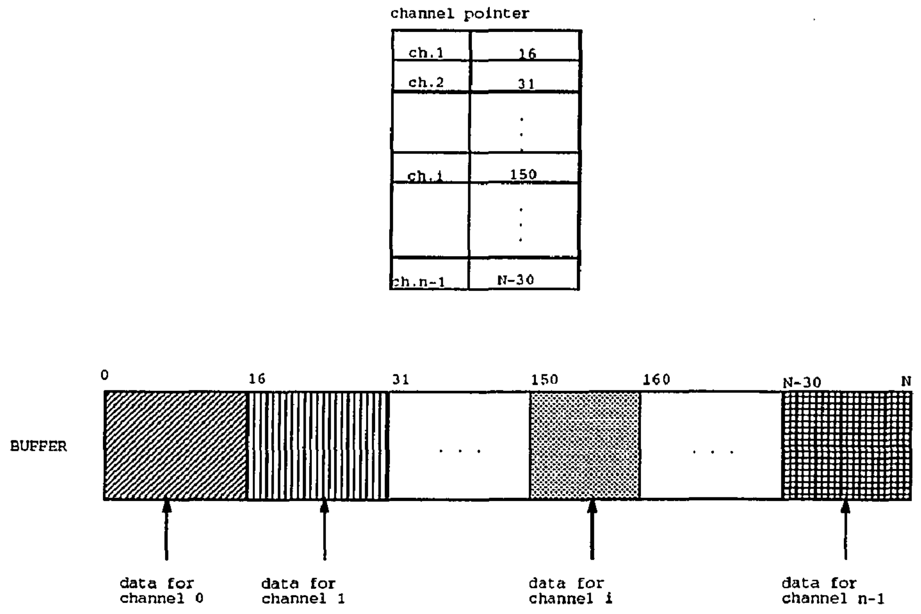


Figure 4.3: Buffer Space

"e" (end of packet) are three bits in the tag field. When a request comes in to read/write data from/into a region, each storage location takes an action according to these three possible tags. Bit settings for shifting data up, down or no action are also shown in Figure 4.4. The end of packet (e) bit is set by the buffer controller (described later) and signifies that the data in the register are the last one of a packet. All three bits in the tag are maintained by the buffer controller.

Buffer Operations and Case Selector: The buffer can read and write simultaneously. Depending on read, write or read/write operations(done in parallel), the tag bits in all storage locations have to be determined accordingly. There are four distinct cases by which the actions of each storage location in the buffer are determined. The function of the case selector is to determine the type of data movement

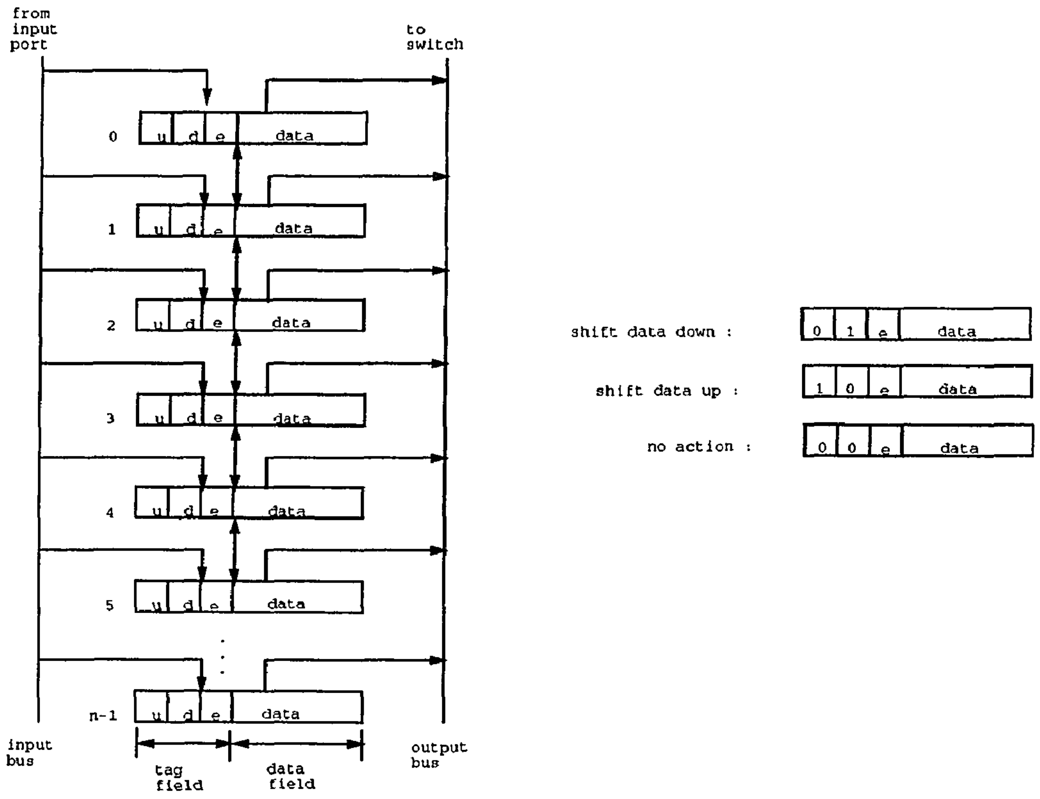


Figure 4.4: Buffer Organization

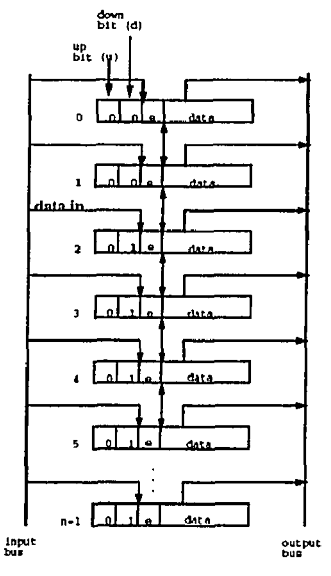


Figure 4.5: Bit Setting Example of Single Write

and feed this formation to the buffer controller. Four cases of data movement are explained next.

case 1). Single Write (Insertion): For a given address to write data in, all storage locations whose addresses are less than the write address leave their data untouched. The storage locations whose addresses are greater than or equal to the write address shift their contents down to open a space in the buffer for incoming data. An example of this case with the write address 2 is shown in Figure 4.5. In Figure 4.5, data are written into storage location 2. The storage locations 0 and 1 do not take any action and tag bits are set to 0(=u) and 0(=d). Storage locations 2 through $n-1$ are shifted down to create a space at address 2 for incoming data and tag bits are set to 0(=u), 1(=d).

case 2). Single Read (Deletion): All storage locations whose addresses are

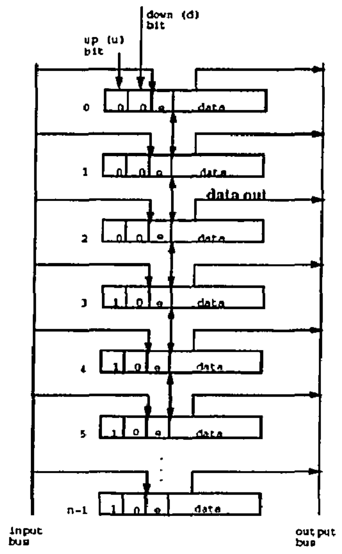


Figure 4.6: Bit Setting Example of Single Read

less than the reading address leave their data as they are. The rest of the storage locations shift the contents of their storage location up. An example of bit setting for this case is shown in Figure 4.6.

case 3). Simultaneous Read and Write (address of read < address of write): In this case, any storage location with addresses smaller than the read address are not affected. The storage locations with addresses which are greater than the read address and less than or equal to the write address should shift their contents upward. The rest of the storage locations take no action. An example of read address at 2 and write address at 5 is shown in Figure 4.7. The bit setting for this case will occur as follows. The down (d bit as shown in Figure 4.7 (a)) and up (u bit as shown in Figure 4.7 (b)) bits are set according to its read and write addresses as in case 1 and 2. Then, "u" and "d" bits are XORed and their results

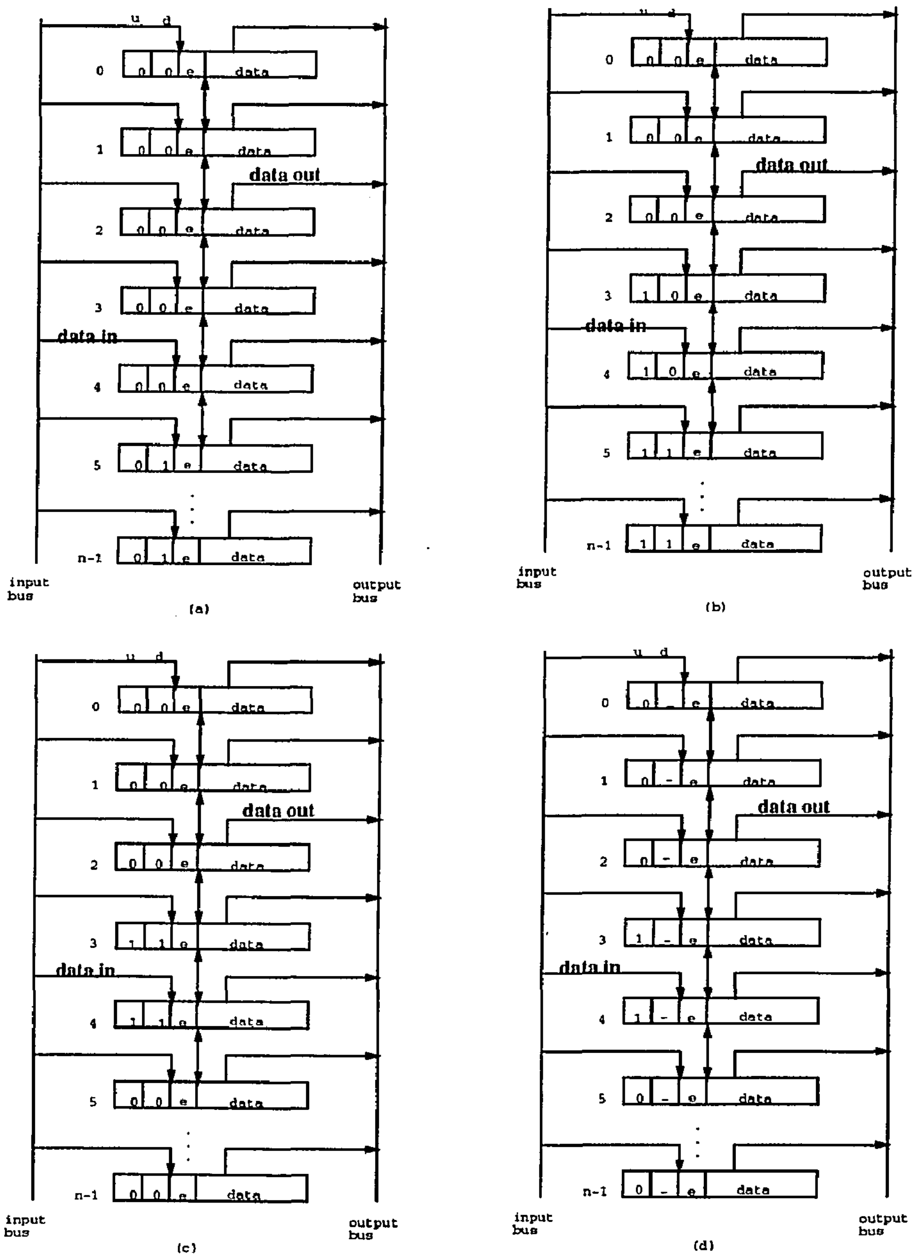


Figure 4.7: Bit Setting Example of Simultaneous Read/Write (Read Address < Write Address)

are left in both "u" and "d" bits (Figure 4.7 (c)). Then, all storage locations take actions according to the "u" bits (Figure 4.7 (d)).

case 4). Simultaneous Read and Write (address of write < address of read): In this case, only the storage locations whose addresses are greater than or equal to the write address and less than the read address, shift their contents down. Any other storage locations require no action. An example of a read address at 5 and write address at 2 is shown in Figure 4.8. The storage locations 0 and 1 need no action. The storage locations 2 and 3 shift their contents down and the storage location 5 through $n-1$ shift their contents up for incoming and out going data. The bit settings of case 4 takes place as following. The "d" bit is set by the read address as in case 2 (Figure 4.8 (a)). Also the "u" bit is set as in the case of 1 (Figure 4.8 (b)) except that the write address is decremented by one before it is fed into the bit setting logic. After all bits are set, the "u" and "d" bits are XORed and their results are left in both bits (Figure 4.8 (c)). The storage locations take action according to the "d" bit (Figure 4.8 (d)).

Buffer Controller: The buffer controller manages the tag of all storage locations in the buffer and it controls the read and write operations. It also generates the end_of_read signal when it detects the EOP(End Of Packet) bit from the tag of the storage location. When the last data byte is written into the buffer, it sets the "e" bit in the tag of the storage location. The inputs of the buffer controller are the case number generated from the case selector, read address and the write address. Once the buffer controller receives all the inputs, it determines the correct bit settings and sets the first three bits ("u","d" and "e") in the tag for all storage locations. The tags in all the storage locations are set in parallel. Supporting

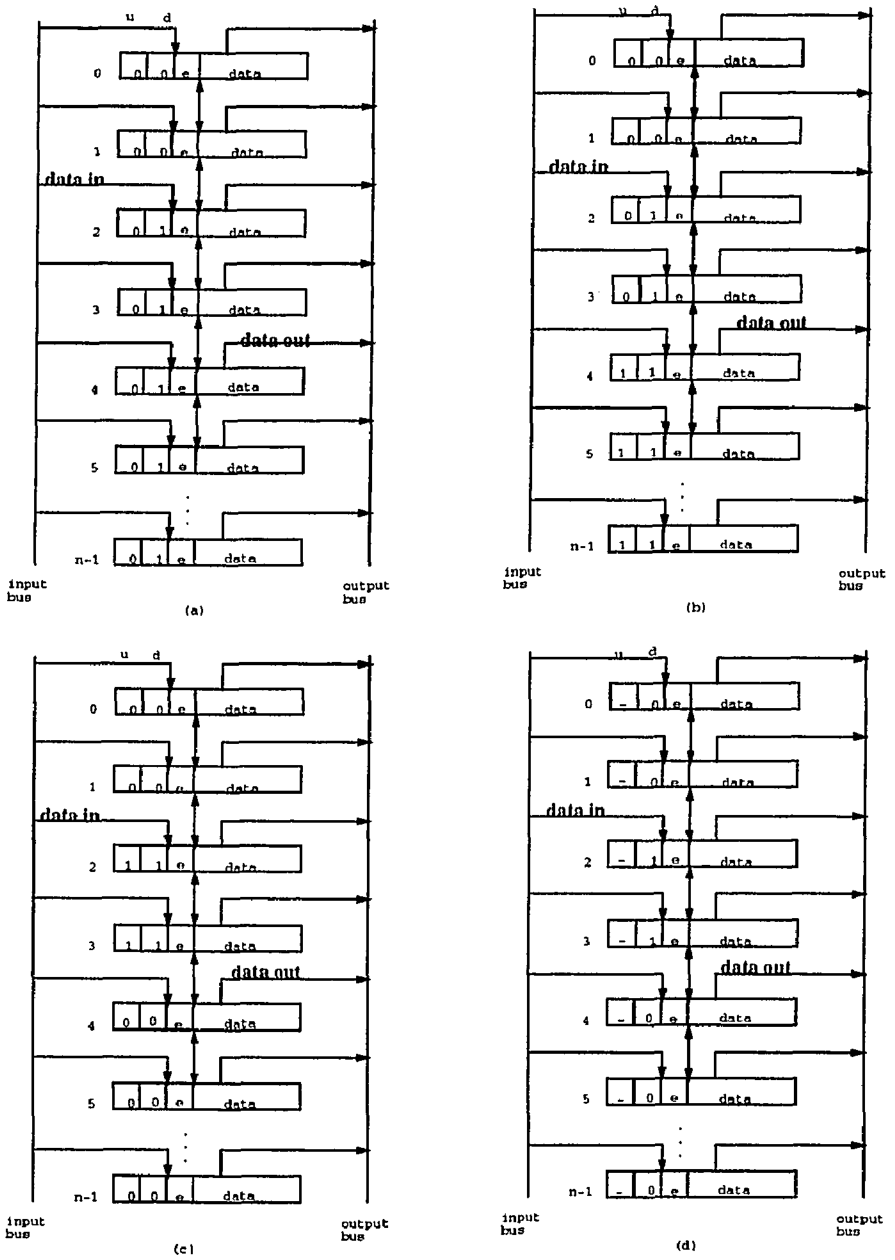
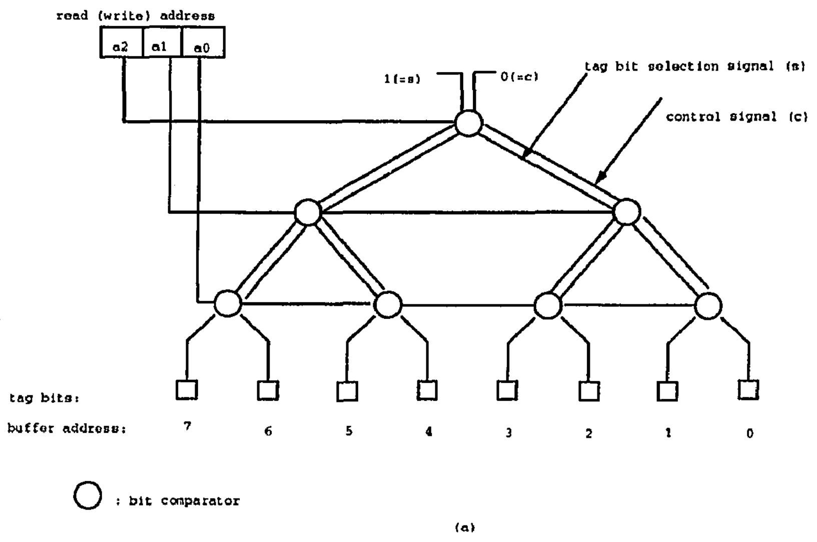


Figure 4.8: Bit Setting Example of Simultaneous Read/Write (Read Address > Write Address)

parallel tag bit settings can be done by associating a comparator to each of the storage locations. Then, the address of the buffer and the read(write) address are fed into the comparator to decide whether the shifting up or down bit (no action if both bits are 0s) should be set to 1 or 0. This scheme results in fast decision making. However, it requires n comparators(with two inputs of $\log_2 n$ bits) for the buffer of size n. We propose parallel bit settings that can be achieved using $n-1$ bit comparators(with two inputs of 3 bits each) in $\log_2 n$ time.

Our proposed method uses comparators organized in a binary tree fashion with one control signal(c), one tag selection signal(s) and one address bit as shown in Figure 4.9. The basic idea of this method is to divide a buffer address into two spaces and set the tag bit in one space to 0 and the tag bit in the other space to 1. For given buffer addresses starting from 0 to $n-1$, the interval of one space will include from 0 to i and the interval of the other space from $i+1$ to $n-1$. As it is shown in Figure 4.9 (a), each leaf of the tree represents an address space in the buffer and it has a tag bit associated with it. The buffer address increases from right to left in the tree. For a buffer with size n, its address can be represented by $a_{p-1}a_{p-2}...a_0$ where $p = log_2 n$. The left most bit of the address is fed into the comparator at the top of the tree and the second left most bit of the address to the comparator at the second level of tree and so on. In addition to the address bit, the tag bit selection signal (s) and the control signal (c) are used as inputs to the comparator. The initial values for the signals "s" and "c" are 1 and 0. The "s" signal carries bit setting information. It will be 0 or 1 when it reaches the leaf node. The "c" signal is used as a control signal. Whenever the comparator at the node receives the "c" signal with value 1, it means that the decision for the node



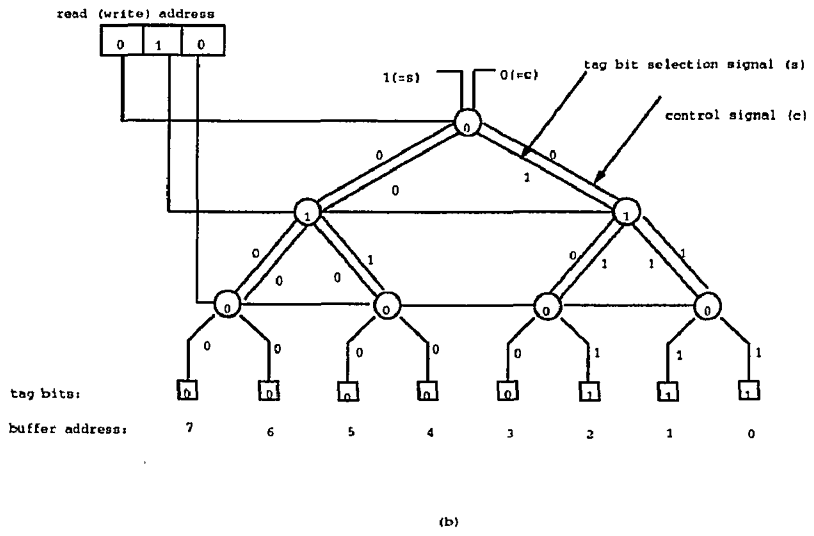


Figure 4.9: (a) Logical View of Binary Tree Method. (b) An Example of Setting Tag Bits.

and the subtree of the node are determined. Signal "s" is then propagated to its children. The logic of the node is shown in Figure 4.10. An example of setting tag bits is in Figure 4.9 (b). In this example, given the address 2(=010 in binary), the tag bits in the buffer address greater than 2 are set to 0 and the rest of them will be set to 1. All control signals and tag bit selection signals are shown in the example. Figure 4.11 shows how address bits are fed into the comparator tree. Address feeding logic is another tree whose number of nodes is equal to the number of nodes in the bit setting controller. The function of each node in the address feeding logic is the same. Each node sends the most significant bit of its content to the bit setting controller. Then it rotates its content one position to the left, and sends the content to its left and right child.

Bypass Buffer: The bypass buffer is an intermediate storage between the input port and the buffer. There are two cases where the bypass buffer is used.

Case 1). When the input port starts receiving data from the paired output port, the data have to be held until the routing algorithm handler determines the output channel number. The bypass buffer is used as intermediary storage so that the input port can receive the incoming data while the routing algorithm handler executes the routing algorithm. The size of the bypass buffer should be large enough to hold incoming data while the routing algorithm handler makes the decision on the output channel number and the buffer determines the write address.

Case 2). The bypass buffer is also used as the flit buffer [10] when the router operates under the wormhole and the virtual cut-through scheme (to be discussed later in detail).

Channel Pointers: There is a channel pointer for each of the output channel.

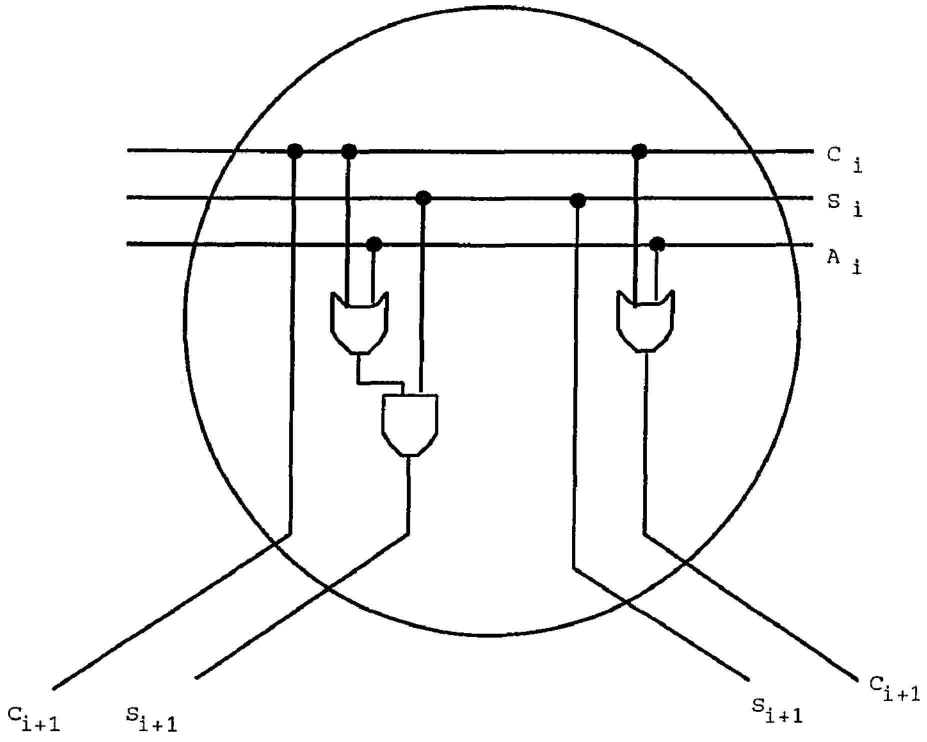


Figure 4.10: Logic at Each Node of Tree

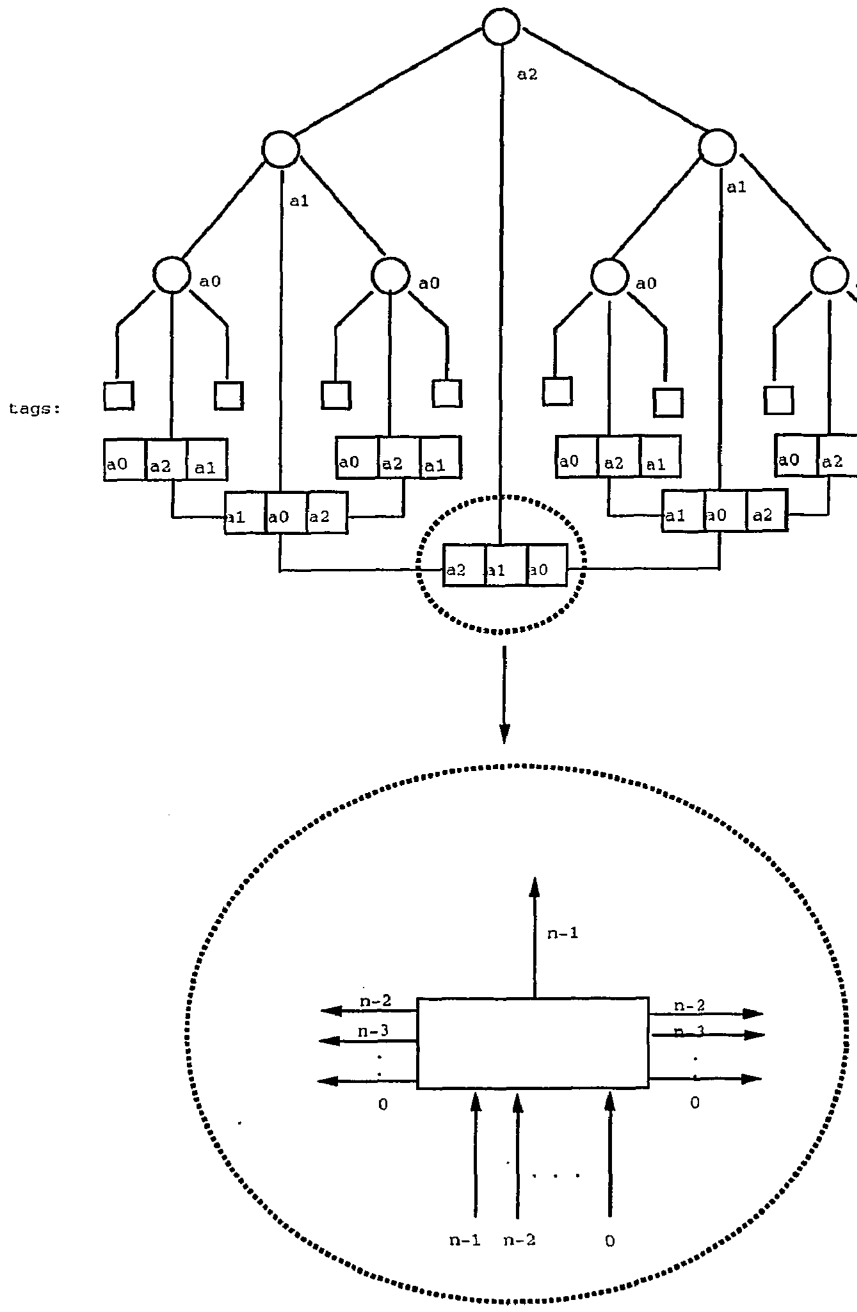


Figure 4.11: Physical Organization of How Address Bits Are Passed to Comparators.

A channel pointer for channel i points to the beginning address of data queued for channel i. There is no channel pointer for channel 0 because it always starts from address 0. The channel pointers are updated at the end of each read and/or write operation. An example organization of the packet flow controller is shown in Figure 4.12.

Buffer Insertion and Deletion

As discussed earlier, when the new packet arrives at the input port, the header information is sent to the routing algorithm handler. After the routing algorithm handler determines the output channel number, it causes an interrupt to the input port that the output channel number and the new header packet are ready. Then, the input port initiates the write operation by hand shaking with the buffer controller. The first data that go to the buffer are the length data. Then the header data go next followed by the data. Details of the write operation are described in table 4.1.

The read operation is initiated by the buffer controller when it receives the acknowledgment from the switch that the requested connection is ready. The switch also sends the channel number that is ready. Then, the buffer controller sends the data to the switch until it detects the EOP bit. Details of the read operation are described in table 4.2.

Timing

The ComCobb chip from UCLA is the first router that implemented the DAMQ buffer [8]. The DAMQ scheme requires a complex buffer management

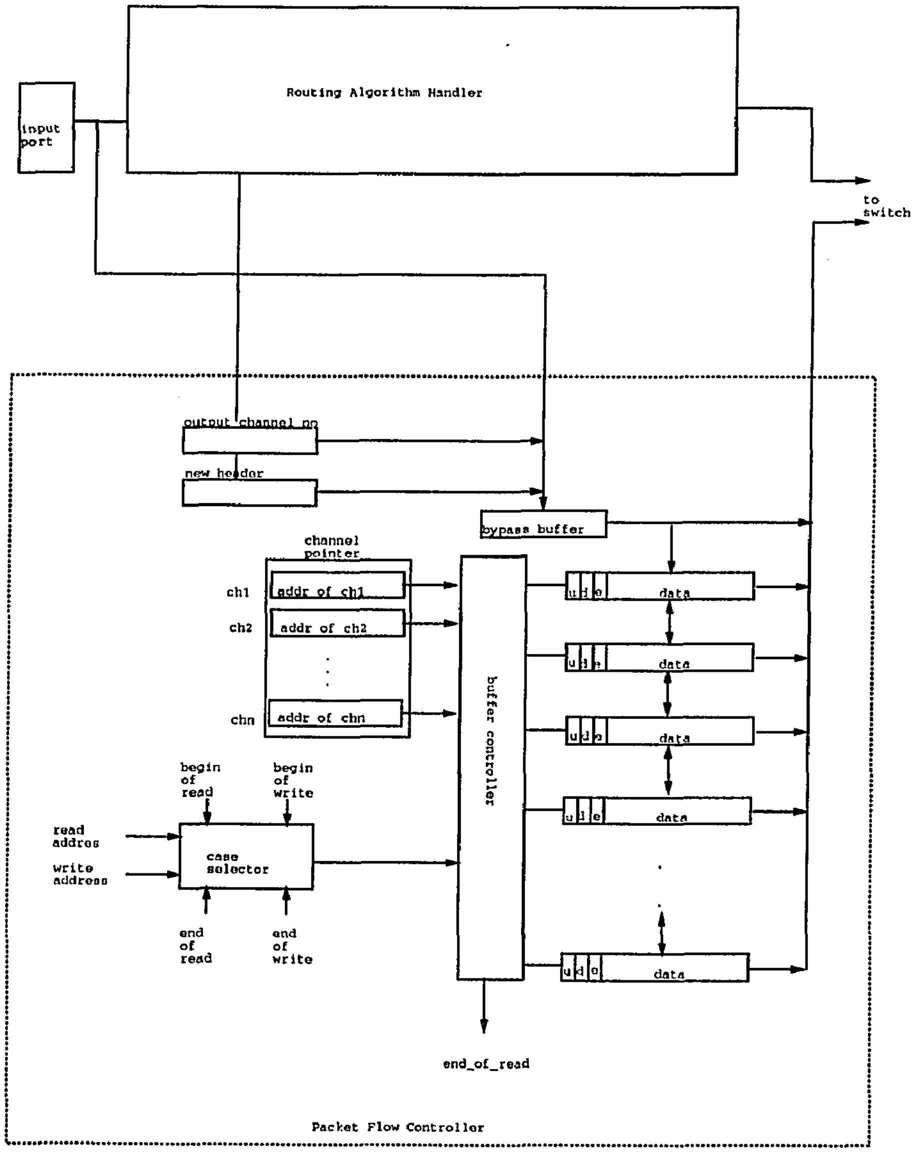


Figure 4.12: An Example Organization of Packet Flow Controller

Table 4.1: Write Operation

t1	Input port sends write a signal to the buffer.
	This signal is fed into the case selector.
t2	Case selector generates the case number and feeds this to the buffer controller.
t3	Shift controller sets tag bits for all storage locations (up, down and no action bit).
t4	Write operation occurs.
	If it was the last write, end of write signal is generated.
	Write address is updated.
	The total free space is updated.
	The channel pointer is updated.

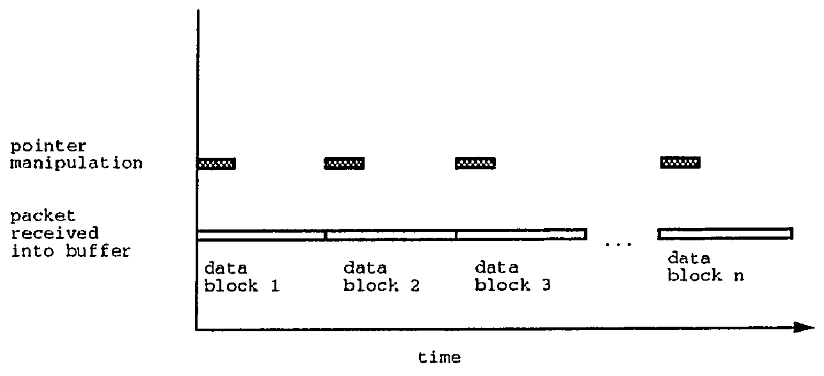
Table 4.2: Read Operation

t1	switch sends acknowledgment to the buffer.
	This signal is fed into the case selector.
t2	Case selector generates the case number and feeds this to the buffer controller.
t3	Shift controller sets tag bits for all storage location (up, down and no action bit).
t4	Read operation occurs.
	If it was the last read, end of read signal is generated.
	If it was the last read, end of packet bit is set into the register pointed to by the current read address.
	The total free space is updated.
	The channel pointer is updated.

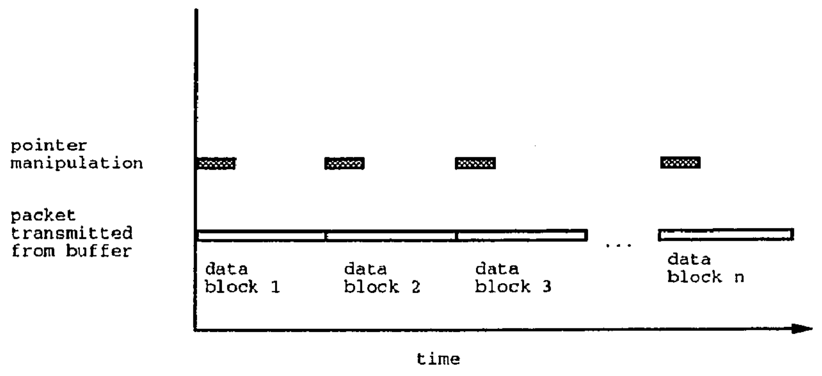
operation. The ComCobb chip used linked list concepts to dynamically allocate the buffer spaces. In the self-compacting buffer, the buffer controller with the case selector and the channel pointers are two key components for the buffer management operation. To minimize the overhead of the buffer management operation, the buffer management operation is overlapped with data transmission/reception. This is done by performing the buffer management operation for $(n+1)th$ block of data while the $(n)th$ block of data is being received/transmitted. Thus, the time (δ) for the buffer management operation will not be seen if (δ) is less than the transmission/reception time of a block of data. This is shown in Figure 4.13. Both the self-compacting buffer scheme and the ComCobb chip utilize the same concept of overlapping the buffer management with data transmission/reception. They achieve the same performance with respect to timing when their total buffer size and block size are equal.

Complexity

In the ComCoBB chip, each block of the buffer is associated with a header byte register, a length byte register and a pointer register. These registers are necessary to implement a linked list to support the DAMQ buffer management. The ComCoBB chip has four channels and each channel can have a maximum of four blocks. In the self-compacting buffer, each block of the buffer has a tag and a shifter. There are $(n+1)$ channel pointers for n channels. Table 4.3 shows a comparison of hardware complexity between the ComCobb chip and the self-compacting buffer. From this table, we can derive the overhead function as the following:



(a) Packet Reception Timing



(b) Packet Transmission Timing

Figure 4.13: Timing Diagram of ComCoBB Chip

Table 4.3: Hardware Complexity of ComCobb Chip and Self-compacting Buffer

ComCobb chip		
Function	*Size*	Quantity
pointer	ln(n)	n
header	8 bits	n
length	2 bits	n
head	ln(n)	5
tail	ln(n)	5
data	t bytes	n
shifter	4 bits	2*n

Self-compacting Buffer		
Function	*Size*	Quantity
tag	3 bits	n
chan ptr	ln(n)	5
data	t bytes	n
shifter	ln(n)	n-1

n = the number of blocks

t = the number of bits per block

$$overhead(ComCobb) = \frac{(ln(n) * (n + 10) + n * 18)}{(n * t * 8)}$$

and

$$overhead(self - compacting) = \frac{(ln(n) * (n + 4) + n * 3)}{(n * t * 8)}$$

The self-compacting buffer has significantly less overhead with respect to latches than the ComCobb chip. Table 4.4 shows the calculation of overhead with several different buffers and block sizes.

4.1.2 Variations

The packet flow controller described before is mainly responsible for buffer management. It includes allocation and deallocation of buffer space for the incoming and outgoing data. The router that we propose supports the circuit switching as well as packet switching. In particular, it supports store-and-forward, wormhole

Table 4.4: An Example Overhead Calculation with 8 Bytes per Block.

buffer size	*ComCobb*	*Self-comp.*
2	37.5%	9.37%
4	39.1%	10.8%
8	38.7%	9.4%
16	38.3%	12.5%

and virtual cut-through for packet switching. In the following, the necessary signals and operations that are required to support the multiple switching techniques are described.

Blocked Signal: The blocked signal is used in circuit switching. The input port and the switch can generate the blocked signal. The input port generates the blocked signal if the buffer is full for the incoming data. The switch generates the blocked signal if the requested output channel is not available. Once the signal is generated, it is back propagated through the reverse order the signal has traveled and goes to the originator of the signal.

Path Cleared Signal: The path cleared signal is used in circuit switching. Circuit switching requires the complete path from the source node to the destination node be reserved before it sends out data. The path cleared signal is generated by the destined router if all the intermediate paths are available.

Supporting Circuit Switching: The circuit switching does not require any buffer because its complete paths from the source node to the destination node are reserved before the transmission of data. When operating under circuit switching, there are two phases to transfer packets. The first phase is the "path set up" phase. In this phase, all input ports that are included in the required paths will receive "path set up" packets that have the destination address and that are the sole content of the packet. If any input port is busy at the time of receiving the "path set up" packet, the input port raises the blocked signal. If the input port is idle, it forwards the packet to the next router. The state of the input port at this time will change to busy state. When the input port forwards the packet through an output channel, the output channel is dedicated to the input port until one of two cases happen. The first case is that, the complete path cannot be set up. In this case, the blocked signal will be back propagated from the busy router and the input port releases the dedicated output channel. The second case is when the path is not needed between the source and the destination node. The second phase is the packet transfer phase. In this phase, the input already has the dedicated output channel. Thus, when the packet arrives at the input port, it simply relays the packet through the dedicated output channel. When the usage of the paths are not needed, the path cleared signal is sent from the destination node and all input ports that were part of the path go back to idle mode and release the dedicated output channel.

Supporting Store-and-forward: In the store-and-forward, the request for the output channel connection is sent by the buffer controller in two cases. The first case is at the end of reception of a packet if the packet is the first one in the buffer. The second case is at the end of transmission of a packet and if there are remaining packets in the queue from the same channel. When the routing algorithm handler determines the output channel number, it will inform it to the input port. Then, the input port will initiate hand shaking with the buffer controller to start writing data into the buffer. The first data will be the length data [70]. The second data word is taken out of the new header register which contains new header information prepared by the routing algorithm handler. Transmitting data to the switch can occur in parallel with the reception of the data into the buffer.

Supporting Wormhole: The wormhole routing is similar to the circuit switching in that once the input port gets an output channel, it does not release the output channel until the completion of the packet transfer. But it does not preallocate all paths needed to forward the packet as in the circuit switching. Rather, it does that as it progresses along the path that leads to the destination. Even if the front of the packet is blocked, input ports do not release the output port channel but wait until the front packet can advance. In the wormhole scheme, there is only one packet in the buffer at a time. The input port writes the data into the bypass buffer while waiting for the connection ready acknowledgment from the switch. If it receives the acknowledgment signal from the switch, it starts transmitting the data directly from the bypass buffer to the output port. The read and write operation may overlap in the wormhole scheme as well. When the input port is ready to forward a packet again after being blocked, it does not have to re-request the connection of the output channel because it never released it. Thus, in wormhole routing, it can save time that is required for arbitration of the output channel. But if the input ports are being blocked for long periods of time, it wastes output channel resources.

Virtual Cut-through: In the virtual cut-through, all operations are identical to the wormhole scheme until the packet can not advance to the next switch. In this case, the incoming data are stored into the buffer from the bypass buffer and transmitted later. The input port also releases the connection back to the switch so that other input ports can use it if possible. After the blocked condition is cleared, the buffer controller has to request the connection again for the data because the connection to the switch was released when buffered. Figures 4.14 to 4.16 show the control flow of the store-and-forward, the wormhole and the virtual cut-through schemes.

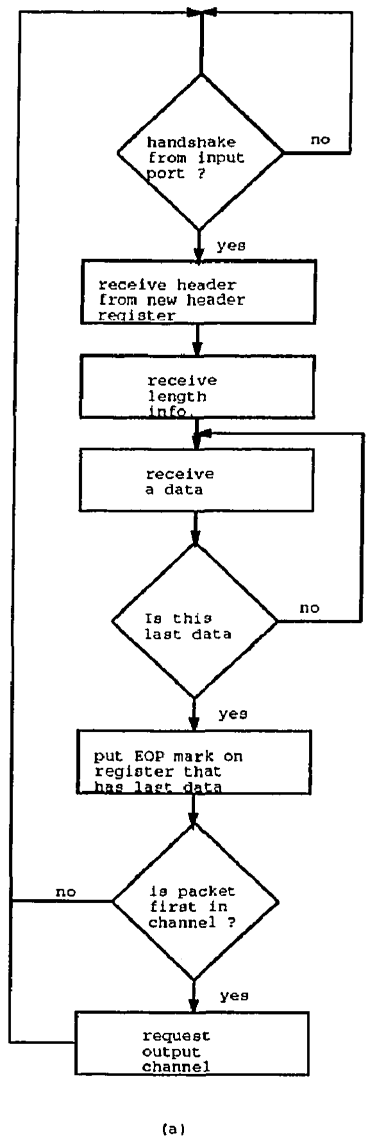
4.2 Performance of DAMQ Buffers in k-ary n-cubes and Delta Networks

This section compares the performance of DAMQ, FIFO and ideal buffers. The data show how closely a realistic buffer design (the DAMQ buffer) approximates an ideal buffer, which is much more expensive to implement. The data expand the data reported by Tamir and Frazir [8] by considering a broader class of network topologies, and multiple packet sizes. Tamir and Frazir's results considered only a 64 node Omega network and single flit packets. This section reports results for several examples of k-ary n-cubes and Delta networks for multiple, fixed packet sizes. In addition to this, this section provides additional data on random packet sizes.

This section also presents a useful technique for making fast, accurate approximations of network performance using results from simulations of a single router.

4.2.1 Methodology

The principle metric for comparing the different router implementations is the average latency experienced by a packet traveling through a network constructed


handshake from input port ?
no
yes
receive header from new header register
receive length info.
receive a data
Is this last data
no
yes
put EOP mark on register that has last data
no
is packet first in channel ?
yes
request output channel
(a)

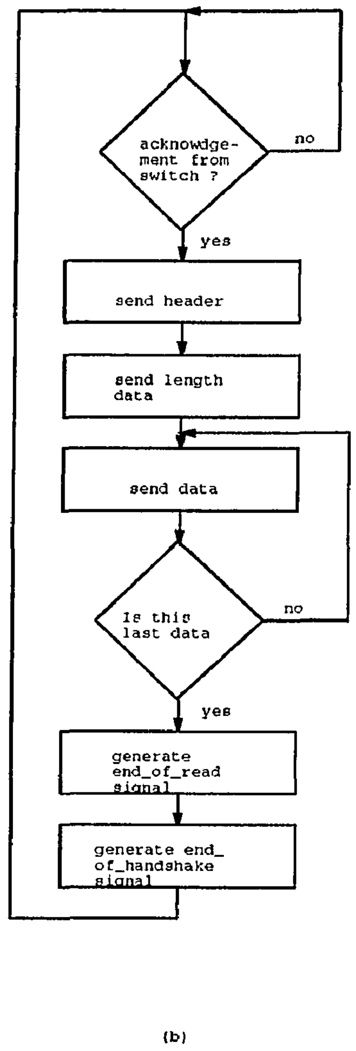

acknowdge-ment from switch ?
no
yes
send header
send length data
send data
Is this last data
no
yes
generate end_of_read signal
generate end_of_handshake signal
(b)

Figure 4.14: Flow of Buffer Controller for Store-and-forward. (a) Writing to Buffer (b) Reading from Buffer.

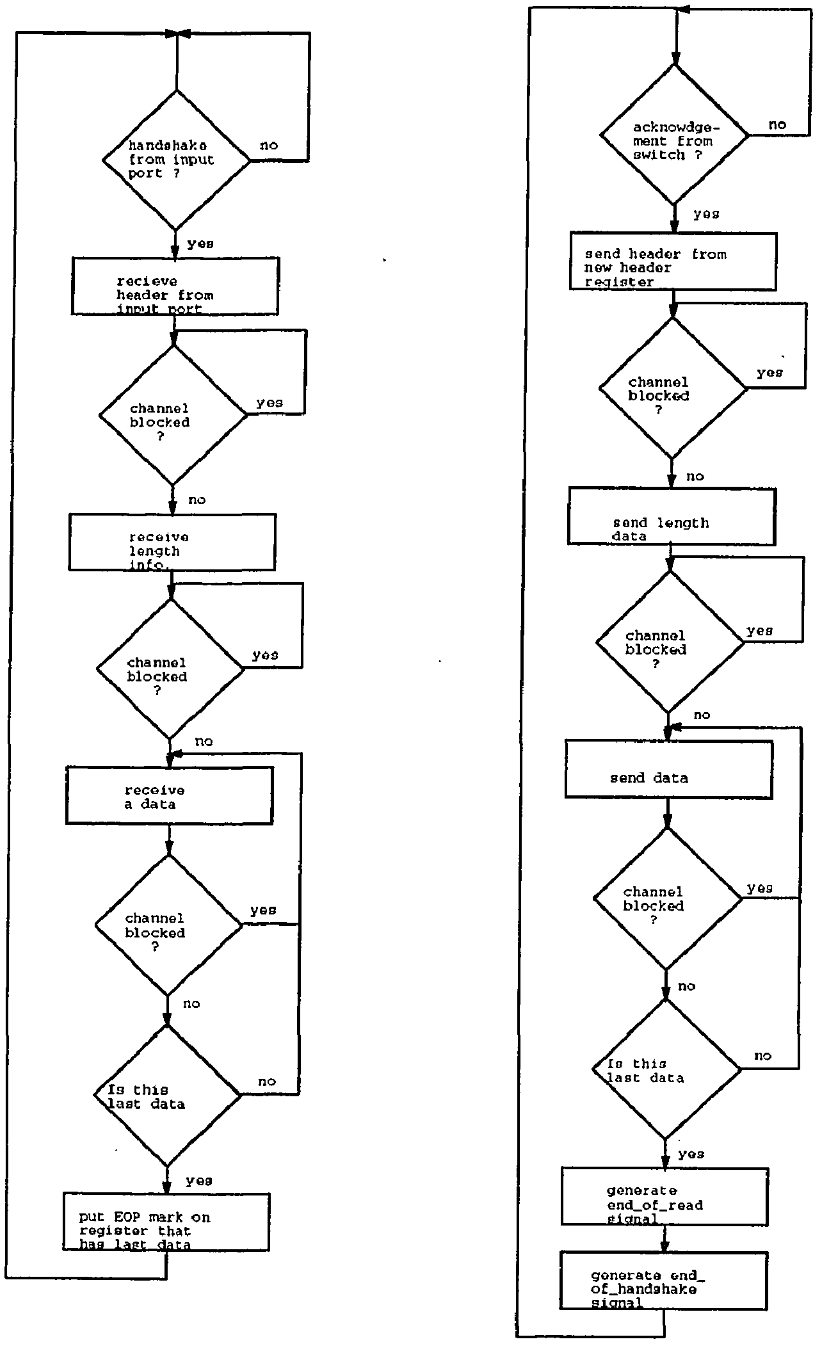


Figure 4.15: Flow of Buffer Controller for Wormhole. (a) Writing to Buffer (b) Reading from Buffer

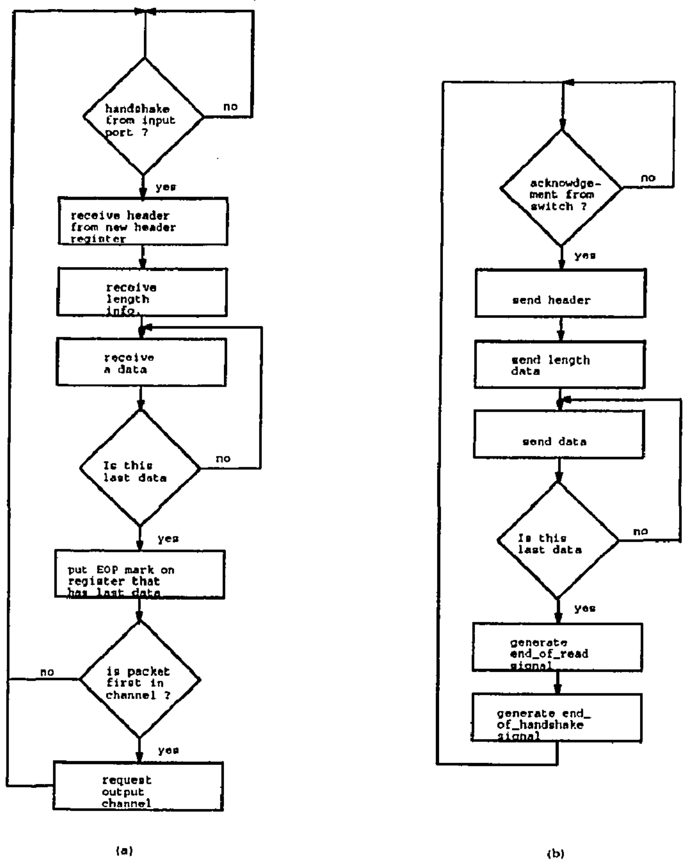

handshake from input port ?
no
yes
receive header from new header register
receive length info.
receive a data
Is this last data
no
yes
put EOP mark on register that has last data
is packet first in channel ?
no
yes
request output channel
(a)
acknowdge-ment from switch ?
no
yes
send header
send length data
send data
Is this last data
no
yes
generate end_of_read signal
generate end_of_handshake signal
(b)

Figure 4.16: Flow of Buffer Controller for Virtual Cut-through. (a) Writing to Buffer. (b) Reading from Buffer.

from a collection of such routers connected in some topology. The topologies considered are specific examples of k-ary n-cubes and Delta networks.

The data reported in this section come from three sources:

1. *Published Data*: All of the data for ideal buffers come from published simulation results used to validate various analytic models [72, 73]. A small portion of the DAMQ and FIFO results comes from Tamir and Frazier [8] (the data for single-flit packets on a 64 node Omega network).

2. *Simulations of Complete Interconnection Networks*: These were obtained from our network simulator instrumented to collect statistics such as channel utilization, latency and routing distribution of a DAMQ buffer.

3. *Simulations of Single Switches*: For k-ary n-cubes and Delta networks, each router in the network has the same set of routing probabilities from inputs to outputs, and the same set of channel utilizations on the input and output ports. This symmetry can be exploited to approximate the performance of a complete network with simulation results for a single router. This approximation has been used in several analytic models of k-ary n-cubes [72, 73] and Delta networks [29], assuming ideal buffers. The data presented here apply to DAMQ buffers. We present data for single router simulations to show that this approximation compares very favorably with the data from full network simulations.

All simulations were made with the following assumptions:

1. Infinite buffers. This simplified the simulation tremendously, and reflects the

fact that most routers are designed with a sufficient number of buffers that blocking is negligible.

2. Packet destinations are uniformly distributed across all processors (but not including the source processor).

3. Fixed packet size.

4. Packet interarrival times are geometrically distributed with parameter p (ie. the probability that the next packet arrive after n dead cycles is p). This implies an average arrival rate of p.

5. Infinite buffers at the source and destination "processors".

6. Virtual cut-through flow control.

7. Conflicts within a router were resolved as follows. Within a router, it is possible that multiple packets destined for the same output port can have conflicts. These were resolved with an arbitration scheme that granted the output port to the input with the longest blocking time. The DAMQ buffer also requires arbitration among packets within a channel that are ready to be sent at the same time. These were resolved in the same way: packets that have been blocked the longest have the highest priority.

8. The latency of a packet transmission was measured from the time the first flit of the packet is injected into a network router to the time that the last flit leaves the last router in its path.

9. Steady-state was assumed to be reached by simulating a large number of network cycles. Several experiments verified that increasing the number of simulated cycles had negligible impact on the results.

4.2.2 Performance of k-ary n-cubes Constructed with DAMQ Buffers

A k-ary n-cube is a network with n dimensions having k nodes in each dimension. The k-ary n-cube network has the same set of channel utilization on the input and output ports. Figure 4.17 shows an example of routing probability of each input and output channel for a 1024 node 3D Torus network. The k-ary n-cube network uses the dimension ordered routing on a virtual channel developed by Dally [10]. In the dimension ordered routing, the routing is performed in order of decreasing dimension $(k-1, k-2, \ldots 1, 0)$ by comparing the k digits of the radix k addresses of the packet destination and router. A packet with destination d at router i is sent along the highest dimension at which d and i differ; when $d = i$, the packet has arrived at its destination. To prevent deadlock, the buffers for each input at a router are divided into channels to prevent cyclic dependencies among buffers at different routers. Virtual channel 0 is taken when $d < i$, and 1 otherwise. In our simulation, the unidirectional channels were assumed for simplicity.

A router in a k-ary n-cube network requires $n + 1$ channels; n channels for n dimensions, and one channel for a local processor. Under a uniform workload and dimension ordered routing on virtual channels, each router in a k-ary n-cube network has the following properties:

1. The channel utilization of the n routing channels is given by:

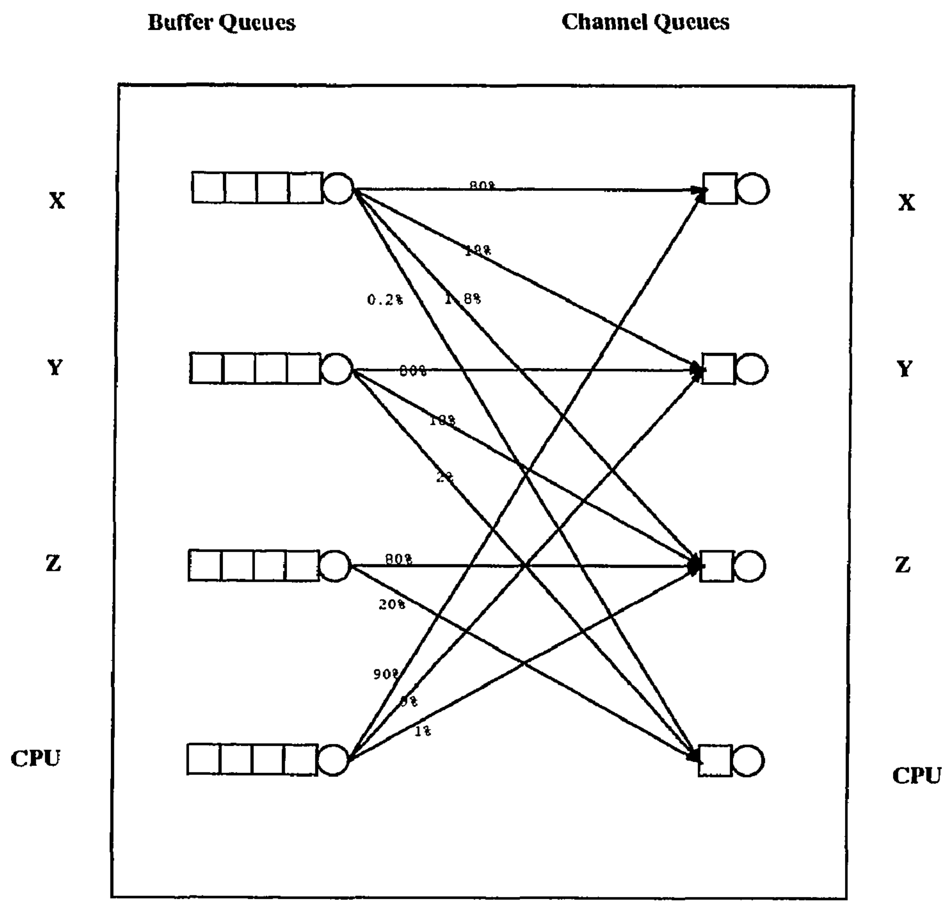


Figure 4.17: Routing Probability Distribution for a Unidirectional 3-D Torus Network ($k = 10$ $n = 3$).

$$\rho = \frac{mBnk_d}{n} = mBk_d$$

where ρ is channel utilization, m is message generation rate, n is network dimension, B is message size, k_d is the average distance a message must travel in each dimension and is given by $k_d = (k-1)/2$ for end-around connection and $k_d = (k - (1/k))/3$ for no end-around connection [72].

2. The channel utilization of the input and channels of the local processor is m.

3. The routing probabilities of the n routing channels are:

   For given input channel i and output channel j, the routing probability, $R_{i,j}$, is:

$$R_{i,j} = (k-2)/(n) \qquad \text{for } i = j$$

and

$$R_{i,j} = (2)/(k) * k^{-(n-j)} \qquad \text{for } i = n+1$$

and

$$R_{i,j} = (2)/(k) * k^{-(i-j-1)*(1-1/k)}$$

for the rest of channels.

4. The routing probabilities for the input port from the local processor are:

   For an output channel j,

$$R_{i,j} = k^{-(n-1)} \qquad \text{for } j = n$$

and

$$R_{i,j} = k^{-(j-1)} * (1 - 1/k)$$

for the rest of the channels.

A simulation run of a single router provides the the average waiting time, w, per packet. Then, the average latency of a message, T, through the network can be calculated by:

$$T = (1 + wB)nk_d + B.$$

for networks that use virtual cut-through. Here, $(1 + wB)$ represents the delay at a router and multiplying the average distance(nk_d) to it, we can get the average latency for a unit packet. Since, we are assuming that the virtual cut-through is used, B is added to get the average latency for a message.

Fixed, Unit-Length Packets

Figures 4.18 to 4.23 show plots of average latency versus channel utilization for the following k-ary n-cubes, assuming fixed, unit-length packets:

1. $n = 2$, $k = 8, 10$ (These are two dimensional meshes with end-around connections.)

2. $n = 3$, $k = 6, 8$ (These are three dimensional meshes with end-around connections.)

3. $n = 7, 8$, $k = 2$ (These are hypercubes.)

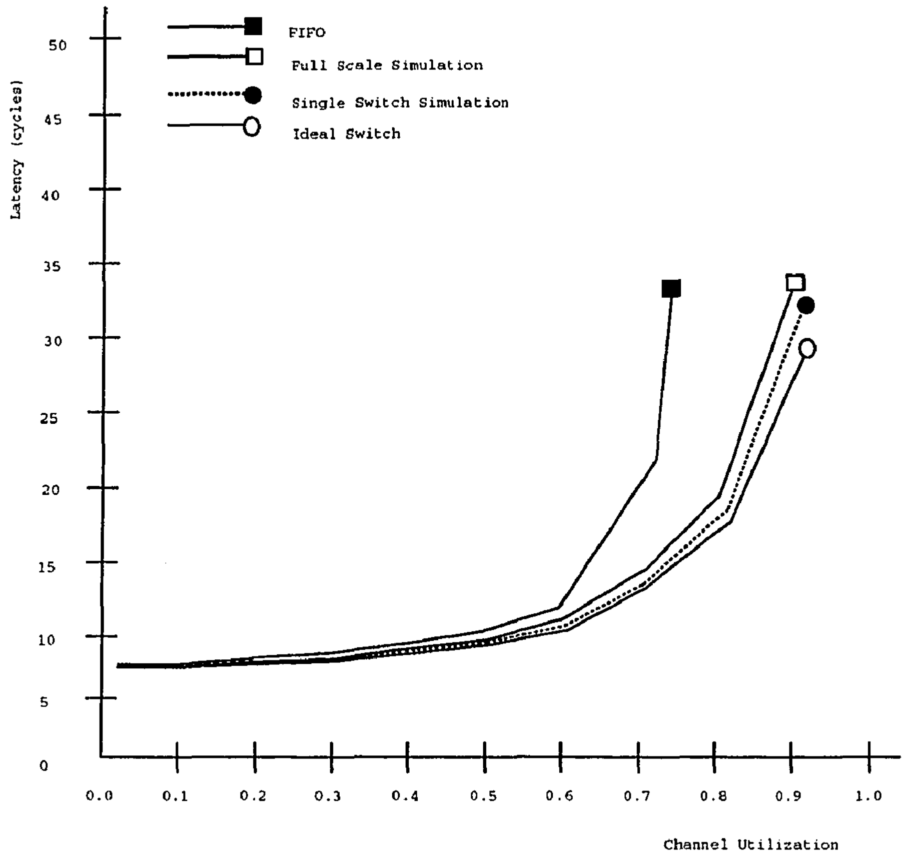


Figure 4.18: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 3-D Torus ($k = 6$ $n = 3$).

Each plot compares ideal, FIFO and DAMQ buffers. Single router DAMQ results are also shown to validate the single router approximation. The latency data for ideal buffers were taken from [72] and [73]. Channel utilization was measured at the class of channel with the highest amount of traffic (the "bottleneck" channels). For the mesh cases this is any of the n routing channels. For the hypercubes these are the "processor" input channels.

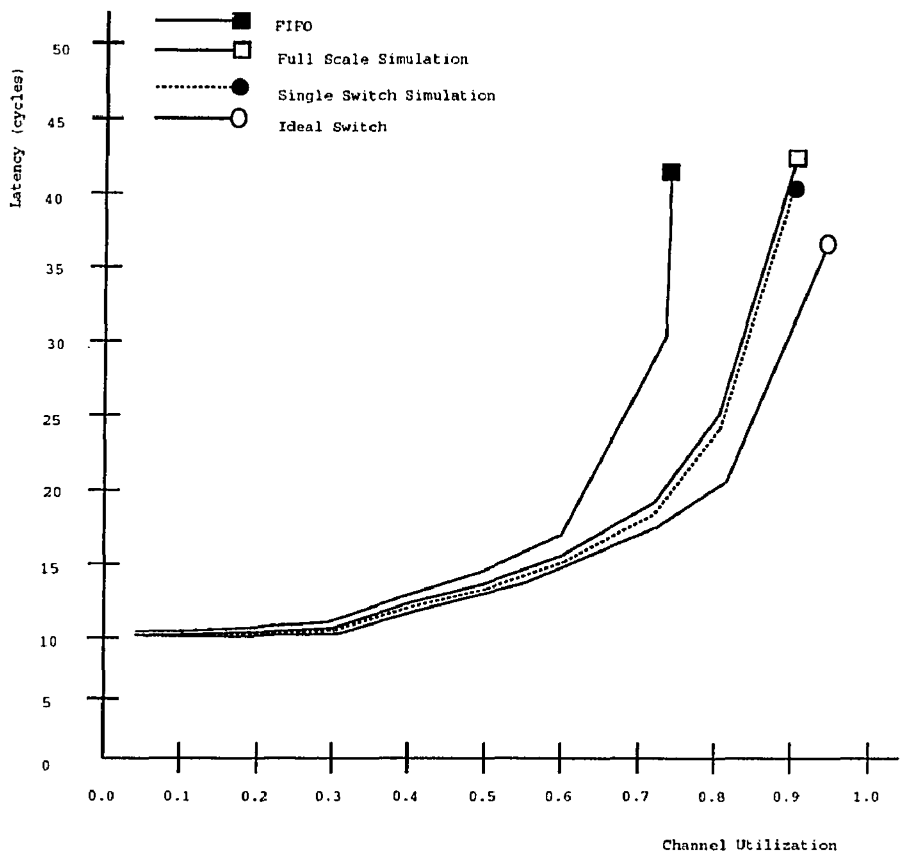


Figure 4.19: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 3-D Torus ($k = 8$ $n = 3$).

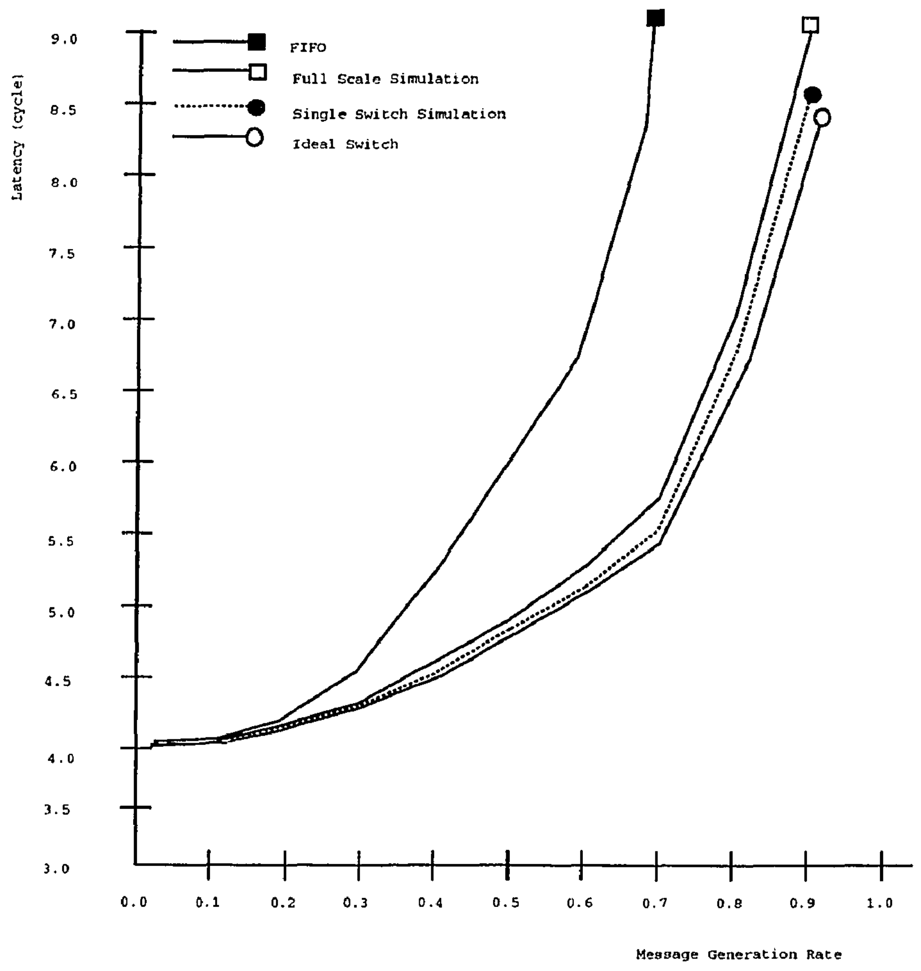


Figure 4.20: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for Hypercube ($k = 2$ $n = 8$).

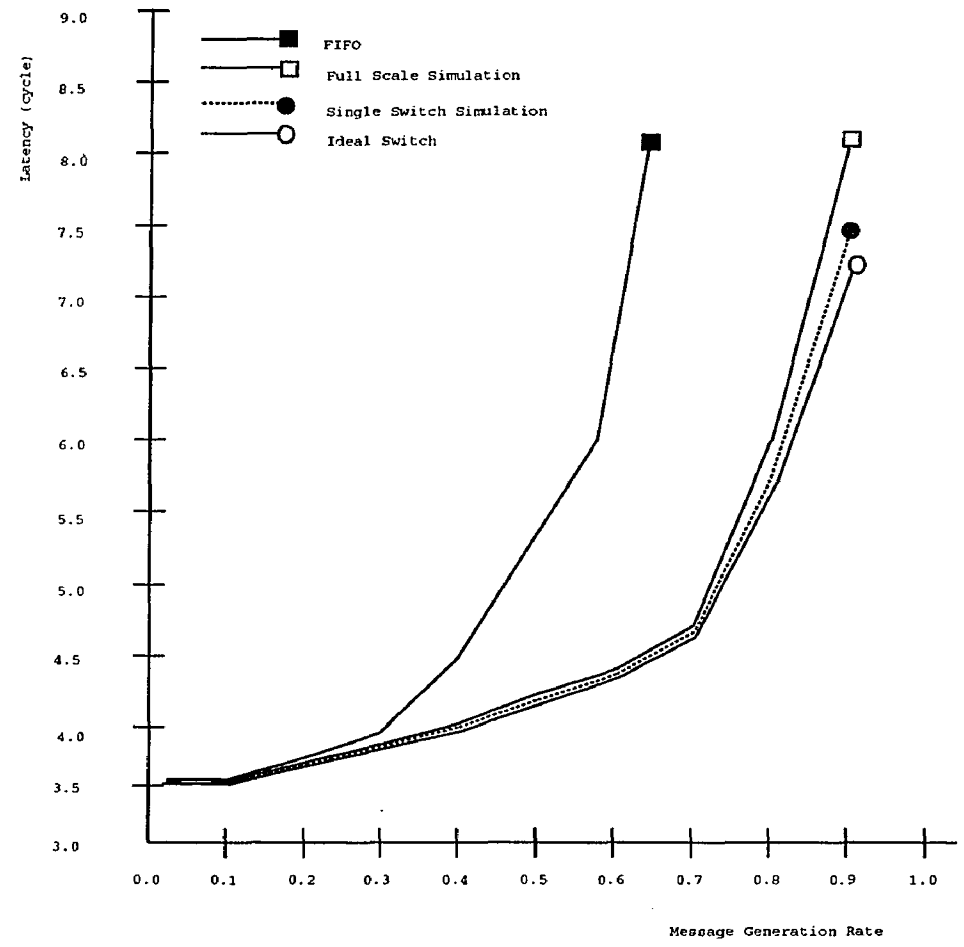


Figure 4.21: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for Hypercube ($k = 2$ $n = 7$).

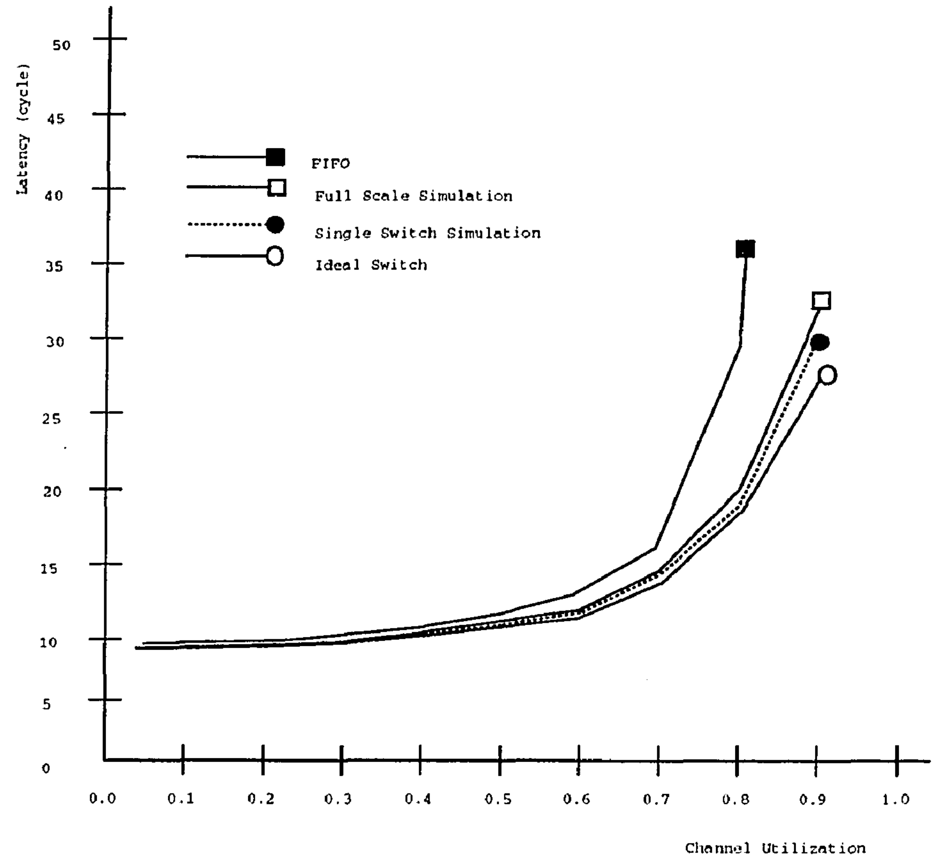


Figure 4.22: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 2-D Mesh ($k = 10$ $n = 2$).

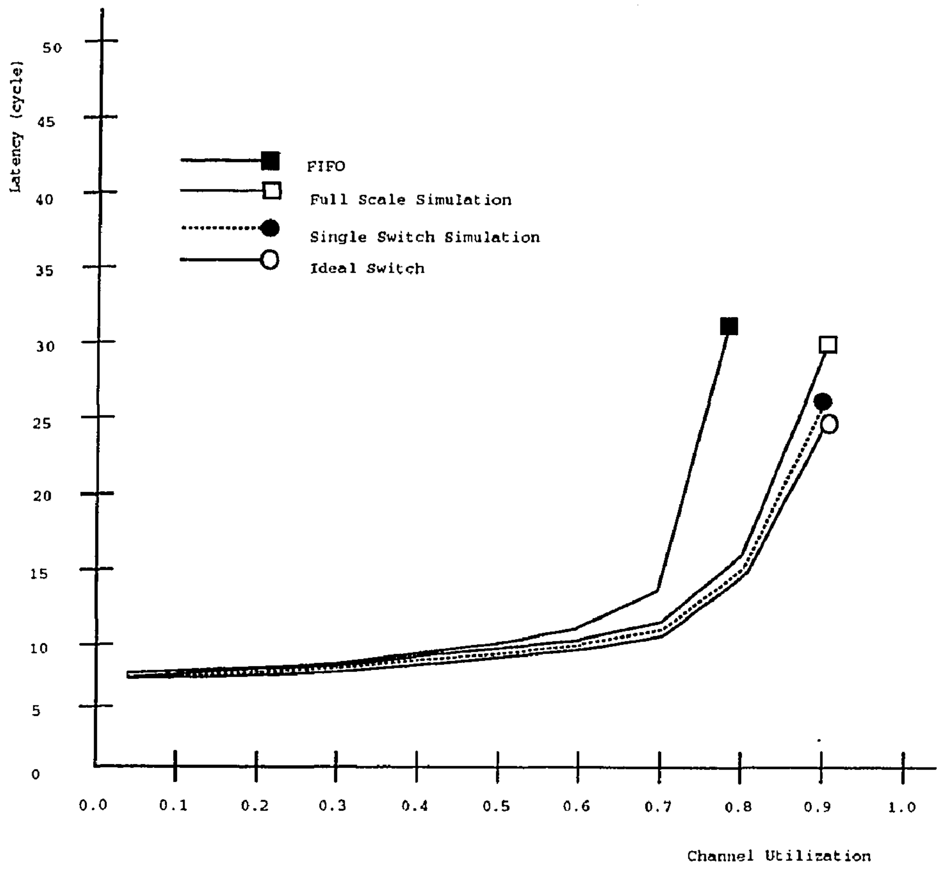


Figure 4.23: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 2-D Mesh ($k = 8$ $n = 2$).

As expected, DAMQ buffers provide significant improvement over FIFO buffers, and do not perform quite as well as ideal buffers. The single router results provide a very good approximation of full-network performance. Furthermore, the single router simulations took 10 to 100 times less simulation time than the full network simulations. Although the differences are small, the single router model consistently underestimates the latency for high channel utilization. The reason for this was described by Agarwal [72] for a similar effect encountered when extending single router analytic results: in k-ary n-cubes using the dimension ordered routing algorithm, packets suffer higher-than-average delays in the higher dimension and this fact was verified by simulation [72].

Fixed, Multi-flit Packets

Figures 4.24 to 4.26 shows the impact of increasing the packet size to 2, 4 and 8 flits for an 8-ary 2-cube. The trends are similar to those observed for a packet size of one. The absolute latency values, however, increase markedly as the packet size is increased; the increase is roughly proportional to the increase in packet size. This corresponds to the fact that whenever a packet must wait in a queue, the wait time is proportional to the size of the packets in front of it. As before, single router simulation provides a very good approximation.

4.2.3 Performance of Delta Networks Constructed with DAMQ Buffers

A Delta network is defined as an a^n-by-b^n switching network with n stages consisting of a-by-b crossbar switches. In the Delta network *digit routing* is used. In digit routing, a digit with a routing tag at each stage determines which output

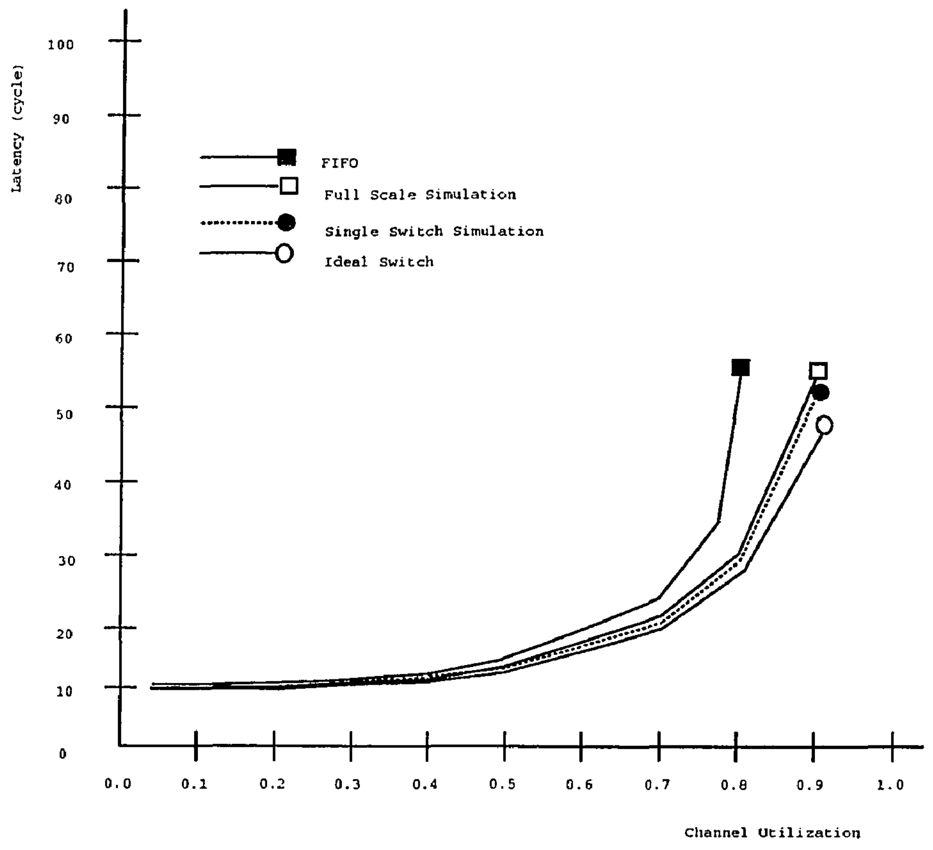


Figure 4.24: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 2D Mesh (k = 10 n = 2). Packet size is 2 unit packets.

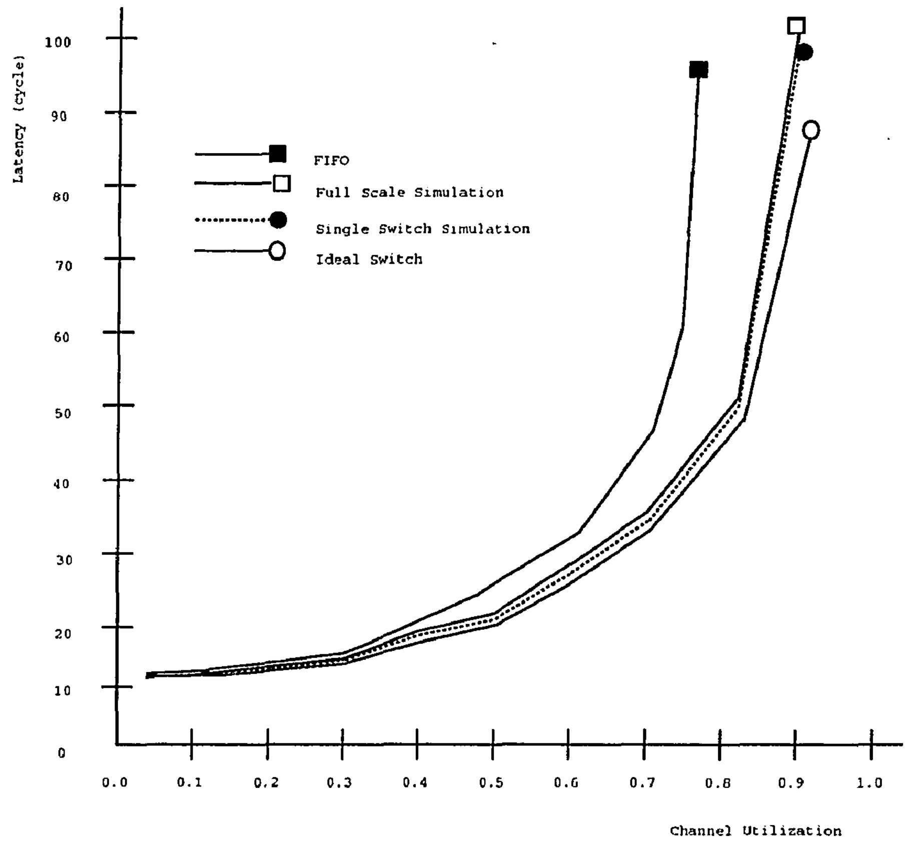


Figure 4.25: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 2D Mesh ($k = 10$ $n = 2$). Packet size is 4 unit packets.

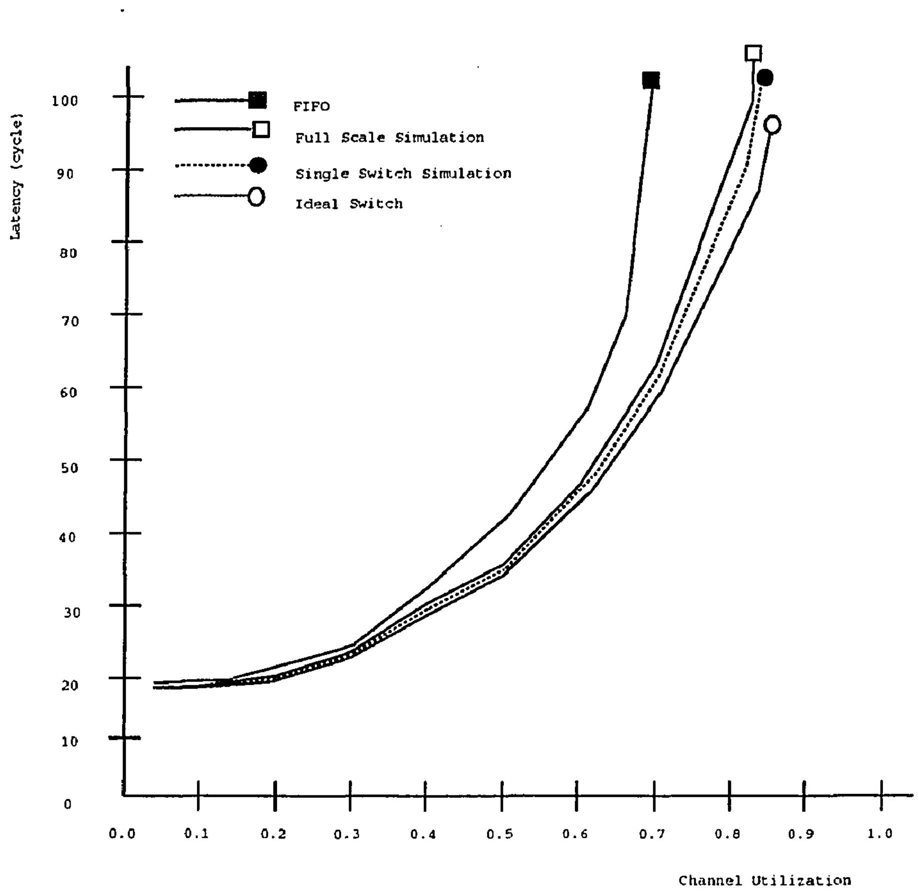


Figure 4.26: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 2D Mesh ($k = 10$ $n = 2$). Packet size is 8 unit packets.

channel should be used for routing data. The digit routing is inherently deadlock-free. Since Delta networks are multistage, no special ports are needed for local "processors".

An Omega is a particular example of a Delta network. A router in an Omega network comprised of n-by-n switches, under uniform workload, has the following properties [29]:

1. The channel utilization of the n input and n output channels is simply:

   $$\rho = mB$$

   where m is message generation rate and B is the length of a message.

2. The routing probabilities for the n inputs are: $1/n$.

As for k-ary n-cubes, a simulation run of a single router provides the average waiting time, w, per packet. The average latency of a message, T, through a complete network can be estimated by:

$$T = (1 + wB)n + B.$$

where n is the number of stages that a message travels and B is the length of the message. Similar to the average latency for a message in k-ary n-cube, $(1 + wB)$ represents the delay at a router and multiplying by the number of stages gives the average latency for a unit packet. Since the virtual cut-through is used, adding B to the average latency for the unit packet yields the average latency for a message of length B.

Fixed, Unit-Length Packets

Figures 4.27 and 4.28 compare the performance of three stage and four stage radix 4 Omega networks, assuming fixed, unit-length packets. The data for ideal Latency data for ideal buffers was taken from [29].

Again, the trends are the same as for k-ary n-cubes: DAMQ buffers provide significant improvement over FIFO buffers, and the single router results provide a very good approximation of full-network performance.

Fixed, Multi-Flit Packets

Figures 4.29 to 4.31 show that increasing the packet size increases the latency by a proportional amount, as was observed for the k-ary n-cubes. The saturation points remain about the same.

Random Packet Sizes

Both the self-compacting buffer and the ComCobb chip support variable packet sizes. This means that buffer fragmentation can occur if the size of a packet is not the multiple of the block size in the buffer. In this section, we measured the performance of the DAMQ buffer with random packet sizes and compare the results to the FIFO buffer and to fixed size packets (with no fragmentation). The objective of this measurement is to discover performance changes resulting from fragmentation in a parallel system which supports variable packet sizes. The simulation was done on an Omega network with 16, 64 and 256 nodes constructed of 4 x 4 switches. The implementation of the simulator was the same as described earlier. In all cases, the size of the buffer was fixed to 128 bytes. We chose random

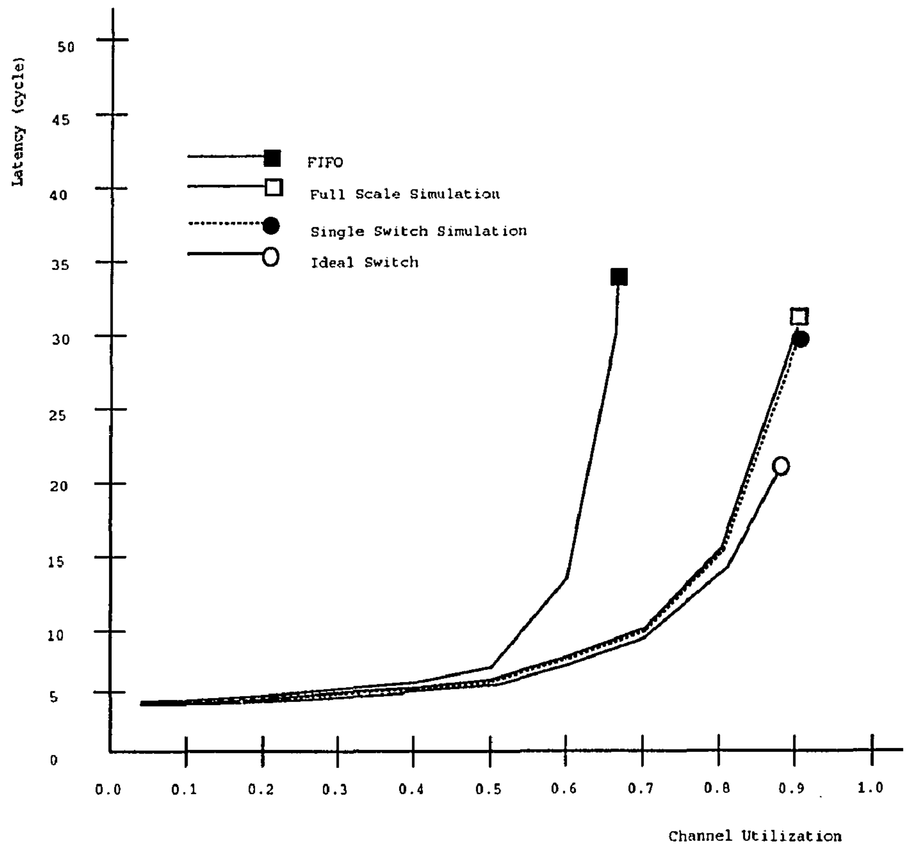


Figure 4.27: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 4 x 4 switch Omega network with 256 nodes (4 stages).

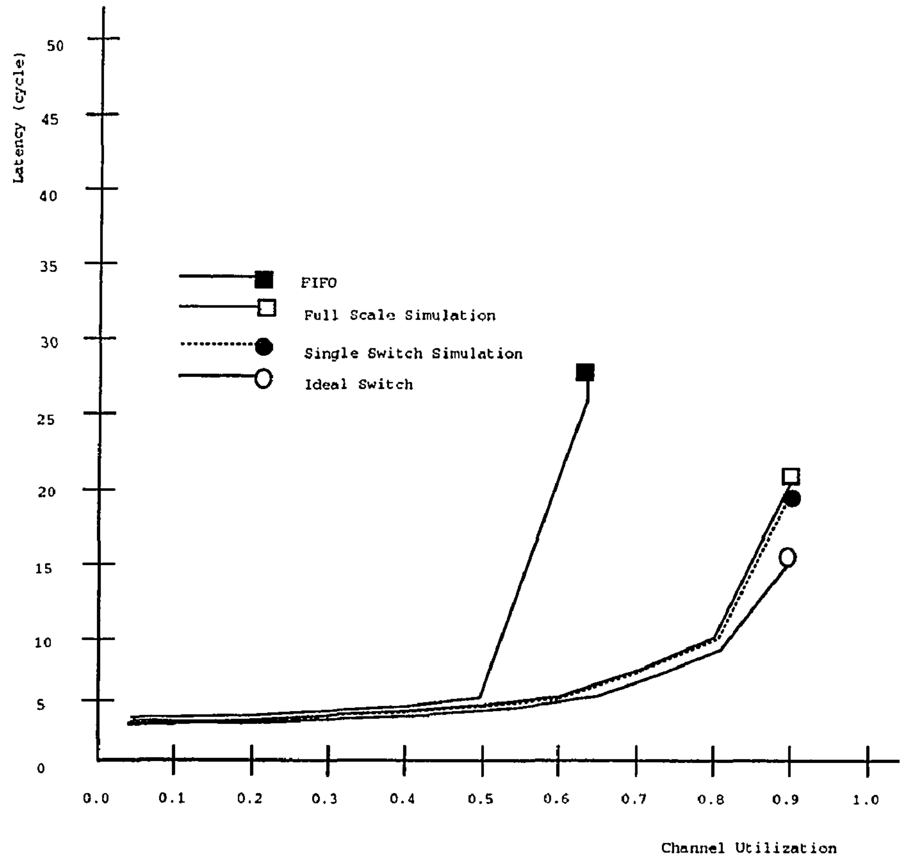


Figure 4.28: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for 4 x 4 Switch Omega Network with 64 Nodes (3 stages).

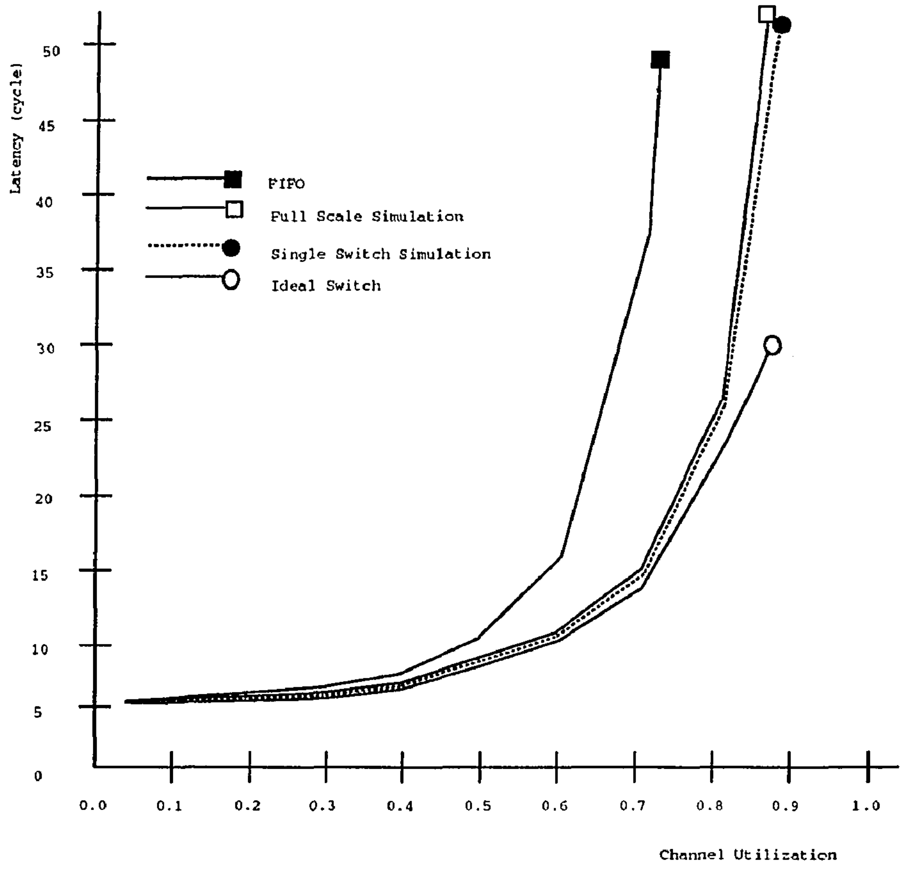


Figure 4.29: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for Omega Network (256 nodes). Packet size is 2 unit packets.

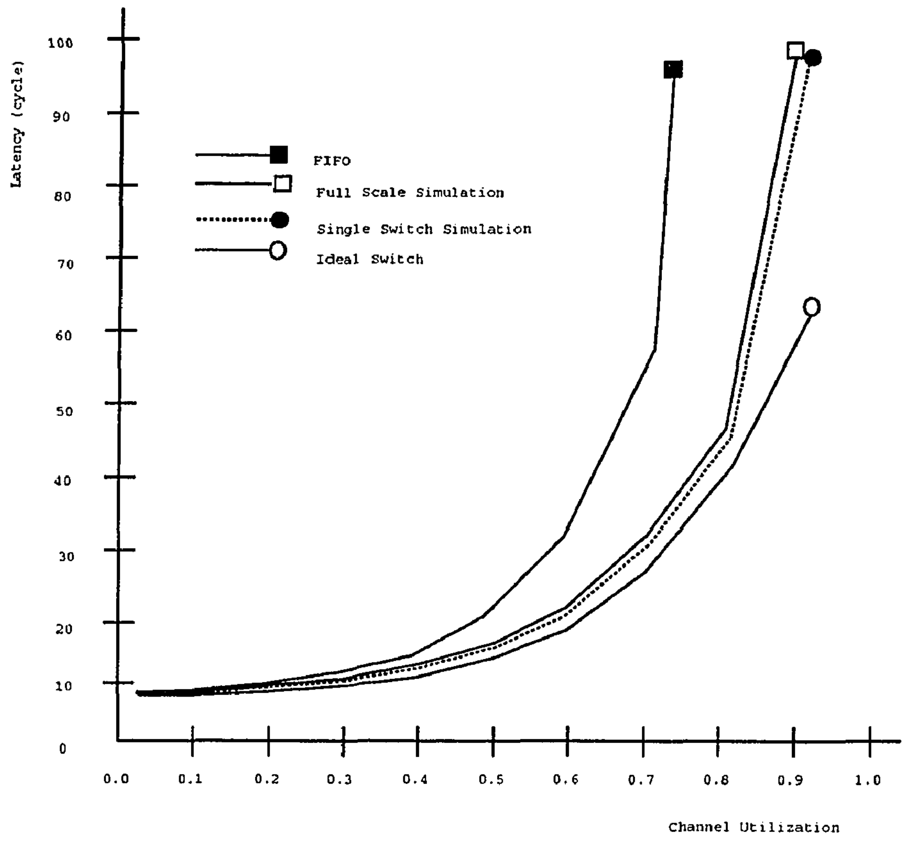


Figure 4.30: Comparing the Network Latency of DAMQ Scheme to the Ideal and FIFO Buffer Management Scheme for Omega Network (256 nodes). Packet size is 4 unit packets.

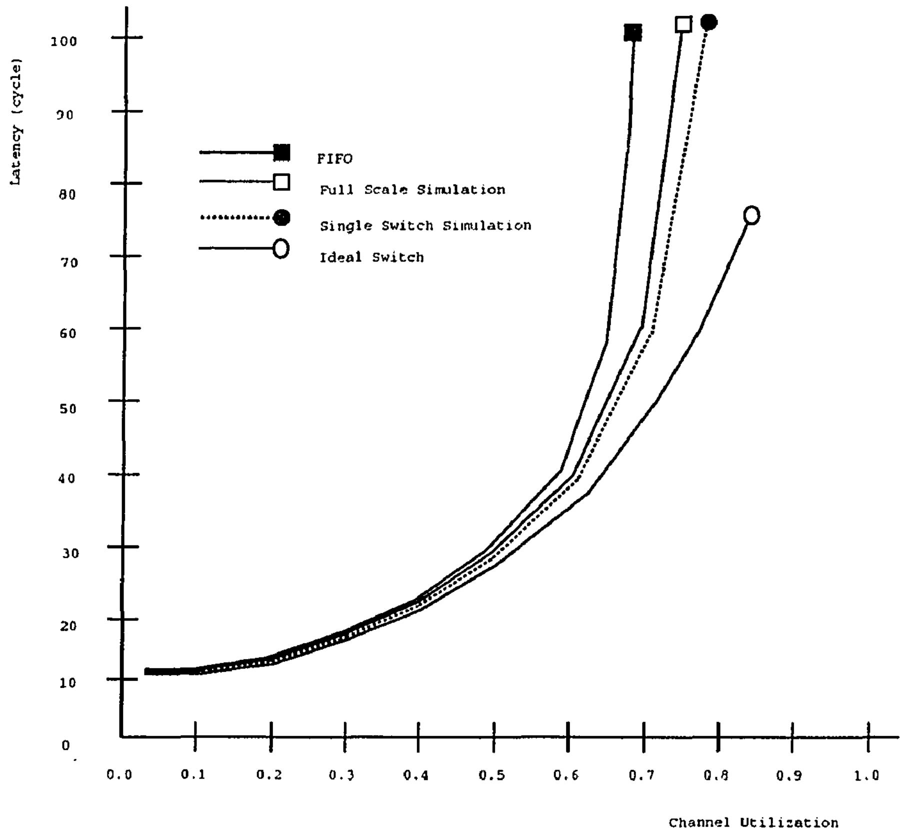


Figure 4.31: Comparing the Network Latency of DAMQ Buffer to the Ideal and FIFO Buffer Management Scheme for Omega network (256 nodes). Packet size is 8 unit packets.

packet size to vary from 8 bytes to 72 bytes. Within each range of packet sizes, a packet size had the same probability of being generated. The packet size of a fixed length of 40 bytes is the average packet size of random packet length. We also selected the block size of the DAMQ buffer to be 8 bytes so that there is no buffer fragmentation for fixed size packets. We ran three sets of simulation, first for fixed size packet, second for variable packet size with the DAMQ and the third for variable packet sizes but with FIFO buffers. Figure 4.32 shows the measurements of network throughput vs. latency on 16, 64 and 256 nodes. The throughput is the average number of packets per clock cycle arriving at the destination node. As it can be seen in Figure 4.32, the fixed packet size achieves higher throughput with lower latency. The throughput gain of fixed packet size gets higher as the throughput increases. This is due to the fact there can be a higher possibility of buffer fragmentation which results in lower utilization of buffers.

4.3 Conclusion

This chapter presented an efficient way to implement a high performance router using "self-compacting buffers". This technique offers the high performance possible with a Dynamically Allocated Multi-Queue, and requires less hardware than the alternative scheme proposed by Tamir and Frazier [8].

The second part presented extensive simulation results comparing the performance of a self-compacting DAMQ buffer against an ideal buffer. The comparison extends previous work by considering a much broader range of network topologies, including several examples of k-ary n-cubes and Delta networks. In all cases, the self-compacting buffer has performance comparable to an ideal buffer up to 80 %

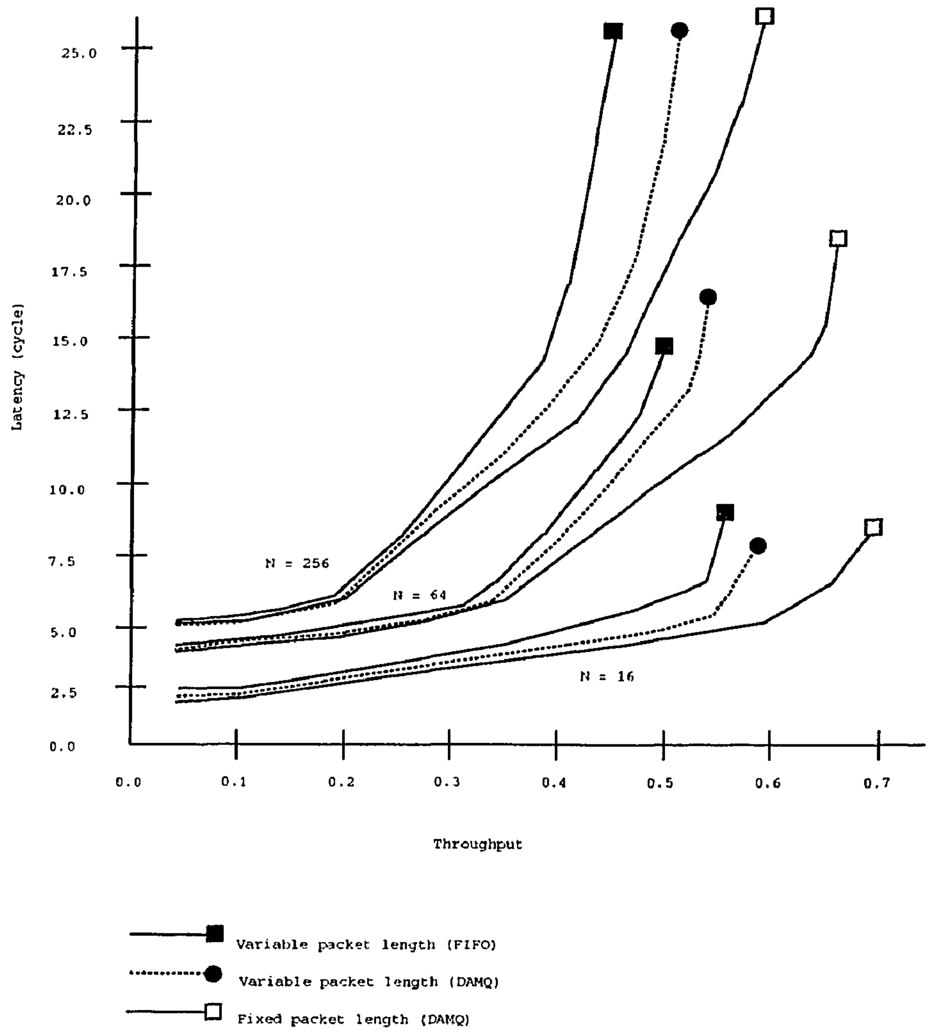


Figure 4.32: Comparison of DAMQ Buffer to FIFO with Random Packet Size for Omega Nework.

of the saturation bandwidth of the ideal buffer.

Additional simulation results showed how the performance of an entire network can be quickly and accurately approximated by simulating just a single router. The single router simulator required 10 to 100 times less simulation time, and was about 10 times smaller, than the full network simulator. As a specific example, the data plotted in Figure 4.19 took about 3 days of simulation time for the full networks, whereas single router simulation took several minutes.

CHAPTER 5

INPUT AND OUTPUT PORT CONTROLLER ARCHITECTURE AND ORGANIZATION

In this chapter we present an architecture of input/output port controllers for the flexible oblivious router. The basic function of the I/O port controller is to receive/transmit data between paired I/O ports. It is also required that the I/O port controller initiate the activities of router managing entities such as the routing algorithm handler and the packet flow controller. In addition, the I/O port controllers have the capability to receive/send variable length packets and a variable number of header packets. The capability of receiving/sending a variable number of header packets is critical for a router if the router is used in different network sizes. Lastly, the I/O port controller allows signals to be propagated between routers. This function was required by the router to support multiple switching techniques.

The remainder of the chapter is organized as following. In section one, a detailed description of the input port controller architecture is presented. In section two, the output port controller architecture is described. Finally, section three summarizes the chapter with a concluding remark.

5.1 Input Port Controller Architecture

As shown in Figure 5.1, the input port controller consists of a buffer, a protocol unit, a length counter, a header byte counter and several signals. Following

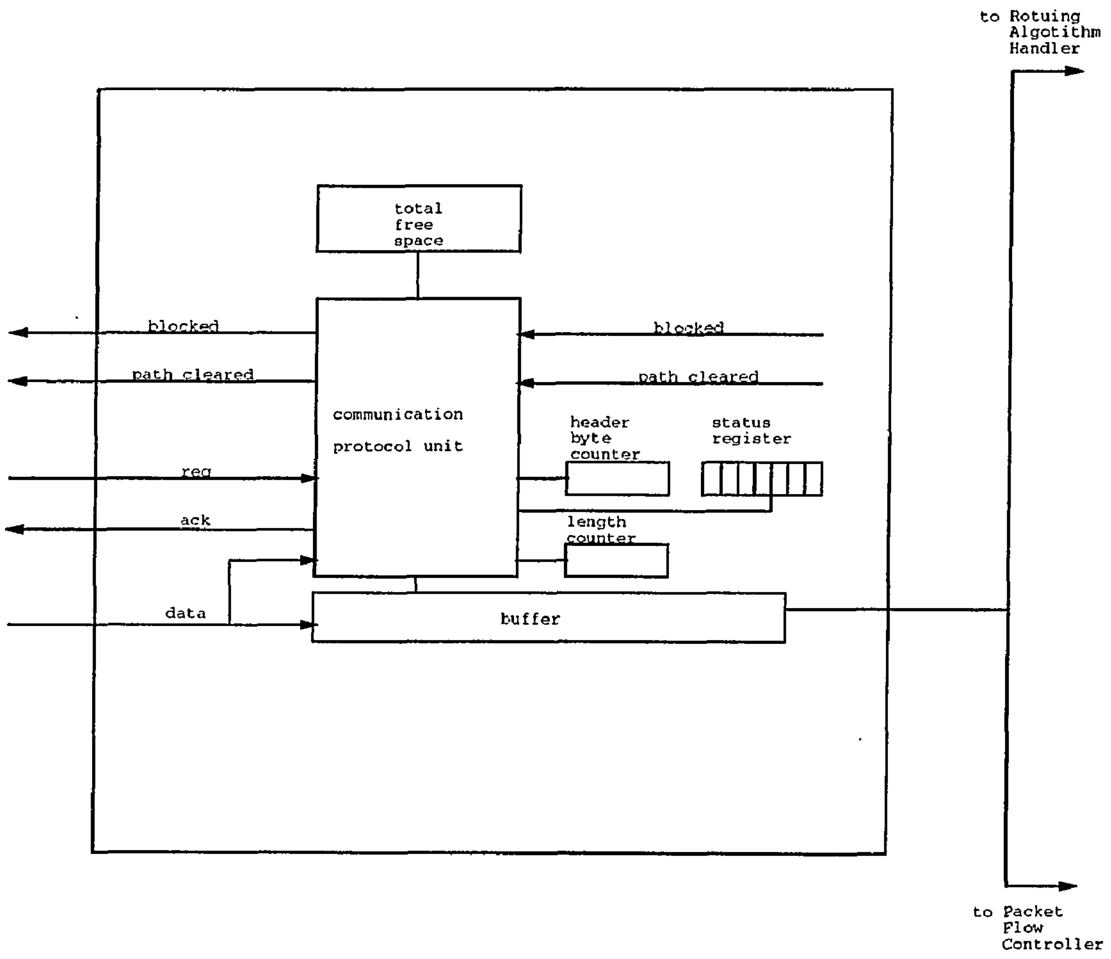


Figure 5.1: Logical Block Diagram of a Port Controller

are detailed descriptions for each of the modules.

Header Byte Counter: The header byte counter is used to support a variable size of header information. For a network with n processing nodes, it is required to have $\lceil log_2(n) \rceil$ bits in the header for routing information. If $log_2(n)$ is greater than the data bandwidth of the input port, we need t $(= \lceil \lceil log_2(n) \rceil / (data\ bandwidth) \rceil)$ bytes for routing information. The header byte counter stores the value t, which is used to extract the header information from the incoming packet and to transfer that header information to the routing algorithm handler.

Total Free Space: The total free space information is managed by the packet flow controller. This information is needed by the input port to determine whether it can receive more packets or not. If the size of incoming packets is greater than the value in the total free space, the input port rejects the request from the connected output port.

Length Counter: The length counter holds information about the size of the incoming packet. It is used as a counter to receive the packet and to recognize when the reception is done. When the router supports fixed length packets, the value in the length counter does not change for any incoming packet. However, it changes when each new packet is ready to be received and the router is operating under the variable length packet.

req(request): The output port that is connected to the input port uses the "req" signal to indicate that it has a packet to send.

ack(acknowledgment) : The input port uses the "ack" signal to give acknowledgment to the connected output port that it is ready to receive a new packet.

path cleared signal: The path cleared signal is used for supporting circuit switching. When all the paths between the source and the destination node are available, the path cleared signal is generated by the router of the destination node. This signal is back propagated to the source node. Upon receiving the path cleared signal, the source node starts transferring the packet. When the end of the packet is detected by the destination node, the path cleared signal is once again generated by the destination router. All input/output ports receiving the path cleared signal release the resource that they used to transfer the packet. Thus, the circuit is released.

Blocked Signal: The blocked signal is used to support the circuit switching as well. When a source node needs to set up a circuit, it sends a packet that has the destination address in it. As this packet travels through the paths that are part of the circuit, it may not be possible to have all paths available. If the path is not available, the blocked signal is generated from the router. This blocked signal is back propagated to the source node.

Status Register: The status register has two bits. The first bit is updated by the routing algorithm handler to indicate whether the state of the routing algorithm handler is in busy state or idle state. The second bit is managed by the packet flow controller to indicate whether the packet flow controller is ready to accept the data or not. The input port needs to know the state of the packet flow controller because it will not be able to receive any data until the routing algorithm handler determines the output channel number. The execution time of the routing algorithm will vary depending on the network topology and routing algorithm used for the network.

Buffer: The buffer is used to store packets when there is a mismatch of transmission/reception speed between the connected output port and the routing algorithm handler (or the packet flow controller). Part of the packet will be stored in the buffer temporarily. This will occur when the routing algorithm handler (or packet flow controller) is slower to receive the incoming packets than the outgoing packets are being transmitted.

Communication Protocol Unit: The communication protocol unit is the engine of the input port. The action of the communication protocol unit is initiated upon the reception of the request signal that comes along with the packet length count. Figure 5.2 shows how the "req" and the "ack" signals are interacting for packet reception and transmission. When the request signal arrives, the communication protocol unit checks to see if the routing algorithm handler is ready to execute its routine to determine the address of the next switch. If the routing algorithm handler is ready, then the communication protocol unit compares the length count value against the total free space. If the total free space is greater than the size of new a packet, the communication protocol unit grants the connection to the output port. If there is not enough space available, it continues comparisons until the total free space is greater than the size of the new packet. Once the acknowledgment is sent out to the output port, the input/output port starts reception/transmission of packet. The format of the packet is shown in Figure 5.3, it shows the first data coming in are the header information (length information is already sent with "req" signal). The communication protocol unit uses the header byte counter to determine how many bytes out of the incoming data it is supposed to transfer to the routing algorithm handler. After transferring header bytes, the communication protocol unit checks to see if the packet flow controller is ready to

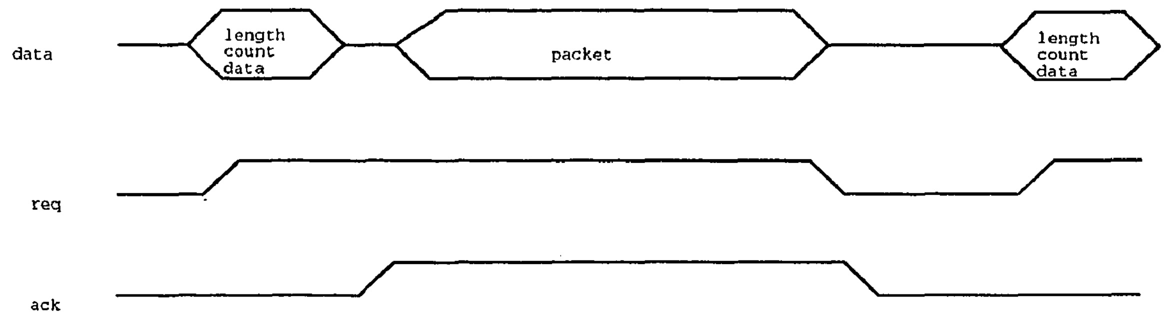


Figure 5.2: Input/Output Port Protocol

receive the rest of the packet. It uses the status register to check the state of the packet flow controller. If the packet flow controller is ready to accept the data, the communication protocol controller transfers the rest of the packet to the packet flow controller. Otherwise, it buffers the incoming packet and keeps checking the availability of the packet flow controller. After it completes the packet transfer, it lowers the "ack" signal to indicate the end of packet reception.

5.2 Output Port Controller Architecture

Header Byte Counter: The usage of the header byte counter in the output port is the same as in the input port.

Length Register: The length register stores length information of the packet. This information is extracted and stored in the length register while the packet is arriving at the output port and sent out to the connected input port as part of the communication protocol.

reql (request) signal: The "reql" signal is used by the crossbar switch to request the connection to the output port.

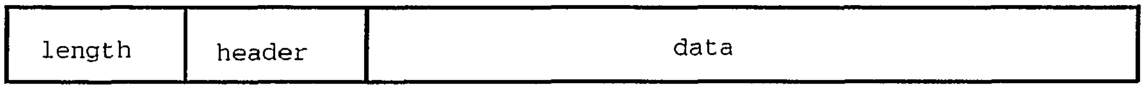


Figure 5.3: Packet Format

req2 (request) signal: The "req2" signal is used by the output port to request the connection to the connected input port.

ack1 (acknowledgment) signal: The output port uses an "ack1" signal to let the crossbar switch know that it is ready to receive the packet.

ack2 (acknowledgment) signal: The "ack2" signal is raised by the connected input port and indicates that the output port can send packets.

path cleared signal: The usage of this signal is the same as described in the input port.

blocked signal: The usage of this signal is the same as described in the input port.

Communication protocol unit: The communication protocol unit of the output port controls the packet transmission/reception between the crossbar switch and the output port and also between the output port and the connected input port. The same communication protocol shown in Figure 5.2 and the packet format shown in Figure 5.3 are used. The communication protocol unit overlaps the reception of the packet from the crossbar switch with the transmission of the packet to the input port for fast packet delivery. The communication protocol unit is activated by the "req1" signal from the crossbar switch. If the output port is in idle state, the communication protocol unit sends the "ack1" to the crossbar switch to tell that it is ready to receive the packet. If the output is in busy state, the "ack1" signal is held low until the state of the output port becomes the idle state. The first information arriving out of the packet is the length data. This information is latched into the length counter. As soon as the communication protocol unit receives the length information, it starts establishing the connection with the input port by raising the "req2" signal. When the communication protocol unit receives the "ack2" signal from the input port, it transfers the packet to the input port if it is available. Figure 5.4 shows a logical block diagram of an output port controller and Figure 5.5 shows an example connection of the I/O port for Delta network of size 8. It utilizes a 2 x 2 switching element with 8 bits per data line.

5.3 Conclusion

In this chapter, we presented an I/O port controller architecture for a flexible oblivious router. Besides the function of basic communication protocol, the I/O port controller has three important functions that include:

- The ability to support variable packet lengths.
- The ability to support a variable number of header packets.
- Provide a signal propagation capability to support circuit and packet switching.

These three functions are crucial for a router that supports variable length packets and multiple switching techniques. In particular, the ability to support a variable number of packets allows the router to be used in many different networks and broadens the area of the router's application.

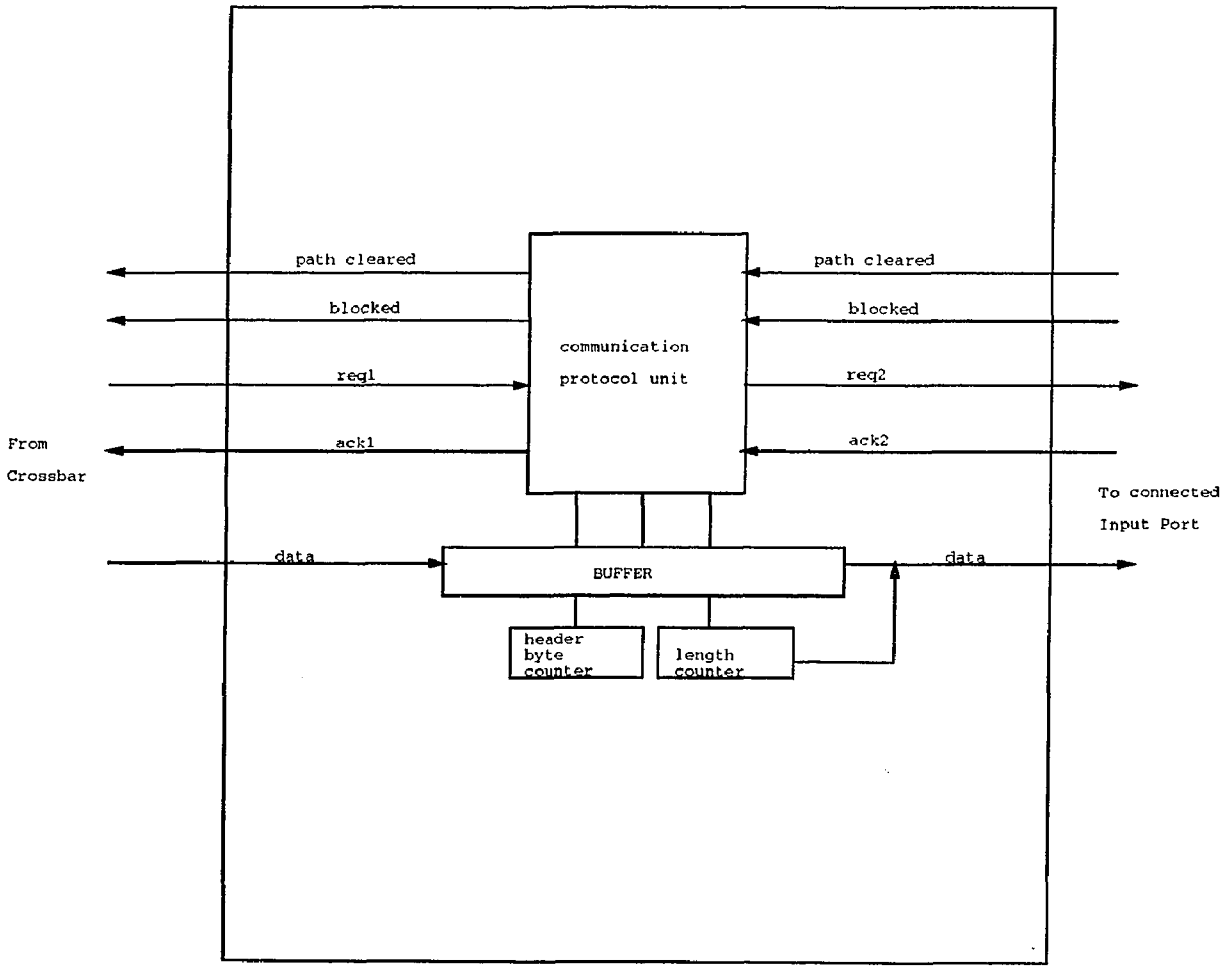


Figure 5.4: Logical Block Diagram of an Output Port

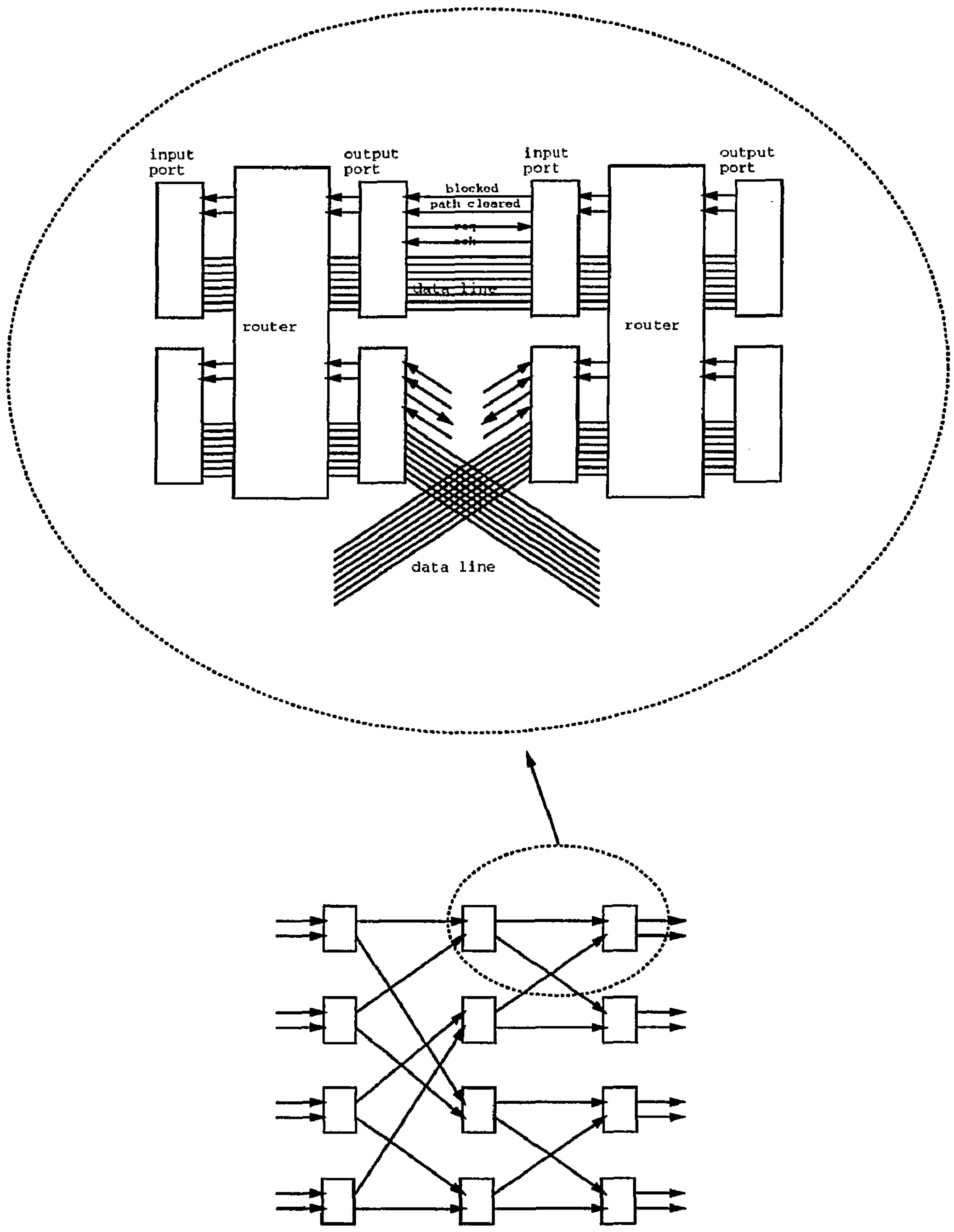


Figure 5.5: Input/Output Port in 2 x 2 Baseline Network

CHAPTER 6

CONCLUSION AND FUTURE STUDY

This dissertation provided the result of our study on a flexible router architecture. The contributions of this work include proposing a flexible oblivious router architecture and evaluation of a DAMQ buffer. This section summarizes the contributions of this dissertation.

Chapter 2 provided background surveys of topics and issues related to the design of a router. It included network topology, switching techniques, routing algorithms, flow control and virtual channels along with examples.

In chapter 3, we have presented a router architecture that can support a large set of oblivious routing algorithms. We have studied the routing algorithms of over 40 interconnection networks and identified the common functions, and the instruction set that is necessary to execute all the routing algorithms [11]. The architecture is designed to have a programming capability so that other oblivious routing algorithms not considered in our investigation can be supported as well. The programming capability also allows routing algorithms to be modified if any error in the routing algorithm is found or a better algorithm is developed later. Because the architecture supports a wide range of interconnection networks, it can be mass-produced and has the potential of being an "off-the-shelf" product. The overall conclusion is that general purpose cost effective routers can be designed, suggesting the possibility of a common router for multiple interconnection networks.

In chapter 4, we presented a novel approach of implementing the DAMQ buffer with a technique called "self-compacting buffer" and its evaluation. This technique requires less hardware than the alternative scheme proposed by Tamir and Frazier [8]. We also presented extensive simulation results comparing the performance of a self-compacting DAMQ buffer against an ideal and FIFO buffer. The comparison extended previous work which was done only on a 64 node Omega network. We have considered a much broader range of network topologies and sizes including several examples of k-ary n-cubes and Delta networks. In all cases, the self-compacting buffer has performance comparable to an ideal buffer. In addition, we showed how the performance of an entire network can be quickly and accurately approximated by simulating just a single switching element. The single switch simulator required 10 to 100 times less simulation time, and was about 10 times smaller, than the full network simulator. One specific example showed that the data plotted in Figure 4.20 took about 3 days of simulation time for the full network simulator, whereas single switch simulation took several minutes.

In chapter 5, we presented an I/O architecture for the flexible oblivious router. Besides the function of basic communication protocol, the I/O port controller had three important functions these include:

- The ability to support variable packet lengths.
- The ability to support a variable number of header packets.
- Provide a signal propagation capability to support circuit and packet switching.

These three functions are crucial for a router that supports variable length packets and multiple switching techniques. In particular, the ability to support a variable number of packets allows the router to be used in many different sizes of network and widens the areas of router application.

6.1 Major Contributions

The major contributions of this study can be summarized as the following:

- The first part of this research sought the possibility of proposing a general purpose router architecture.
- Common functions and an instruction set required to function with multiple interconnection networks have been identified.
- A novel approach called a "self-compacting buffer" to implement the DAMQ buffer has been developed.
- The performance of the DAMQ buffer on a broad range of networks including k-ary n-cubes and Delta networks was studied and extensively reported in this work.
- A single router simulation that is simple, fast and accurate has been proposed, implemented and demonstrated.
- An I/O port controller architecture that can support multiple switching techniques, variable length packets and a variable number of header packets was proposed.

6.2 Future Research Directions

This section suggests areas for future work to complement this study. Our short term goal is to develop the proposed architecture into hardware, and then evaluate its performance on a real multiprocessor system. For a long term goal, we suggest continued study to determine the possibility of a router architecture that supports adaptive routing. In our study, we concentrated on developing a router architecture based on oblivious routing. Overall, adaptive routing has the potential to outperform the oblivious routing. Chapter 3 has shown our approach for oblivious routing. A similar approach for adaptive routing would be possible.

There are several ways in which the studies introduced in chapter 4 can be extended. First, the single router simulation technique can be applied to other network topologies and router organizations in which the routing probabilities and channel utilizations are symmetric. Second, a real router design would be pipelined over several clock cycles. A performance study considering this effect could provide better information about the benefits of that design. Third, it would be interesting to compare the performance of wormhole flow control versus virtual cut-through for a range of finite buffer sizes.

BIBLIOGRAPHY

[1] B. W. O'Krafka, D esign and Evaluation of Directory-Based Cache Coherence Systems, *Ph.D. Thesis, University of California, Berkeley, 1992*

[2] W. J. Dally and C. L. Seitz, "The torus routing chip," *J. Distributed Systems*, vol. 1, No.3, pp. 187–196, 1986.

[3] Intel Scientific Computers, A Technical Summary of the iPSC/2 Concurrent Supercomputer, Order No. 280115-001, 1988.

[4] C. L. Seitz, "The Cosmic Cube," *CACM*, vol 28, No.1, pp. 22–33, 1985.

[5] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H.T. Kung, M. Lam, B. Moor, C. Peterson, J. Pieper, L. Rankin, P.S. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iWARP: An integrated solution to high-speed parallel computing," *Proc. Supercomput. Conf., IEEE*, pp. 330–338, November 1988.

[6] InMOS Ltd., "The 9000 Transputer Products Overview Manual," Order Code: DBTRANSPST/1 1991.

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

[8] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol.14, No. 6, pp. 725–737, 1992

[9] P. Kermani and L. Kleinrock, "Virtual cut through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[10] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. on Computers*, vol. C-36, No.5, pp. 547–553, May 1987.

[11] J. Park, S. Vassiliadis and J. G. Delgado-Frias, "Flexible Router Architecture: Instruction set and Organization," *IBM Technical Report TR 01.C752*, pp. 1–102, September, 1993.

[12] J. Park, S. Vassiliadis and J. G. Delgado-Frias, "Flexible Router Architecture," *IBM Technical Report TR 01.C749*, pp. 1–35, September, 1993.

[13] J. Park, S. Vassiliadis, B.W. O'krafka and J. G. Delgado-Frias, "Design and Evaluation of a DAMQ Multiprocess Network Switch with Self-compacting Buffer," *IBM Technical Report*, May 1994.

[14] R. Arlauskas, "iPSC/2 system: A second generation hypercube," *Third Conf. Hypercube Concurrent Comput. and Appl., ACM*, pp. 33–36, 1988.

[15] W. C. Athas and C. L. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Comput. Mag.,* vol. 21, pp. 9–24, August 1988.

[16] BBN Advanced Computers Inc., "Butterfly parallel processors overview," *BBN Rep. 6148,*, March 1986.

[17] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems,* vol. 3, No. 2 pp. 194-205, March 1992.

[18] W. J. Dally, "Network and processor architecture for message-driven computers," *VLSI and Parallel Computation*, Suaya and Birtwhistle, Eds. Los Altos, CA: Morgan Kaufmann, 1990.

[19] C. L. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-29, No.8, pp. 694–702, Aug. 1980.

[20] K. Whang and F. A. Briggs, "Computer Architecture and Parallel Processing," *MaGraw-Hill Book Company*, 1987.

[21] M. J. Quinn, "Designing Efficient Algorithms for Parallel Computers," *MaGraw-Hill Book Company*, 1984.

[22] E. Horowitz and A. Zorat, "The Binary Tree as Interconnection Network: applications to multiprocessor systems and VLSI," *IEEE Trans. on Computers*, vol. 30, No.4, pp. 247–253, April 1981.

[23] C. E. Leiserson, "Fat-tree: universal networks for hardware-efficient supercomputing," *Proc. of 1985 Intl. Conf. Parallel Processing*, pp. 393–402, 1985.

[24] F.J. Meyer and D.K. Pradhan, "Flip-tree: fault-tolerant graphs with wide containers," *IEEE Transaction on Computers*, vol.37, No.4, pp. 472–478, April 1988.

[25] J.R. Goodman and C.H. Sequin, "Hypertree: a multiprocessor interconnection topology," *IEEE Transaction on Computers*, vol.30, No.12, pp. 923–933, Dec. 1981.

[26] B.L. Menezes and R. Jenevein, "The KYKLOS multicomputer network and its message traffic," *IEEE Transaction on Computers*, vol.34, No.8, pp. 765–768, Aug. 1985.

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

[28] A. DeCegama, Parallel Processor Architectures and VLSI Hardware, Englewood Cliffs, New Jersey, Prentice Hall, 1989.

[29] C. P. Kruskal and M. Snir, "A unified theory of interconnection network structure," Ultracomputer Note ♯106.

[30] W. Lin and C. L. Wu, "Reconfiguration Procedures for a Polymorphic and Partitionable Multiprocess," *IEEE Transactions on Computers*, vol. C-35, No. 10, pp. 910–916, Oct. 1986.

[31] Y. Tamir and G. L. Frazier, "Hardware Support for High-Priority Traffic in VLSI Communication Switches," *Journal of Parallel and Distributed Computing*, vol. 14, No.4, pp. 402–416, April 1992.

[32] J. W. Dolter, P. Ramanathan, and K. G. Shin, "A Microprogramable VLSI Routing Controller for HARTS," *Intern. Conf. on Computer Design: VLSI in Computers and Processors*, Cat. No. 89CH2794-6, pp. 160-163, October 1989.

[33] C. Y. Chang, T. W. Hou and C. K. Shieh, "Design of Wormhole Router for Distributed Memory Multiprocessor," *Electronic Letters*, vol. 27, No. 25, pp. 2385–2387, 1991.

[34] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol. 41, No.6, pp. 725–737, June 1992.

[35] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transaction on Computers*, vol.39, No.6, pp. 775–785, 1990.

[36] H. Sullivan and T. R. Brashkow, "A large scale homogeneous machine," *in Proc. 4th Annu. Symp. Comput. Architecture*, pp. 105–124, 1977.

[37] D. H. Linder and J. C. Harden, "An adaptive and Fault-Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, vol. 40, No. 1, pp. 2–12, January 1991.

[38] W. J. Dally and H. Akoi, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, No. 4, pp. 466–475, April 1993.

[39] H. C. Chi and Y. Tamir, "Decomposed Arbiters for Large Crossbars with Multi-Queue Input Buffers," *IEEE Inter. Conf. on Comput Design: VLSI in Computers and Processor,* Cat. No. 91CH3040-3, pp. 233–238, 1991.

[40] K. Choi and W. S. Adams, "VLSI Implementation of a 256 x 256 Crossbar Interconnection Network," *Proceedings. Sixth International Parallel Processing Symposium,* Cat. No. 92TH0419-2, pp. 289–293, 1992.

[41] M. K. Vernon and U. Manbar, "Distributed round-robin and first-come-first-serve protocols and their application to multiprocessor bus arbitration," *in Proc. 15th Annu. Int. Symp. Comput. Architecture,* pp. 267–277, 1988.

[42] T. Lang and M. Valero, "M-users B-servers arbiter for multiple-buses multi-processors," *Microprocessing and Microprogramming,* pp. 11–18, Oct. 1982.

[43] S. Nugent, "The iPSC/2 Direct-Connect Communications Technology," *Third Conf. on Hypercube Concurrent Comput. and Applications,* vol. 1, pp. 51–60, Jan. 1988.

[44] S. Konstantinidou, "Chaos router: architecture and performance," *Comput. Archit. News,* vol. 19, No.3 pp. 212–221, May 1991.

[45] M. S. Chen and K. G. Shin, "Addressing, routing and broadcasting in hexagonal mesh multiprocessors," *IEEE Transaction on Computers*, vol.39, No.1 pp. 10–18, Jan. 1990.

[46] Seitz, C.L. Athas, W.C. Flaig, C.M. Martin, A.J. Seizovic, J. Steele and C.S. Wen King Su, "The architecture and programming of the Ametek Series 2010 multicomputer," *Third Conference on Hypercube Concurrent Computers and Applications*, vol.1, pp. 33–36, 1988.

[47] Peterson, C. Sutton, J. and Wiley, P., "iWarp: a 100-MOPS, LIW microprocessor for multicomputers," *IEEE Micro*, vol.11, No.3 pp. 26–29, 81–89, June 1991.

[48] Dally, W.J. Chien, A. Davison, R. Fiske, J.A.S. Furman, S. Fyler, G. Gaunce, D.B. Horwat, W. Kaneshiro, S. Keen, J.S. Lethin, R.A. Noakes, M. Nuth, P.R. Spertus, E. Totty, B. Wallach, D. and Wills, D.S., "The J machine: a fine grain parallel computer," *Symposium on High Performance Computing for Flight Vehicles*, vol.3, No. 1-4, pp. 7–15, 1992.

[49] Lillevik, S.L., "DELTA: a 30 gigaflop parallel supercomputer for Touchstone," *Northcon. Conference Record*, pp. 294–304, 1990.

[50] B. W. Arden and H. Lee, "Analysis of chordal ring network," *IEEE Transaction on Computers*, vol.30, pp. 291–296, 1981.

[51] W. J. Dally, "Virtual-Channel Flow Control," *In Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 60–68, May 1990.

[52] W. K. Tsai, Y. C. Kim and N. Bagherzadeh, "A hierarchical mesh architecture," *Proc. 4th Annual Parallel Processing Symposium*, pp. 923–933, 1990.

[53] N. S. Woo and A. Agrawala, "A symmetric tree structure interconnection network and its message traffic," *IEEE Transaction on Computers*, vol.34, No.8, pp. 765–768, Aug. 1985.

[54] F. T. Leighton, "New lower bound techniques for VLSI," *Math. Syst. Theory*, vol.17, No.1, p. 47, 1984.

[55] P. Banerjee, "Algorithm-based fault tolerance on a hypercube multiprocessor," *IEEE Transaction on Computers*, vol.39, No.9, pp. 1132–1144, 1990.

[56] A. S. Youssef and B. Narahari, "The banyan-hypercube networks," *IEEE Transaction on Parallel and Distributed Systems*, vol.1, No.2, pp. 160–169, 1990.

[57] P. W. Dowd and K. Jabbour, "Spanning multiaccess channel hypercube computer interconnection," *IEEE Transaction on Computers*, vol.37, No.9, pp. 1137–1142, 1988.

[58] N. Tanabe, T. Suzuoka, and S. Nakamura, "Base-m n-cube high performance interconnection networks for highly parallel computer prodigy," *1991 Intl. Conf. Parallel Processing*, pp. I509–I516, 1991.

[59] L. D. Wittie, "Communication structures for large networks of microcomputers," *IEEE Transaction on Computers*, vol.30, No.4, pp. 264–273, April 1981.

[60] L. N. Bhuyan and D. P. Agrawal, "A general class of processor interconnection strategies," *Procd. 9th Ann. Symp. Computer Architecture*, vol.39, No.1, pp. 10–18, 1990.

[61] V. Benes, "Optimal Rearrangeable multistage connection networks," *Bell System Technical Journal*, vol.40, No.4 Pt. 2, pp. 1641–1656, 1964.

[62] R. J. McMillen and H. J. Siegel, "Routing schemes for the augmented data manipulate network in an MIMD system," *IEEE Transaction on Computers*, vol.31, No.12 pp. 1202–1214, December 1982.

[63] G. B. Aams and H. J. Siegel, "The extra stage cube:a fault-tolerant interconnection network for supersystems," *IEEE Transaction on Computers*, vol.31, No.5 pp. 247–253, 1982.

[64] V. Cherkassky and et al., "Reliability and fault diagnosis of fault tolerant multistage interconnection networks," *Proc. 14th Fault Tolerant Computers Symp*, pp. 246–251, June 1984.

[65] D. S. Parker and C. S. Raghavendra, "The gamma network," *IEEE Transaction on Computers*, vol.33, No.4 pp. 367–373, 1984.

[66] A. El-Amawy and S. Latifi, "Properties of and performance of folded hypercubes," *IEEE Transaction on Parallel and Distributed System*, vol.2, No.1, pp. 31–42, 1991.

[67] S. P. Dandamudi and D. L. Eager, "Hierarchical interconnection networks for multicomputer systems," *IEEE Transaction on Computers*, vol.39, No.6, pp. 786–797, 1990.

[68] N. Pippenger, "On crossbar switching networks," *IEEE Transaction on Communications*, vol.COM-23, pp. 646–659, 1975.

[69] P. K. Bansal, K. Singh and R. C. Joshi, "Quad tree: a cost effective fault tolerant multistage interconnection," *IEEE INFOCOM '92: Conf. on Computer Communications*, vol.2, pp. 860–866, 1992.

[70] J. Park, S. Vassiliadis and J. G. Delgado-Frias, "Input and Output Port Controller Architecture and Organization," *IBM Technical Report*, 1994.

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

[72] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Transaction on Parallel and Distribution*, vol.2, No.4, pp. 398–412, 1991.

[73] S. Abraham and K. Padmanahan, "Performance of the Direct Binary n-Cube Network for Multiprocessors," *IEEE Transaction on Computers*, vol.38, No.7, pp. 1000–1011, July 1991.