1997

Design and Analysis of Optical Interconnection Networks for Parallel Computation.

Yueming Li
Louisiana State University and Agricultural & Mechanical College

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.

UMI

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

DESIGN AND ANALYSIS OF OPTICAL INTERCONNECTION NETWORKS FOR PARALLEL COMPUTATION

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in Partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

By
Yueming Li
B.S., University of Science & Technology Beijing, 1982
M.S., University of Science & Technology Beijing, 1984
M.S., South Dakota State University, 1994
December 1997

Acknowledgments

First and foremost, I am especially grateful to my major professor, Dr. S.Q.Zheng. This work would have been impossible without his guidance, patience, encouragement and assistance.

I would like to thank the members of my doctoral committee, Dr. Jerry Trahan, Dr. Doris L. Carver, Dr. John Drilling, Dr. S. Sitharama Iyengar, and Dr. Xianhe Sun for their comments, suggestions and instructions which help improve the quality of this work.

I would like to thank other staff and faculty in the Department of Computer science for their help and assistance during my studies at LSU.

I would like to thank my friends, Guodong Qin, for his great help and numerous errands while I was away from campus, and Xin Liangyang and his wife for the lodging and delicious food when I defended this work. Thanks to all my other friends for their friendship.

To my parents, Tianhuan Li and Zhongdi Li, thanks for your persistent encouragement and support.

To my parents-in-law, Yiqian Liu and Shuping Chen, thanks for your enormous help and for your picking up the responsibility to take care of my son.

To my wife, Xiaojiang Liu, thanks for your love, understanding and never-ending push. Also thanks for all the duties that would have been mine if I had not been that busy due to this work.

To my son, Danxiang Li, thanks for your understanding why your Dad was not with you for years.

Contents

List of Tables

List of Figures

Abstract

In this doctoral research, we propose several novel protocols and topologies for the interconnection of massively parallel processors. These new technologies achieve considerable improvements in system performance and structure simplicity.

Currently, synchronous protocols are used in optical TDM buses. The major disadvantage of a synchronous protocol is the waste of packet slots. To offset this inherent drawback of synchronous TDM, a pipelined asynchronous TDM optical bus is proposed. The simulation results show that the performance of the proposed bus is significantly better than that of known pipelined synchronous TDM optical buses.

Practically, the computation power of the plain TDM protocol is limited. Various extensions must be added to the system. In this research, a new pipelined optical TDM bus for implementing a linear array parallel computer architecture is proposed. The switches on the receiving segment of the bus can be dynamically controlled, which make the system highly reconfigurable.

To build large and scalable systems, we need new network architectures that are suitable for optical interconnections. A new kind of reconfigurable bus called segmented bus is introduced to achieve reduced structure simplicity and increased concurrency. We show that parallel architectures based on segmented buses are versatile by showing that it can simulate parallel communication patterns supported by a wide variety of networks with small slowdown factors.

New kinds of interconnection networks, the hypernetworks, have been proposed recently. Compared with point-to-point networks, they allow for increased resource-sharing and communication bandwidth utilization, and they are especially suitable for optical interconnects. One way to derive a hypernetwork is by finding the dual of a point-to-point network. Hypercube Q_n, where n is the dimension, is a very popular point-to-point network. It is interesting to construct hypernetworks from the dual Q_n^* of hypercube of

Q_n. In this research, the properties of Q_n^* are investigated and a set of fundamental data communication algorithms for Q_n^* are presented. The results indicate that the Q_n^* hypernetwork is a useful and promising interconnection structure for high-performance parallel and distributed computing systems.

Chapter 1

Introduction

Many real-life problems such as weather forecast modeling, molecular modelling, computer-aided design of VLSI circuits, large-scale database management, artificial intelligence, and strategic defense initiatives are computation intensive. The circuit density of a single chip is approaching the limit. This means the processing speed of a single processor is reaching the limit. To further increase the speed, we must turn to parallel computing. Therefore, the importance of parallel computing for real life application is obvious.

In parallel computing, time complexities of many computation problems are constrained by data movement among processors. Designing efficient interconnection networks has long been a focus in parallel computing research. To resolve this communication bottleneck, researchers have been trying to invent better network topologies and to find new transmission media that have bigger bandwidth.

Various network architectures have been proposed. These networks can be classified as static connection networks and dynamic connection networks. Typical static connection networks include linear array, ring, binary (fat) tree, star, mesh, torus, hypercube, cube-connected cycles (CCC) and k-ary n-cube [34]. Expected features of those static networks include small and constant node degree, small network diameter, symmetry, and scalability. With regard to dimensionality, low-dimensional networks reduce contention because having a few high-bandwidth channels results in more wire sharing and thus a better queuing performance than having many low-bandwidth channels. In addition, low-

dimensional networks have a higher maximum throughput and lower average block latency than high-dimensional networks. It has been argued that ring, mesh, torus, k-ary n-cube, and CCC all have some desirable features for building future MPP systems.

Dynamic connection networks include bus systems, multistage interconnection networks (MINs) and crossbar switch networks. Characteristics of dynamic networks include minimum latency, bandwidth per processor, wiring complexity, switching complexity, and connectivity and routing capability. The high bandwidth and routing capability are usually at the cost of high wiring and switching complexity.

Designing networks connecting a large number of processors that support massively parallel computing is a challenge. As discussed above, in designing a network for interprocessor communication, one has three major choices: point-to-point networks, MINs and multiple-bus systems (also see [18, 40, 71] for surveys). MINs suffer from flexibility and a relatively large lower bound for propagation delay through the stages of the network. Most previously investigated multiple-bus schemes are either restricted to processor-memory interconnections (shared-memory architecture) or proposed to augment point-to-point interprocessor networks for improved broadcast and multicast performance in the electrical domain. Point-to-point networks have poor resource sharing, especially when dimensionality is high. One can see the trade-offs in designing an efficient network from the above discussion. The trend is low-dimensional networks, which is consistent with the above discussion. This observation is evidenced by several most recently implemented or proposed network architectures, which include the multiple buses in the Wisconsin multicube [23], Orthogonal buses in the OMP [31, 33], sparse torus in the Tera computer [4], and the fat tree used in CM-5 machine [34].

The most common transmission media has been copper wire for electronic interconnection. However, due to the bandwidth limit of electronic interconnections, researchers are turning their attention to optical interconnections. Several approaches, such as free space optical devices, wavelength-division multiplexing (WDM) and time-division multi-

plexing (TDM), of constructing optical interconnections for multiprocessor systems have been studied, and the advances in optical devices and fiber optic communications have made the optical interconnections feasible. Prototypes of optical interconnections have been constructed and shown promising. New generation massively parallel computers using optical interconnections may become a reality in the near future.

Notice the interaction between the network topology and the transmission media. A new transmission medium usually requires new network topology for efficient communication. Traditional point-to-point networks may be suitable to electronic interconnection, but they may be not suitable for optical interconnection. The purpose of this research is to find better transmission protocols and network topologies that will result in more efficient parallel computation systems.

Optical waveguides can be used to implement a bus. Signal propagation in an optical bus is unidirectional and has predictable delay per unit length. Furthermore, an optical bus can connect more processors than an electrical bus. Processors connected by an optical bus are linearly ordered. Such a bus system is considered as a one-dimensional parallel computer architecture — a linear processor array. It is important to investigate linear arrays because they can be used as building blocks to construct parallel architectures of higher dimensions to achieve improved scalability and performance.

One class of promising parallel architectures are the distributed-memory SIMD (Single Instruction Stream Multiple Data Stream) computer systems equipped with pipelined optical buses using TDM access protocols. In such a system, the transmission latency between furthest processors is the end-to-end propagation delay of light over a waveguide. Since messages are transmitted concurrently in a pipelined fashion on a bus, this latency is hidden. Due to their remarkable advantages, such optical bus systems have received much attention (e.g. [26, 27, 41, 49, 55, 56, 57, 65]). More powerful multidimensional processor arrays connected by such optical bus systems have also been proposed recently (e.g. [59, 65]).

The TDM multiaccess methods can be divided into two categories: fixed assignment and demand assignment. All previously proposed pipelined buses use fixed assignment, for example, the *time-division source-oriented multiplexing* (TDSM) multiaccess method and *time-division destination-oriented multiplexing* (TDDM) multiaccess method. With TDSM, each packet slot is assigned to a source processor while with TDDM, each packet slot is assigned to a destination processor. The major disadvantage of the fixed-assignment TDM (FA-TDM) is the requirement that the packet slots for each processor are fixed regardless whether or not it has a packet to transmit. A demand-assignment TDM method (DA-TDM) allocates packet slots to processors dynamically according to their demands and the traffic situation.

In this research, we introduce the idea of a flagged packet. We then use this idea to modify the known pipelined optical bus structure so that a demand-assignment TDM multiaccess method using a linear priority scheme can be implemented by hardware. To improve the fairness of the DA-TDM multiaccess method, we incorporate reconfigurability into our pipelined optical bus to implement the round-robin priority scheme. The scheduling of the DA-TDM multiaccesses with the round-robin priority scheme is implemented by reconfiguring the bus in hardware. We compare the performance of our pipelined DA-TDM optical bus with the pipelined FA-TDM optical bus by simulations, in terms of average message response time and fairness. We also explore the potential of using our buses to construct multichannel switches and multidimensional processor arrays.

For parallel applications that are suitable for synchronous TDM, the computation power of the system in terms of parallelism and concurrency is often limited. It can be improved by introduction of reconfigurability. For example, it is proposed in [56] that programmable switches are added to the transmitting segments of the optical bus. An average case $O(\log N)$ and worst case $O(N)$ time complexities for parallel selection have been achieved on such a system. The idea of programmable delays using electrooptically switched fiber loops has been used in the designs of TDM time slot interchangers

(e.g. [30, 37, 66, 78]). Also, two optical bus structures with conditional delays have been proposed in [26] and [57]. In the bus structure of [26], switches are on the transmitting segment of the bus, and because of this, parallel computation on subarrays, if not impossible, is difficult. To solve this problem, an optical bus with considerably more switches was proposed in [57]. Such a bus can be physically partitioned into several buses for parallel subarray computation. In this research, we introduce the technique for binary prefix summation and processor reordering. Structurally, our system is very simple and cost-effective. For example, we demonstrate that our system is very powerful by showing that parallel selection can be done in optimal time on our system. We also show that using this linear array architecture several fundamental parallel communication and computation operations, which include broadcasting, multicasting, compaction, partition and concurrent subarray computation, can be carried out efficiently.

The realization of general-purpose, massively parallel computers hinges largely on being able to build scalable interprocessor networks. To build large and scalable systems, we need new network architectures that are suitable for optical interconnections. High-dimensional networks can not be candidates due to their poor wire sharing property which has been well studied during the last few years. On the other hand, low-dimensional networks show a better wire sharing property. To compensate the lost concurrency and parallelism resulting from the switch from high-dimensional networks to low-dimensional networks, it has been proposed that the low-dimensional networks be enhanced in some ways. For example, meshes can be enhanced by multiple buses to improve broadcasting performance [74, 60, 81]. A number of multiprocessors connected by multiple reconfigurable buses have also been proposed. Examples include, among many, the bus automaton [69], the reconfigurable mesh [51] and the polymorphic torus [42]. The common feature of these machine models is that the bus configurations can change under program control. Some of these models have been shown surprisingly powerful because of the dynamically reconfigurable communication paths achieved by enhancement. For example, by using bus

control techniques to reconfigure communication paths as an integral part of computation, it is shown in [36, 46, 53, 54] that n numbers can be sorted in constant time on an $n \times n$ reconfigurable mesh. One side effect of enhancement is that the resulting systems may be too complicated to implement. In this research, we are going to propose a class of reconfigurable buses, called segmented buses. A segmented bus connecting p processors, denoted by $B(p)$, is a bus that can be dynamically partitioned into several segments, each connecting a subset of processors, by switches. We also generalize the concept of segmented bus to obtain parallel architectures of higher dimensions, called k-dimensional mesh connected by segmented buses (k-D MCSB). We show that the segmented bus and the k-D MCSB are versatile parallel computing architectures by showing that they can simulate a wide variety of useful network structures. In particular, we show that $B(p)$ can simulate any linear array or ring of no more than p processors with a constant slowdown factor, and $B(p)$ can simulate a $(2p-1)$-processor complete binary tree, X-tree and one-dimensional multigrid with an $O(\log p)$ slowdown factor. Then, we use these results to show that a k-D MCSB can simulate a k-D mesh or torus with a constant slowdown factor, an $N \times N$ MCSB can simulate an $N \times N$ mesh-of-trees, an $N \times N$ multigrid network and $N \times N$ pyramid network with an $O(\log N)$ slowdown factor. It would not be complete without considering the algorithmic aspect of the segmented bus based architecture. We demonstrate the advantages of parallel architectures based on segmented buses by giving a parallel algorithm for the prefix computation problem.

Traditionally, interconnection networks are characterized by graphs. Network topologies under graph models have been extensively investigated. Many network structures have been proposed, and some have been implemented. Observing the improved electrical bus and switch technologies, and maturing optical interconnection technologies, Zheng pointed out that the conventional graph structure is no longer adequate for the design and analysis of the new generation interconnection structures and proposed a new class of interconnection networks, the hypernetworks [84].

The class of hypernetworks is a generalization of point-to-point networks, and it contains point-to-point networks as a subclass. In a hypernetwork, the physical communication medium (a hyperlink) is accessible to multiple (usually, more than two) processors. The relaxation on the number of processors that can be connected by a link provides more design alternatives so that greater flexibilities in trade-offs of contradicting design goals are possible. The underlying graph theoretic tool for investigating hypernetworks is hypergraph theory [8]. Hypergraphs are used to model hypernetworks. Existing results in hypergraph theory and combinatorial block design theory, which is closely related to hypergraph theory, can be used to design hypernetworks. For example, in [87], Zheng introduced several low diameter hypernetworks based on the concept of Steiner Triple System. In [88], Zheng and Wu proposed a scheme for constructing a new hypernetwork from an existing one using the concept of dual graph in hypergraph theory. They showed that the dual H^* of any given hypergraph H is a hypergraph that has some properties related to the properties of H so that one can investigate the properties of H^* based on the properties of H. Since the structure of H and its dual H^* can be drastically different, finding hypergraph duals can be considered as a general approach to the design of new hypernetworks. They investigated the structure of the dual K_n^* of an n-vertex complete point-to-point network K_n.

The hypercube is a popular point-to-point network that has many desirable features such as small diameter, symmetry, and support of a large class of efficient parallel algorithms. In this research, we propose a class of hypernetworks, the Q_n^* (read as Q_n star) hypernetworks. The Q_n^* hypernetwork is the dual of the n-dimensional hypercube Q_n. We discuss the topological properties and fault tolerance aspects of Q_n^*, and present a set of parallel data communication algorithms for Q_n^*.

Chapter 2

Optical Buses

Recently, optical technologies for interconnection of massively parallel computers have received considerable attention. Optics can utilize free-space as well as guided wave technologies. There are many desirable characteristics of optical interconnections such as high speed, high bandwidth, increased fanout, longer interconnection lengths, low power requirements, and reduced crosstalk. These characteristics have significant system configuration and complexity implications.

2.1 Non-pipelined Optical bus

The refraction property of light makes it possible to be transmitted in a fiber. When a light ray is sent from one substance to another, some of it is reflected and some passes into the new substance, referring to Figure 2.1. The ray of light getting into the new substance is usually bent from its original angle. The extent to which the ray bends depends on the *index of refraction* of each of the two substances and the wavelength of the light. When the light is sent at an angle greater than a certain threshold, called *critical angle*, the light is completely reflected (none passes through). Now let us have a strand of glass, called *core*, wrapped by another layer of slightly different glass, called the *cladding* with index of refraction of the core being higher than that of the cladding. A light sent into the core at a particular angle will stay in the core because any of the light trying to escape the core through the cladding will be reflected back into the core, as shown in Figure 2.2. By

transmitting light through the core, it is possible to transmit bits in the form of pulses of light.

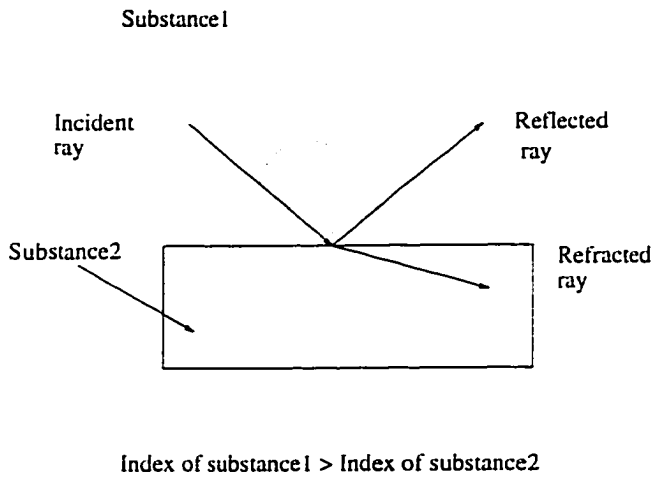


Figure 2.1: Reflection and Refraction of Light.

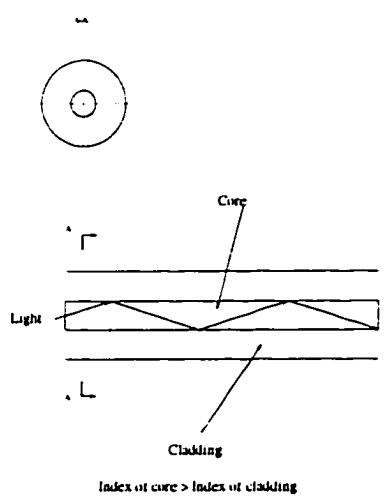


Figure 2.2: Transmission of light in a fiber.

In its simplest way, optical fibers can be used like electrical wires. One example is the Synchronous Optical Network (SONET) which is part of a large suite of telephone standards known as the *Synchronous Digital Hierarchy* (SDH), standardized by CCITT, the worldwide telephony standards body. Compared to an electrical wire, an optical fiber has much larger bandwidth, much smaller physical size, longer interconnection lengths, lower error rate, and reduced crosstalk. For example, a normal electrical cable has a chan-

nel bandwidth in the range of several Mbps to 150 Mbps depending on the transmission media that are used [77], while a single fiber can have a bandwidth as high as 50 to 75 terabits per second [58]. Optical fibers have been successfully used in telecommunication networks to replace copper wires. It is expected that fiber optics will soon be used in tightly connected systems.

Due to the huge bandwidth of fiber optics compared to electrical signaling devices, fiber sharing will be an important feature of the new generation interconnection networks which connects electronic processors with optical fibers. From past experience, high-dimensional networks show poor wire sharing [34]. It is expected that low-dimensional networks like meshes and tori are better suitable for fiber optics. Observing this new trend in interconnection networks, some researcher advocates hypernetworks [84] that have excellent wire sharing ability.

2.2 Pipelined Optical Bus

Based on two important optical transmission properties, namely unidirectional propagation and predictable propagation delay of optical signals, pipelined optical buses using time division multiplexing (TDM) multiaccess methods have been proposed [26, 27, 41, 49, 57, 62, 65]. Such an optical bus transmits packets in a pipelined fashion, achieving bandwidths higher than that of non-pipelined bus communications. Multidimensional processor arrays using pipelined optical buses as building blocks have been proposed to achieve improved system scalability [59, 65].

In the following, we briefly review the pipelined optical bus proposed in [27, 49, 62]. This is necessary because our newly proposed bus will be compared with those existing buses. Unless otherwise specified, we assume that the architectures discussed operate in the SIMD fashion.

A pipelined optical bus consists of three folded waveguides, the message waveguide, the reference waveguide and the select waveguide, connecting n processors, $P_0, P_1 \ldots, P_{n-1}$

(refer to Figure 2.3). The processor indices are linearly ordered. We call processor P_{n-1} and P_0 the *head processor* and *tail processor* of the bus, respectively. The message waveguide is used for carrying messages. The reference and the select waveguides are used together for carrying address information encoded using the coincident pulse technique [12, 41]. The bus is divided into three segments, the *transmitting segment*, which is the upper half of the bus with taps from processors, the *receiving segment*, which is the lower half of the bus with taps to processors, and the *U segment*, which is the folded part that connects the transmitting and receiving segments. Let w be the pulse duration in seconds, and v_l be the velocity of light in these waveguides. Define a *pulse time unit* (or simply, time unit) as $w \times v_l$. The spatial separation of any two adjacent taps on the waveguides is D time units. Loops are added on the receiving segments of the reference and the message waveguides. Each loop causes the light a unit time delay in a waveguide.

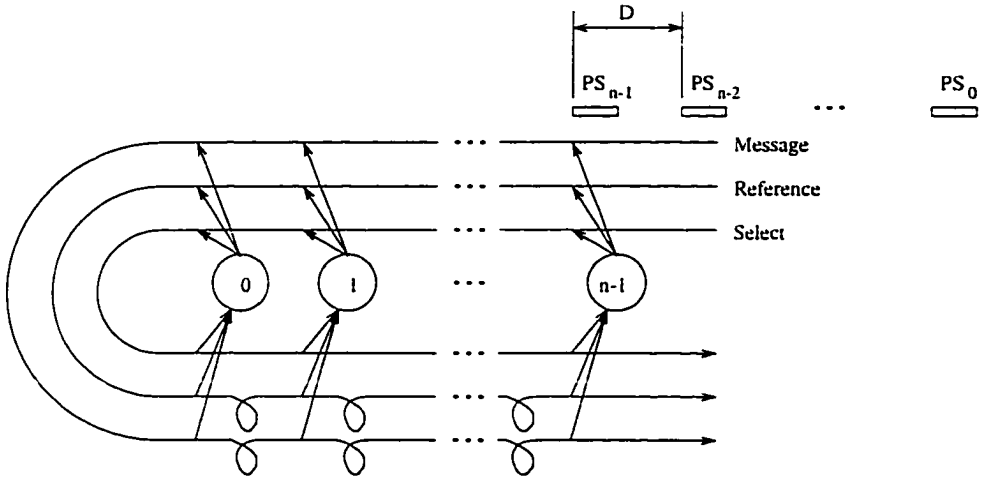


Figure 2.3: A pipelined optical bus system.

Referring to Figures 2.3 and 2.4, we explain the *coincident pulse technique*. When a source processor sends a packet (which will be defined shortly), it sends a reference pulse and a select pulse. The select pulse is transmitted later than the reference pulse with an appropriate time delay so that the two pulses will arrive at the destination processor at the same time. That is, the coincidence of the two pulses occurs at the desired destination. Whenever a processor detects a coincidence of a reference pulse and a select pulse, it starts

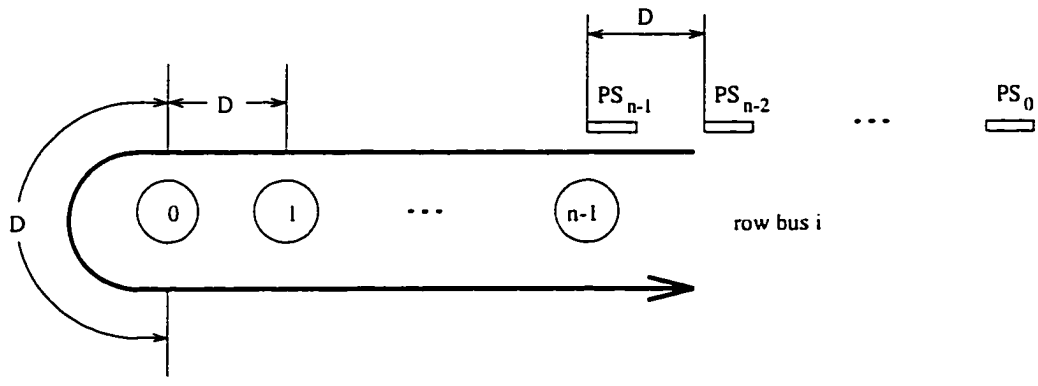


Figure 2.4: A train of packet slots.

to read from the message waveguide. More specifically, suppose that a processor is going to send a packet to processor j. We use t_r to denote the time when the processor transmits its reference pulse and t_s to denote the time when it transmits its select pulse. The two light pulses will coincide at processor j if and only if $t_s = t_r + j$.

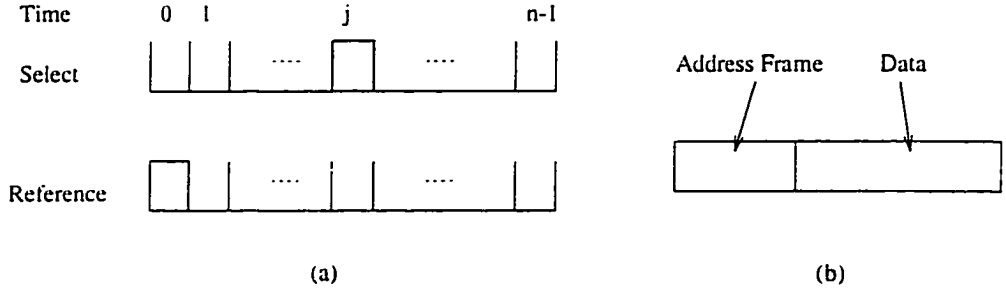


Figure 2.5: (a) An address frame. (b) A packet slot.

Let n denote the number of processors attached to the bus. We call the duration of each light pulse a *pulse slot*. Then a sequence of n pulse slots is called an *address frame*, as shown in Figure 2.5(a). Note that the existence of a select pulse at pulse slot j means that a message is to be sent to the processor j. According to the definition, the address frame for broadcasting can be easily implemented by setting a select pulse at each pulse slot of an address frame. Define a *packet* as a collection of information including an address frame and a data frame (see Figure 2.5(b)). Let L be the packet length in terms of time units. Clearly, it must be $L > n$ and $D \geq L$ in order to provide correct addressing and prevent packet overlaps.

The TDM multiaccess methods can be divided into two categories: fixed assignment and demand assignment as discussed below.

2.2.1 FA-TDM Method

The fixed-assignment method is the simplest multiaccess protocol for the bus discussed above. In the fixed-assignment TDM (FA-TDM), the n packet slots are numbered from $n-1$ down to 0, as shown in Figures 2.3 and 2.4. The packet slots are assigned to each processor in a fixed manner. The two often used techniques are *time-division source-oriented multiplexing* (TDSM) multiaccess method and *time-division destination-oriented multiplexing* (TDDM) multiaccess method. In the TDSM method, *Packet slot* (PS for short; it has L pulse slots) PS_i is fixed assigned to a source (sending) processor i. Imagine a train of n packet slots is originated on a bus. If processor i has a packet to send, it loads its packet to PS_i. In the TDDM method, a packet slot is fixed assigned to a destination (receiving) processor. If some processor i wants to send a message to a processor j, processor i must load its message to the packet slot assigned to processor j.

The major disadvantage of the fixed-assignment TDM (FA-TDM) is the requirement that the packet slots for each processor are fixed regardless whether or not it has a packet to transmit. For example, assume that packet slot i is assigned to processor i. If processor i does not have a packet to send in a particular bus cycle, packet slot i will be wasted. This situation happens to both TDSM and TDDM methods. The TDDM method has another problem. When two source processors want to send messages to the same destination processor, a packet slot collision or contention happens since the two source processors want to load their packets to the same packet slot. Therefore, a reservation scheme must be provided when the TDDM method is used.

2.2.2 DA-TDM Method

To avoid the disadvantages of FA-TDM, a demand-assignment TDM method can be used. A demand-assignment TDM method (DA-TDM) allocates packet slots to processors dy-

namically according to their demands and the traffic situation. It belongs to the class of asynchronous TDM (ATDM) multiaccess methods.

In spite of the disadvantages of FA-TDM, all previously proposed pipelined buses use it because of its simplicity. In this work, we are going to introduce the idea of flagged packet. We then use this idea to modify the known pipelined optical bus structure so that a demand-assignment TDM multiaccess method using linear priority scheme can be implemented by hardware. The detail of flagged packet and its application to optical pipelined TDM will be discussed in a separate chapter.

Chapter 3

Pipelined Optical Bus with Conditional Delays

In the previous chapter, we discuss the pipelined optical buses using the TDM access protocol. With this technique, one class of promising parallel architectures, the distributed-memory SIMD (Single Instruction Stream Multiple Data Stream) computer systems, have been proposed and studied. In such a system, the transmission latency between furthest processors is the end-to-end propagation delay of light over a waveguide. Since messages are transmitted concurrently in a pipelined fashion on a bus, this latency is hidden. Due to their remarkable advantages, such optical bus systems have received much of attention (e.g. [26, 27, 41, 49, 55, 56, 57, 65]). If enhanced with reconfigurability, the computation power of such systems can be further increased as shown in [59, 65].

In this work, we propose a modified pipelined TDM optical bus for implementing a linear array parallel computer architecture. In this bus system, switches are introduced on the receiving segment of the bus to control the signal delays on the waveguide. The states of switches are dynamically programmable under the control of processors according to computation needs. In conjunction with the coincident pulse processor addressing technique, the reconfigurability of signal delays becomes an integral part of parallel computation. We show that using this linear array architecture several fundamental parallel communication and computation operations, which include broadcasting, multicasting, binary prefix sums, processor reordering, compaction, partition and concurrent subarray

computation, can be carried out efficiently. We present parallel algorithms for the selection problem and the sorting problem to demonstrate that these operations constitute a set of powerful tools for designing parallel algorithms.

3.1 A New Optical Bus Architecture

Our new bus architecture is shown in Figure 3.1. On the receiving segment of the reference and message waveguide, we introduce p 2×2 optical switches, S_i, $1 \leq i \leq p$ (For simplicity, the switches on the message waveguide are not shown). Switch S_i can be set to one of two states, *straight* and *cross*, by processor i. When a switch is set cross, ω time delay occurs when a signal passes the switch, in comparison to the case when the switch is set straight, because of the extra traversed distance. The Ti:LiNbO3 switches of [7] can be used for this purpose. We use the new offset message transmission scheme proposed in the previous section to avoid incomplete message read. The setting of the address frame for any processor to send a packet to processor j is shown in Figure 3.2, assuming that all switches are in the cross state. Note that the address frame for this bus system requires one more pulse slot with respect to the bus system discussed in the previous section.

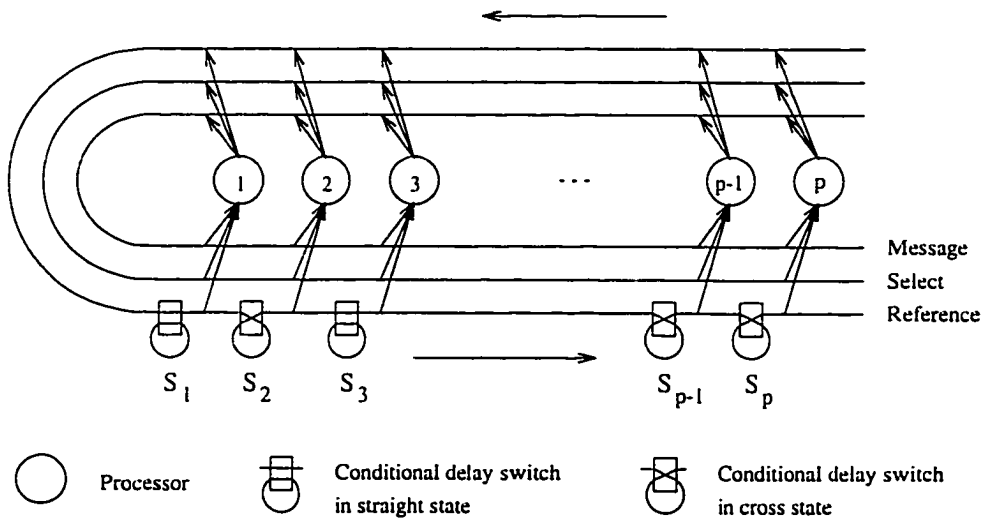


Figure 3.1: An optical bus with conditional delays.

The adjustable delays on the receiving segments of the reference and message waveguides make our bus system more powerful. Previously, two pipelined optical buses with

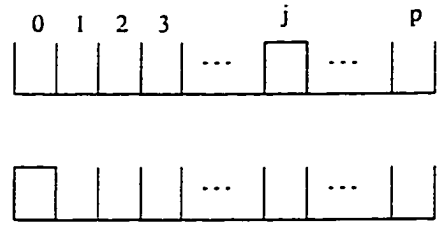


Figure 3.2: An address frame of the bus with conditional delays, assuming that all switches are in the cross state.

conditional delays were proposed in [26] and [57], both buses require more hardware. Our bus requires p switches[1]. Compared with the bus of [26], our bus, which has half the number of switches, is more powerful. The bus proposed in [57] has $2(p-1)$ 2×2 switches, $3(p-1)$ 1×2 switches, and $3(p-1)$ 2×1 switches. It is believed that the additional reconfigurability of this bus, which has about eight times the number of switches, does not exhibit additional power compared with our bus design. In what follows, we show that our bus architecture supports several useful fundamental operations. In the next section, we show how to use these operations to construct efficient parallel algorithms. Since the length of a bus grows linearly with respect to the number of processors, the time taken by a bus cycle in turn grows linearly with respect to the system size. In the complexity analysis of operations and algorithms that involve bus communications, we separate communication time from computation time. The communication performance is measured in terms of the number of bus cycles required.

Broadcast and Multicast In a multicast operation, each processor may send a packet to a group of processors, and each processor receives at most one packet. In a broadcast operation, which is a special case of multicast, a packet is sent from one processor to all other processors. There are different ways to carry out a broadcast operation. One way is to set all switches to the cross state and let the source processor use the address frame shown in Figure 3.3(a). Another way is to set all switches to the straight state and

[1] Actually, $p-1$ switches are sufficient if we slightly modify all operations and algorithms discussed in this and subsequent sections. We choose to use p switches for the simplicity of our discussions.

let the source processor use the address frame shown in Figure 3.3(b). Similarly, there is more than one way to perform a multicast operation. Assuming that all switches are in the cross state, Figure 3.4 shows the address frames for a multicast operation with three source processors, each sending a packet to a group of three destination processors. Clearly, one bus cycle is sufficient for either a multicast or a broadcast operation.

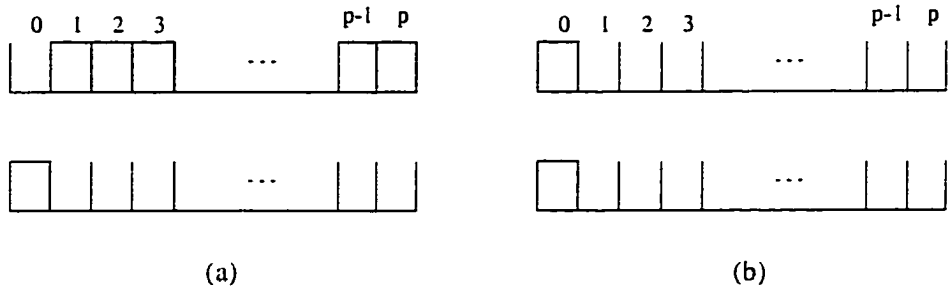


Figure 3.3: Address frames for broadcasting. (a) All switches are in the cross state. (b) All switches are in the straight state.

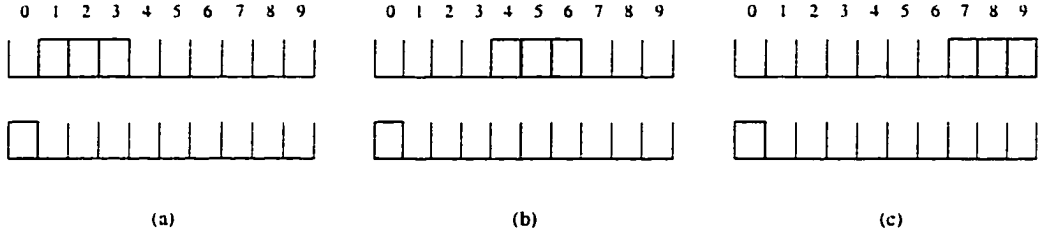


Figure 3.4: Address frames for multicasting, assuming that all switches are in the cross state. Three source processors send messages to three subsets (a), (b) and (c), of destination processors.

Binary Prefix Sums and Processor Reordering Given p one-bit values, b_j, $1 \leq j \leq p$, the binary prefix sums (BPS) problem requires the computation of $s_j = b_1 + b_2 + \cdots + b_j$ for all $1 \leq j \leq p$, where "+" is the addition operation. This operation can be done as follows. Processor j sets its switch S_j to the cross state if $b_j = 1$; otherwise S_j is set to straight state by processor j. An empty message is broadcast from processor p. Assume that the bus cycle in which the broadcasting occurs starts at t_0. It takes $(p + j - 1)\tau$ time for the reference light pulse to arrive at processor j if no delays caused by switches are involved, where τ is the time delay corresponding to distance D. If some switch delays

occur, the reference light pulse will arrive at a later time, say t_j, at processor j to cause coincidence with a broadcast select pulse. The number of unit delays can be computed by

$$i = \frac{t_j - t_0 - (p + j - 1)\tau}{\omega}. \tag{3.1}$$

We call the value computed by Equation (3.1) the i-value of processor j. Clearly, the i-value of processor j is s_j.

We say that a processor is *active* if it will participate in the next computation step; otherwise, we say that it is *inactive*. Processor reordering is to assign active processors indices in such a way that the i-th active processor is assigned a new index i. The operation for processor reordering is almost the same as that for BPS. If a processor is active, it sets its switch to the cross state; otherwise, it sets its switch to the straight state. After a broadcast operation, the i-value obtained for an active processor j is its new index. Thus, both the BPS problem and the processor reordering problem can be solved using one bus cycle and $O(1)$ computation time.

Compaction and Partition Suppose that each processor has one data item. Using processor reordering method, in one bus cycle the active processors can be assigned new ordered indices starting from 1, and the total number of active processors can be determined by the i-value of processor p, regardless whether or not it is active. Suppose that the number of active processors is s. Then, each active processor j with its new index i sends its data item to processor i. In one additional bus cycle, the s data items of active processors can be packed in the first s processors, each having one item; furthermore, the compacted data items preserve their original order. Such an operation is called an ordered compaction operation.

A partition operation is to partition p data items, one per processor, into two subsets, one contains s items, and the other contains $p-s$ items. The first subset of items are moved to the first s processors, whereas the second subset of items are moved to the remaining

processors. After the partition operation, each processor contains one original item. This operation can be carried out in two phases. The first phase employs a processor reordering operation followed by a compaction operation to move s items in the first subset to the first s processors. Then, in the second phase, active processors and inactive processors switch their roles, and another pair of processor reordering and compaction operations are performed on the $p - s$ items in the second subset. Therefore, a partition operation can be performed in four bus cycles and $O(1)$ computation time.

Parallel Operations on Subarrays For parallel computation, it is often convenient to consider a linear array connected by a pipelined bus as several subarrays, each consisting of a sequence of consecutive processors. If the processors are controlled properly, the operations on subarrays can be performed in parallel. Parallel computation on subarrays is an important feature for supporting parallel algorithms based on the divide-and-conquer design paradigm. In a divide-and-conquer algorithm, the initial problem is partitioned into several subproblems, and each subproblem, which can be further recursively partitioned, is solved independently. The combination of subproblem solutions yields a solution to the initial problem. An example divide-and-conquer algorithm is given in the next section.

Let us refer to the processors with the smallest and the largest indices in a subarray as the *head* and *tail processor*, respectively, of the subarray. In a divide-and-conquer algorithm, processors in a subarray are notified the identities of head and tail processors of the subarray in the problem partition step. Knowing the head and tail processors, a processor in a subarray can broadcast a packet to all processors in the subarray in one bus cycle using a sequence of select pulses targeted at processors in the subarray (see Figure 3.4). Obviously, concurrent broadcasting on all subarrays is a special case of multicast.

Other operations can also be performed in parallel on subarrays. Consider the BPS problem on subarrays. We set switch S_j to the cross state if $b_j = 1$, and set it to the straight state if $b_j = 0$. After one broadcast operation, each processor computes its i-value.

For a head processor of a subarray, we compute its i' value as follows: $i' = i - 1$ if this processor is active; otherwise, $i' = i$. Then, we use a multicast operation to send the i'-value of the head processor of a subarray to all remaining processors in the subarray, for all subarrays. The binary prefix sum of a processor in a subarray is obtained by subtracting i' from the i-value of the processor. Therefore, The BPS problem on subarrays can be carried out using two bus cycles and $O(1)$ computation time. Consequently, the processor reordering problem for all subarrays can be done in same amount of time.

It is not difficult for a reader to be convinced that, if the head and tail processors are known to all processors in a subarray, for all subarrays, then concurrent compaction and partition operations on all subarrays can be carried out using $O(1)$ bus cycles and $O(1)$ computation time. For brevity, we omit detailed discussions.

3.2 Algorithm Design Examples

In this section, we present examples to show how to use the basic operations described in the previous section as tools to design more complex parallel algorithms. We consider the selection problem and the sorting problem. Both of our selection and sorting algorithms have improved performance compared with previous algorithms for the buses of [26] and [57].

The selection problem is defined as finding the k-th smallest (or largest) in a given set of p elements. This problem has many applications such as merging and sorting, convex hull computation, image analysis, statistical analysis, etc. Our parallel selection algorithm *SELECT* is an adaptation of the sequential algorithm of [9]. Assume that each of the p processors contains one element. We want to find the k-th smallest of these elements. The algorithm is recursive. Initially, $SELECT(k, p)$ is called, and all p processors are active.

procedure $SELECT(k, p')$

1. **Basic case.** If $p' \leq 5$, then send all p' elements to processor 1, and find the k-th smallest one.

2. **Find median of medians.** The p' active processors are partitioned into $\lceil \frac{p'}{5} \rceil$ groups with each group having exactly five active processors except for the last group which may have fewer than five active processors. Let the first processor, which has the smallest index, in each group be the *group head* processor. Within each group, every non-head processor sends its element to its group head processor; that is, processor j sends its element to the $(\lfloor \frac{i-1}{5} \rfloor * 5 + 1)$-th active processor. After receiving all the elements from each member in its group, the head processor in each group finds the median of the five elements in constant time. By a compaction operation, all group medians are moved to the first $\lceil \frac{p'}{5} \rceil$ processors. We set these processors temporally active and all other processors temporally inactive, and call $SELECT(\lceil \frac{\lfloor \frac{p'}{5} \rfloor}{2} \rceil, \lceil \frac{p'}{5} \rceil)$ on the first $\lceil \frac{p'}{5} \rceil$ processors. Let m be the element that we have obtained. The median-of-medians m, which is also referred to as the *pivot element*, is put in processor 1.

3. **Partition.** Processor 1 broadcasts the pivot element m to all the p' active processors. Each of these processors compares its element with m. If the element in processor j is less than or equal to m, the processor sets its local Boolean variable b_j to 1; otherwise, it sets b_j to 0. By a BPS operation, the total number s of elements that are less than or equal to m can be computed (note: s is the i-value of processor p' computed by the BPS operation). If $k \leq s$, then by a partition operation, all elements that are less than or equal to m are moved to the first s processors and the remaining elements are moved to the rest of p' processors, one element per processor. If $k > s$, then by a partition operation, all elements that are greater than m are moved to the first $p' - s$ processors and the remaining elements are moved to the rest of the p' processors, one element per processor.

4. **Recursion.** If $k = s$, processor s returns the pivot element. If $k < s$, set the first s processors active and the remaining processors inactive, then call $SELECT(k, s)$;

else let $k := k - s$, set the first $p' - s$ processors active and the remaining processors inactive, then call $SELECT(k, p' - s)$.

end of *SELECT*

Now let us analyze the time complexity of this algorithm. Let $B(p)$ and $T(p)$ denote the total number of bus cycles and the computation time for a problem size p. The worst-case complexity $B(p)$ of our algorithm is derived as follows. Each of Step 1 and Step 3 takes $O(1)$ bus cycles and $O(1)$ computation time. Step 2 takes $B(\frac{p}{5})$ bus cycles. By adapting worst-case time complexity analysis given in [29] for the sequential algorithm of [9], Step 4 takes no more than $B(\frac{3p}{4})$ bus cycles for $p \geq 24$. Then, we have the following recurrence relation

$$B(p) \leq \begin{cases} \Theta(1) & \text{if } p < 24 \\ B(\frac{p}{5}) + B(\frac{3p}{4}) + O(1) & \text{if } p \geq 24 \end{cases}$$

Solving this recurrence relation yields $B(p) = O(\log p)$. Similarly, we obtain $T(p) = O(\log p)$. Hence, the k-th smallest elements of p elements can be computed using a pipelined optical bus of size p in $O(\log p)$ bus cycles and $O(\log p)$ computation time. In [56], a parallel selection algorithm based on an optical bus of [26] was presented. This algorithm requires $O(\log p)$ bus cycles and $O(\log p)$ computation time on average. But, in the worst case, $O(p)$ bus cycles and $O(p)$ computation time may be required. Hence, our algorithm has an improved performance.

The second problem we consider is sorting p elements using p processors, one element per processor. If the elements are integers, then the BPS and partition operations supported by the conditional-delay feature of our optical bus can be used to implement the radix sort algorithm. For w-bit elements, w iterations are needed, each performing a pair of BPS and partition operations on one bit position of all elements. Note that this implementation of radix sort does not involve any comparison operation between elements, and the total number of bus cycles and computation time required are independent of bus size p.

For sorting arbitrary elements, the bus of [59] was used to implement the sequential quicksort algorithm. It is shown in [59] that such an implementation has average performance of $O(\log p)$ bus cycles and $O(\log p)$ computation time. In the worst case, this algorithm requires $O(p)$ bus cycles and $O(p)$ computation time. With procedure *SELECT* at disposal, this worse case performance can be improved by the following divide-and-conquer algorithm *SORT*. Without loss of generality, assume that $p = 2^q$, and all elements are distinct.

Algorithm *SORT*
begin
  for $i = 0$ **to** $q - 1$ **do**
    for all 2^i subarrays of size 2^{q-i} **do in parallel**
      Use *SELECT* to find the 2^{q-i-1}-th element of the subarray;
      Use the partition operation to partition the elements into two subsets of equal size such that the first 2^{q-i-1} processors of the subarray contain the elements that are not greater than the median of the subarray;
    endfor
  endfor
end

The algorithm consists of $\log p$ iterations. In the first iteration, procedure *SELECT* is invoked to find the $\lfloor \frac{p}{2} \rfloor$-th element (i.e. the median). Using it as the pivot element to partition the elements, the linear array of size p is conceptually divided into two subarrays of equal size. In the k-th iteration, the medians of 2^{k-1} subarrays, each of size 2^{q-k+1}, are computed in parallel. At end of the iteration, each subarray is divided into two subarrays of equal size. This process is continued until each subarray has two processors. The complexity of each iteration is upper bounded by the operations of *SELECT*. Since each

iteration requires no more than $O(\log p)$ bus cycles and $O(\log p)$ computation time, this sorting algorithm requires $O(\log^2 p)$ bus cycles and $O(\log^2 p)$ computation time.

3.3 Summary and Discussions

Optical buses offer much larger bandwidth than electronic ones. Among available optical interconnection technologies for parallel computing, optical buses are probably the easiest to implement. Designing a bus-based parallel computer architecture requires taking both computation and communication aspects into consideration. We proposed a linear array architecture based on a pipelined TDM optical bus. We showed that using the conditional-delay and coincidence pulse techniques, several fundamental operations can be carried out very efficiently on our bus system. We also demonstrated how to design parallel algorithms on our linear architecture. Our bus structure is simpler and more powerful than the one proposed in [26], and much simpler than and as powerful as the bus of [57].

Our linear array can be used as building blocks to construct processor arrays of multiple dimensions to achieve better scalability and performance. For example, Pavel and Akl proposed to use pipelined TDM optical buses of [26] to implement a two-dimensional reconfigurable array [51]. They call this architecture an array with reconfigurable optical buses (AROB). They showed that an AROB is a powerful parallel computing architecture. In order to achieve their claimed reconfigurability, they require each processor to be equipped with several high-speed counters to determine the processor ordering after the buses are reconfigured by switches. Such a counter is used to count the number of a sequence of optical pulses. It would be impractical for an electronic processor to match the bit rate of an optical waveguide, and using optical counters would increase the system cost significantly. Our optical bus provides a simple optical solution to the reconfigurability of AROB.

We would like to note that the idea of programmable delays using electro-optically switched fiber loops has been used in the designs of TDM time slot interchanger (e.g.

[30, 37, 66, 78]). Also, two optical bus structures with conditional delays have been proposed in [26] and [57]; both are more complicated than the one presented in this work. In the bus structure of [26], switches are on the transmitting segment of the bus, and because of this parallel computation on subarrays, if not impossible, is difficult. To solve this problem, an optical bus with considerably more switches was proposed in [57]. Such a bus can be physically partitioned into several buses for parallel subarray computation. Our bus structure is more powerful than the bus of [26], and more cost-effective than the bus of [57].

The time for an optical signal to propagate distance D is τ. The communication of processors are performed in terms of bus cycles. A *bus cycle* consists of end-to-end transmission of p consecutive packet slots. Since the end-to-end latency of an optical pulse is no more than $(2p-1)\tau + (p-1)\omega$ time, a bus cycle takes no more than $3p\tau + (p-1)\omega$ time. Let τ_c denote the processor execution time of an arithmetic (say, multiplication) instruction. Then, a bus cycle takes $O(\frac{3p\tau+(p-1)\omega}{\tau_c})$ units of computation time. Pavel and Akl [59] have argued that for reasonable size buses (say up to 1000 processors), the duration of a bus cycle may be assumed constant and comparable to the time for a CPU operation. The data transmission performance can be further improved if packets of a new bus cycle are transmitted before all the packets of the previous bus cycle reach their destinations. Suppose that each processor has n packets to be transmitted continuously. Then, with overlapped bus cycles, the total time for transmitting the n^2 packets is $(2p + n^2 - 1)\tau + (p-1)\omega$, and, if n is sufficiently large, in average each packet takes only about τ time.

Despite the remarkable effect of pipelined data transmission, we still need to be careful when we assess the performance of such an optical bus. To avoid misleading time complexity claims of a linear array of processors connected by such a pipelined optical bus, it is necessary to separate the communication time from computation time. As in sequential algorithm analysis, we assume that each parallel computation step takes constant time.

The communication time is measured in terms of bus cycles. When p is small, a bus cycle can be assumed taking constant time. However, as p becomes larger, the time for a bus cycle increases proportionally. Of course, for large p, optical amplifiers are needed. The communication power of such a bus also depends on the capabilities of optical transmitters and receivers associated to processors. We conservatively assume that in each bus cycle, each processor can transmit at most one packet and receive at most one packet.

Chapter 4

Asynchronous Optical TDM Buses

As we discussed before, the TDM multiaccess methods can be divided into two categories: fixed assignment and demand assignment. The major disadvantage of the fixed-assignment TDM (FA-TDM) is the requirement that the packet slots for each processor are fixed regardless whether or not it has a packet to transmit. If a processor does not have a packet to send or receive, the packet slot assigned to this processor will be wasted. A demand-assignment TDM (DA-TDM) allocates packet slots to processors dynamically according to their demands and the traffic situation. It belongs to the class of asynchronous TDM (ATDM) multiaccess methods. If the packet generating rate of a processor is uniform, the FA-TDM is very efficient; otherwise, DA-TDM should be considered.

In this chapter, we introduce the idea of a flagged packet. We then use this idea to modify the known pipelined optical bus structure so that a demand-assignment TDM multiaccess method using a linear priority scheme can be implemented by hardware. To improve the fairness of the DA-TDM multiaccess method, we incorporate reconfigurability into our pipelined optical bus to implement the round-robin priority scheme. The scheduling of the DA-TDM multiaccesses with the round-robin priority scheme is implemented by reconfiguring the bus in hardware. We compare the performance of our pipelined DA-TDM optical bus with the pipelined FA-TDM optical bus by simulations, in terms of average message response time and fairness. Our experiments show that the performance of our pipelined DA-TDM optical buses is significantly better than that of

pipelined FA-TDM optical buses. We also discuss the possibilities of using our buses to construct multichannel switches and multidimensional processor arrays.

4.1 ATDM with Linear Priority

In this and next two sections, we introduce two possible implementations of a DA-TDM bus and evaluate their performances against that of a FA-TDM bus.

We modify the bus structure described in Chapter 2 as follows. First, we add one more loop on the receiving segments of the reference and message waveguides. These two loops are in the positions aligned with processor P_{n-1}. Then, we introduce n loops of unit delay on the transmitting segment of each of the reference, select and message waveguides. These loops are in the positions aligned with processors 0 to $n-1$. For each of the reference and message waveguides, we introduce n additional taps to processors on its transmitting segment. For the relative positions of the processors, loops and taps, refer to Figure 4.1.

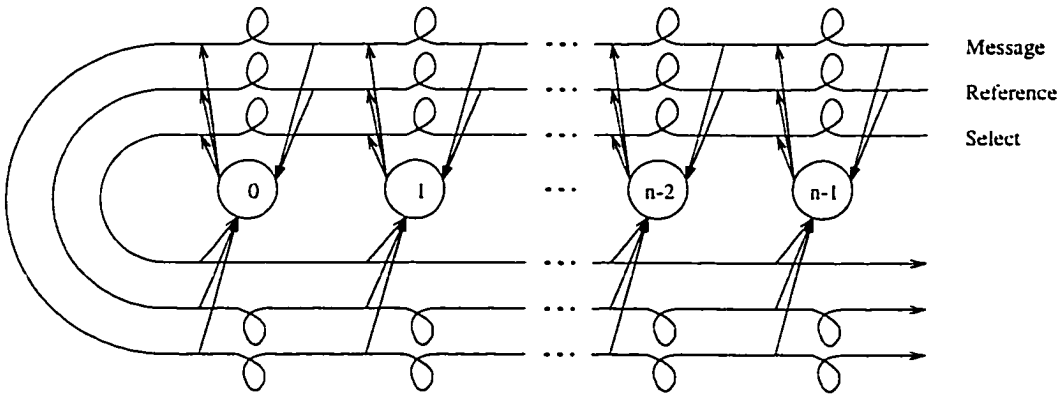


Figure 4.1: The configuration of a DA-TDM bus.

The operation of our pipelined DA-TDM optical bus is simple. If a packet slot carries a packet, we call it a *full packet slot*; otherwise, we call it an *empty packet slot*. When a processor wishes to transmit a packet, it captures the first empty packet slot that passes by and loads the packet. Once a packet slot contains a packet, it becomes a full packet slot, and it will not be loaded again.

A packet consists of two parts: header and data frame. For a packet of the bus described in Chapter 2, the header is an address frame. We add one pulse slot, called a *flag*, to the header, as shown in Figure 4.2 for the new bus. There is always a flag pulse at the flag position of a packet slot on the reference waveguide. These pulses are injected from the head processor, processor P_{n-1}. The state of a flag for an empty packet slot is represented by a flag pulse on the reference waveguide only. The state of a flag for a full packet slot is represented by a coincidence of the light pulses on the reference waveguide and the message waveguide. If the rightmost processor wishes to send a packet, it sends the flag pulse on each of the reference and message waveguides in the flag pulse slot, and then load the packet slot with a packet. For all other processors, their operations are slightly different. If a processor wishes to send a packet but detects a coincidence of the light pulses on the reference and the message waveguides in a flag position, it has to wait because a full packet slot is passing by. Otherwise, it sends a light pulse on the message waveguide at the flag position, sets the unary address in the address frame and loads the data frame. The coincidence of flag pulses on the reference and message waveguides ensures that the processors down the stream will not load this packet slot. The transition from one flag state to the other is shown in Figure 4.3, in which the transmissions of the address frame and data of a packet is omitted.

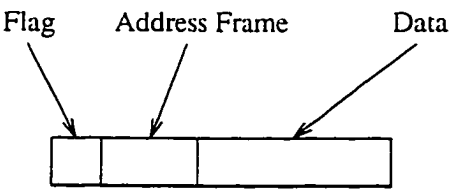


Figure 4.2: A packet with a flag.

Assume that reading a reference flag pulse and making a decision take unit time. Without delaying the flag pulse on the reference waveguide, it would not be possible to send a pulse at the same position on the message waveguide. The purpose of introducing loops to transmitting segments of all waveguides is to allow sufficient time for sending a flag

pulse on the message waveguide after detecting an empty packet slot, while maintaining the correctness of the unary addressing scheme described in the previous section.

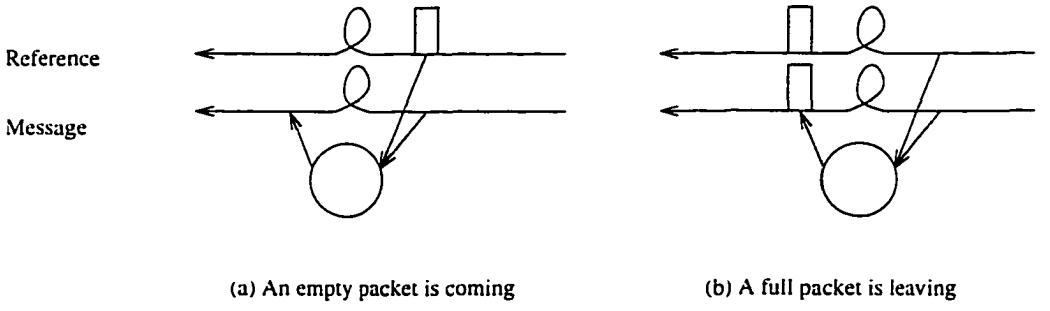


Figure 4.3: The implementation of the flag.

We have extended the coincidence pulse technique to a solution to the problem of detecting the state of a packet slot. This demand-driven packet slot assignment method is equivalent to linear priority reservation scheme. Processor P_i has a priority higher than the priority of P_{i-1}. Thus, the head processor and the tail processor have highest and lowest priority, respectively. When competing for packet slots, the processor that has the highest priority among all competing processors will be the one to succeed. As long as there are processors, regardless of where they are, demanding data transmissions, the packet slots are fully utilized. However, the processors with lower priorities may suffer from starvation, a situation in which they are not able to access the bus because of the existence of demands from processors of higher priorities.

We would like to note that the existence of loops in the positions aligned with the rightmost processor, P_{n-1}, on the transmitting segments of the three waveguides, and the receiving segments of the reference and message waveguides has no effect on the operations described above. They can be deleted. They are included for the purpose of simplifying our presentation of a reconfigurable bus in the next section.

4.2 ATDM with Round-Robin Priority

For improved performance, a more sophisticated priority scheme should be used. In this section, we propose an asynchronous TDM optical bus that uses the round-robin priority

scheme. In such a scheme, processors are arranged as a circular queue. The priorities of processors are reassigned in a rotating manner. More specifically, the current head processor, which has the highest priority, will become the tail processor with the lowest priority after a specified period of time, and at the same time all remaining processors' priorities are incremented. This priority reassignment will be done periodically.

Consider the structure obtained from the bus structure discussed in the previous section as follows: for each waveguide, remove its U-segment and close the two open ends of the transmitting segment and receiving segment to form two rings. Then, for each waveguide, we have two rings as shown in Figure 4.4. The inner ring corresponds to the receiving segment, and the outer one corresponds to the transmitting segment. The loop delays on the transmitting waveguides in the position aligned with processor P_{n-1} that appeared to be redundant in the bus structure described in the previous section now play the role of making these ring structures symmetric.

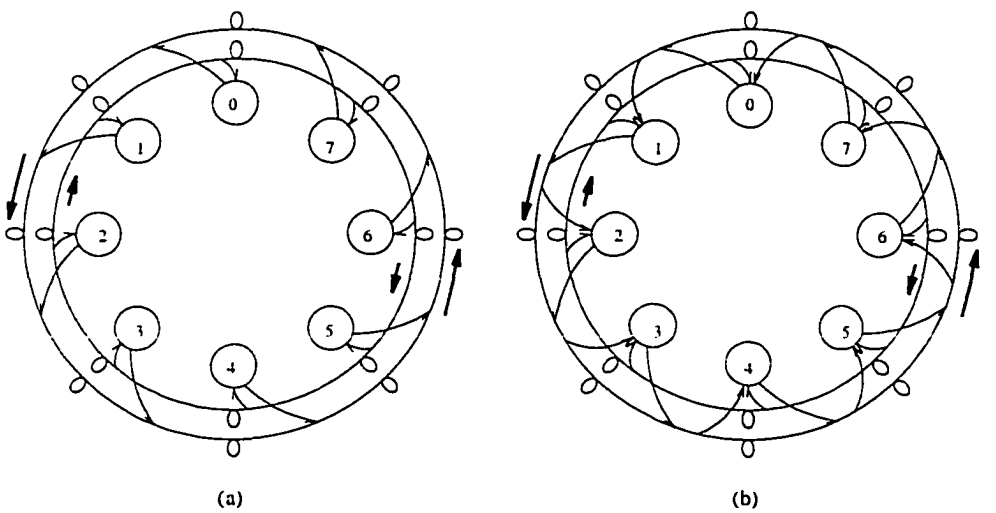


Figure 4.4: The ring structures constructed from a DA-TDM bus. (a) The ring corresponding to the select waveguide. (b) The ring corresponding to the reference and message waveguides.

In order for this ring structure to operate as a bus, we incorporate reconfigurabilities into the ring. The new structure is abstracted in Figure 4.5, where a box includes a switch (represented by a dashed smaller box), taps, and loops. The switch has two states. In one

state, the *straight state*, the switch makes the two waveguide segments to pass through it. In the other state, which we call the *U state*, the switch cuts the segments and connects one pair of open ends, as shown in Figure 4.6. Such a switch can be implemented by two 2×2 switches that have the straight and cross states. The Ti:LiNbO3 switches of [7] can be used for this purpose. Note that a waveguide of length D that forms a U turn in the U state is included between the two 2×2 switches (see Figure 4.6).

If one switch is set to the U state but all remaining switches are set to the straight state, the ring is broken and a folded bus is formed. We use B_k to denote the bus configuration in which P_k is the head processor, $P_{k'}$, where $k' = (k+n-1) \bmod n$, is the tail processor, and all processors are assigned new linearly ordered indices from 0 to $n-1$ such that the new indices of P_k and $P_{k'}$ are 0 and $n-1$, respectively. Setting any switch to the U state corresponds to a unique linear priority scheme. If the switches are selected to be set to the U states in a circular way, the round-robin priority scheme can be implemented. This is exactly what we are going to do. We assume that the initial bus configuration is B_0. Suppose that the current configuration is B_j, then the next configuration is $B_{(j+1) \bmod n}$. For $n = 8$, the system will be reconfigured as $B_0, B_1, B_2, \ldots, B_7, B_0, B_1, B_2, \ldots$ in sequence. From B_j to $B_{(j+1) \bmod n}$, all we need to do is to set the switch associated with P_j to the straight state and the switch associated with $P_{(j+1) \bmod n}$ to the U state simultaneously.

The operations of this bus are divided into overlapped phases, counting from the 0-th phase with bus configuration B_0. Assume that a switch takes δ time to switch from one state to another, and let $D \geq L + \delta + 1$. Each phase, which consists of m, $m \geq n$, consecutive packet slots, is divided into two subphases. The first subphase consists of the first $m-n$ packet slots, and the second subphase consists of the remaining n packet slots. The first slot and the last slot in the second subphase are not used. These two packet slots are called *unused slots*. Let t_i be the time when the i-th phase starts. Assume that $t_0 = 0$, and define $t_i = t_{i-1} + mD$, $i > 0$. For simplicity, we assume $D = 1$ and then we have $t_i = t_{i-1} + m$ At time $t_0 = 0$ the system is in configuration B_0. The system changes

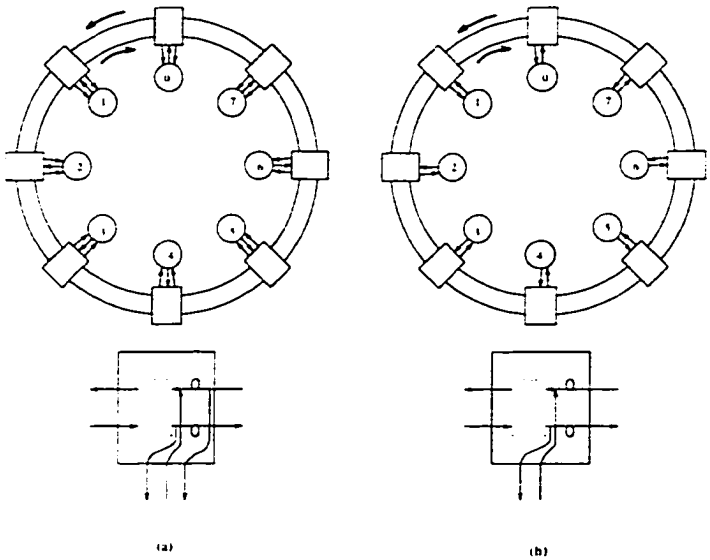


Figure 4.5: DA-TDM bus with hardware round-robin priority scheme. (a) The reference and message waveguides. (b) The select waveguide.

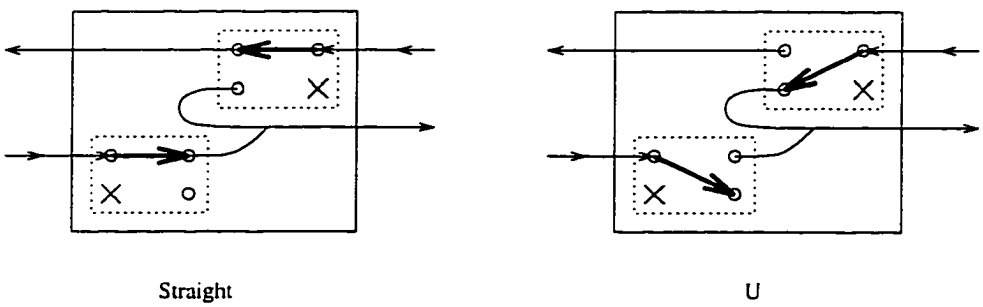


Figure 4.6: Switch implementation.

its configuration at time $t_i - 1$'s, for $i > 0$; or the system starts a new configuration at time t_i's. We now explain how the system works.

Consider the 0-th phase. In its first subphase, the address frames of packet slots are set assuming configuration B_0 (refer to Figure 4.7(a)). In the second subphase of the 0-th phase, although the system remains in configuration B_0, packet slots are loaded with address frames set assuming configuration B_{n-1}. At time $t_1 - 1 = m - 1$, all packets of the first subphase of phase 0 have passed processor P_7 on the receiving segment of B_0, the first unused slot of the second subphase is on the U segment of B_0, and all loaded packets of the second subphase are still on the transmitting segment of B_0. At this time, the switches associated with P_0 and P_1 change their states simultaneously, and consequently, the system becomes B_1. Since the first and the last slot of the second subphase of phase 0 are unused, this state change will not cause any packet loss. Since the address frames of packets of the second subphase of the 0-th phase are set according to B_1, these packets will use B_1 to reach their destinations. There is a problem with this bus reconfiguration. Right before the reconfiguration at time t_1, the last $n - 1$ packets of the first subphase of phase 0 are on the receiving segment of B_0. At time $t_1 - 1 + \delta$, the bus is reconfigured, and there is no way for these packets to reach P_0. Thus, we insist that the last $n - 1$ packet slots of the first subphase of phase 0 will not carry any packet with P_0 as destination. This can be done by programming processors to send packets to any, but P_0, processors using the last $n - 1$ packet slots of the first subphase.

At time t_1, the bus system starts to act as B_1, and phase 1 is initiated (refer to Figure 4.7(b)). Now, the last packet slot of phase 0 becomes the first slot of phase 1. In the first subphase of phase 1, the address frames of packet slots are set according to configuration B_1. In the second subphase of phase 1, although the system remains to be in configuration B_1, packet slots are loaded with address frames set assuming configuration B_2. At time $t_2 - 1$, all packets of the first subphase of phase 1 have passed processor P_0 on the receiving segment of B_1, the first unused slot of the second subphase is on the U segment of B_1,

and all loaded packets of the second subphase are still on the transmitting segment of B_1. At this time, the switches associated with P_1 and P_2 change their states simultaneously, and consequently, the system becomes B_2. Since the first and the last slot of the second subphase of phase 1 are unused, this state change will not cause any packet loss. The last $n-1$ packet slots of the first subphase of phase 1 will not carry any packet with P_1 as destination. In the next phase, the configuration B_2 is used, as shown in Figure 4.7(c).

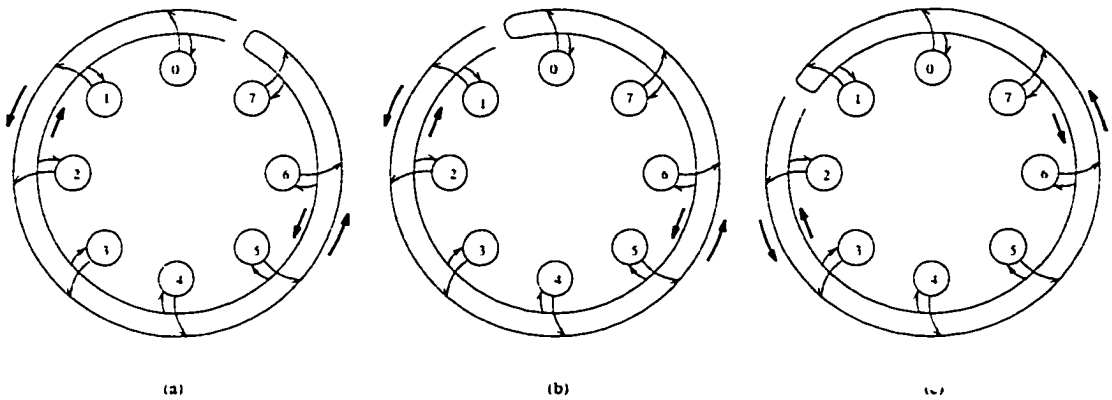


Figure 4.7: DA-TDM bus configurations: (a) B_0 at time t_0, (b) B_1 at t_1 and (c) B_2 at t_2.

In general, this system operates as follows. If the system at time t_j has a configuration B_j, then the system is reconfigured at time $t_{j+1}-1$, and at time t_{j+1}, the system's configuration becomes $B_{(j+1) \bmod n}$. The last packet slot, which is unused, of phase j becomes the first slot of phase $j+1$. In the first subphase of phase $j+1$, the address frames of packet slots are set assuming configuration $B_{(j+1) \bmod n}$. In the second subphase of phase $j+1$, packet slots are loaded with address frames set according to configuration $B_{(j+2) \bmod n}$. The last $n-1$ packet slots of the first subphase of phase $j+1$ will not carry any packet with $P_{(j+1) \bmod n}$ as destination. By a simple induction, it is easy to verify that, operating in this way, all packets will reach their destination processors. Consider the case that $n=8$. The bus configurations at time t_0, t_1 and t_2 are shown in Figure 4.7.

Using this DA-TDM with round-robin priority, $m-1$ out of every m packet slots can carry packets. If m is too large, then the system may behave like one with linear priority scheme. But if m is too small, the packet slots may not be effectively utilized. We can

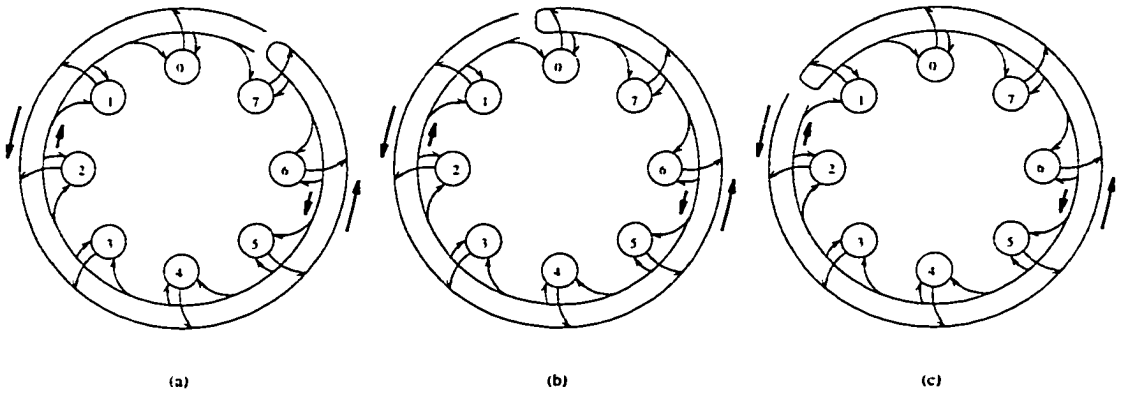


Figure 4.8: DA-TDM bus with double-input processors. Configurations: (a) B_0, (b) B_1 and (c) B_2.

select the m value that achieves the best performance. To further improve the performance of this bus architecture, an additional input from each waveguide can be introduced for each processor as show in Figure 4.8. With double-input processors, the last $n-1$ packet slots of the first subphase of each phase can carry packets with arbitrary processors as their destinations.

4.3 Simulation

In this section, we evaluate the performance improvement obtained by our DA-TDM optical bus systems over the FA-TDM optical bus systems. The performance is measured in terms of average message response time. Let τ be the time between the starting times of two consecutive packet slots. *Message response time* is the elapsed time from the time a message is generated for transmission to the time that the message transmission is completed. We use $n\tau$ as the unit to measure the response time, where n is the number of processors connected by the bus. The *average message response time* (AMRT) is found by dividing the total response time of all the messages by the number of messages. Of course, the smaller the AMRT, the better the performance.

In our simulations, $m = 2n$ and $n = 100$. Negative exponential probability distribution is assumed for the message transmission demands. We also assume that each processor has the same message generating rate per $n\tau$ time. Uniform probability distribution is

used for message sizes. The size of a message is the number of packets in the message. We consider three cases of randomly generated messages: the average message sizes 10, 50 and 100. Let r_i be the average packet generation rate of processor P_i, we define the *bus load* $\alpha = \frac{\sum_{i=0}^{n-1} r_i}{n}$; clearly $0 \leq \alpha \leq 1$. With a fixed bus load, the AMRT may vary when different bus systems are used.

Figure 4.10 shows the comparison of the AMRTs of the FA-TDM bus and the DA-TDM bus with linear priority. The average message size is 10. The figure shows that the AMRT of our DA-TDM bus with linear priority is insensitive to the bus load, and it is significantly smaller than the AMRT of the FA-TDM bus. For example, when bus load $\alpha = 0.5$, the AMRT of the DA-TDM bus with linear priority is $0.16n\tau$ while the AMRT of the FA-TDM bus is $15n\tau$. When bus load = 0.9, the AMRT of the DA-TDM bus with linear priority and the FA-TDM bus is $0.66n\tau$ and $56.88n\tau$, respectively. For this case, the AMRT of the DA-TDM bus is about 85 times shorter than that of the FA-TDM bus.

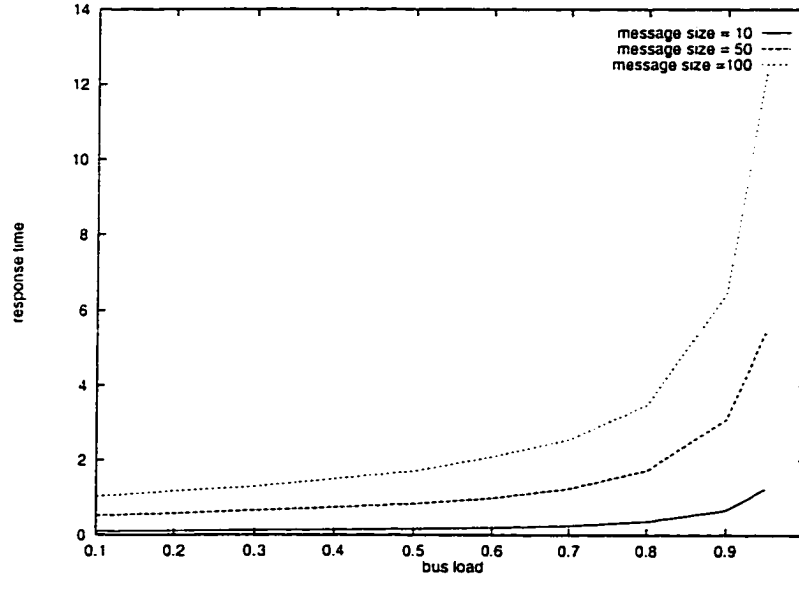


Figure 4.9: Relation between the AMRT and the message size for a DA-TDM bus with linear priority.

Figures 4.11 and 4.9 show the relation between the AMRT and the average message size for a DA-TDM bus and a FA-TDM bus, respectively. It is easy to see that for the FA-TDM bus, the AMRT is lower-bounded by the average message size. This is exactly

the problem of the FA-TDM bus. However, for the DA-TDM bus, when the bus load is low, the average response time is very small. In our experiments, the performance of the DA-TDM bus is two orders better than that of the FA-TDM bus for the assumed traffic pattern.

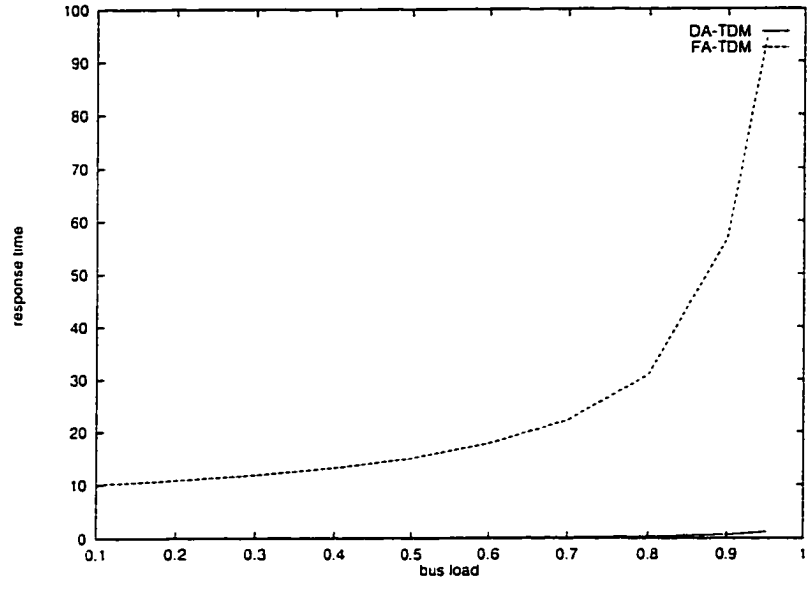


Figure 4.10: Comparison of the AMRTs of a FA-TDM bus and DA-TDM bus with linear priority.

The unfairness of the DA-TDM bus with linear priority scheme is shown in Figure 4.12. Unfairness becomes severe when the bus load is larger. However, we observed that even when the bus load reaches 0.9, the average message response time for the processor with lowest priority is still less than that of the same processor for the FA-TDM bus system.

The simulation results of our DA-TDM optical buses using the round-robin priority are consistent with our expectation. Figure 4.13 plots the average response time of each processor. The curve is the result of the simulating a bus running $10^6 n\tau$ time. If simulation time is sufficiently longer, we will expect a smooth, flat curve. Figure 4.14 shows the AMRT with respect to different bus loads, which are about the same as the AMRT of the DA-TDM bus using linear priority. In conclusion, our round-robin priority scheme does solve the unfairness problem, while maintaining the same AMRT of the DA-TDM with the linear priority scheme.

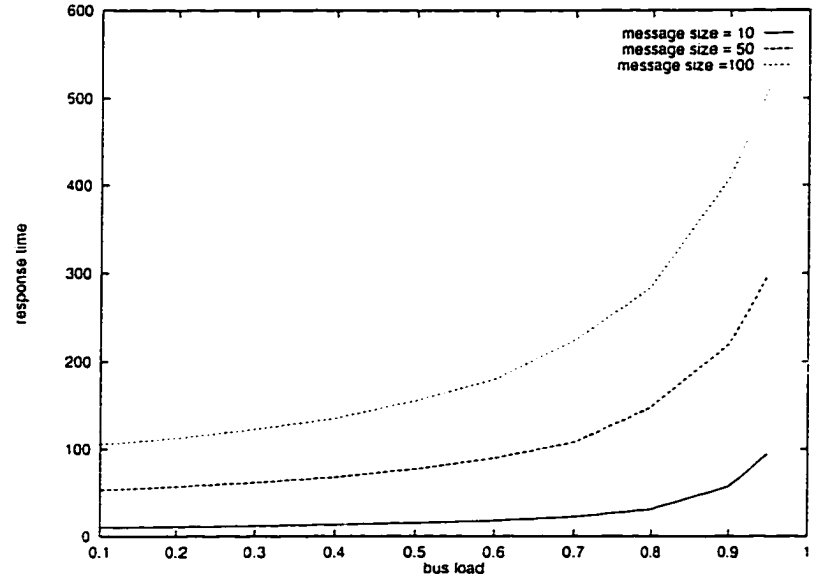


Figure 4.11: Relation between the AMRT and the message size of an FA-TDM bus.

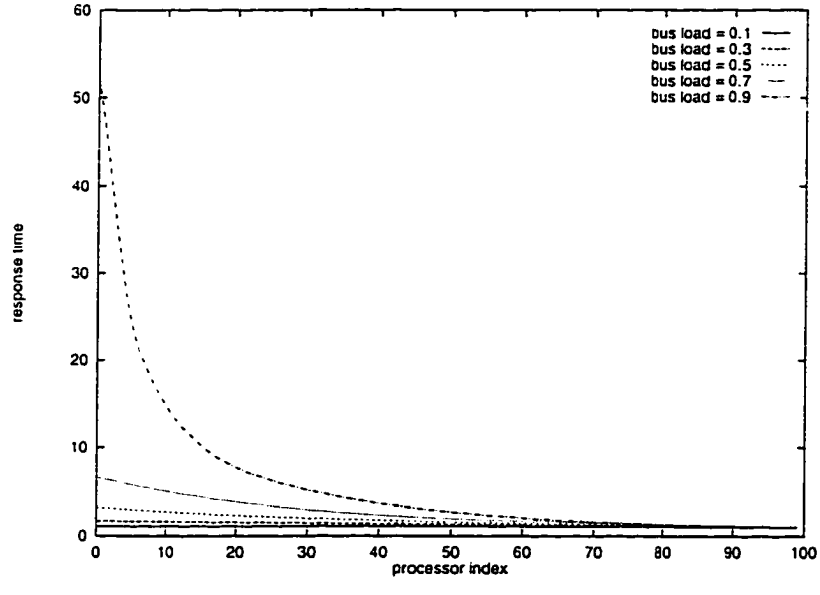


Figure 4.12: The unfairness of the DA-TDM with linear priority scheme.

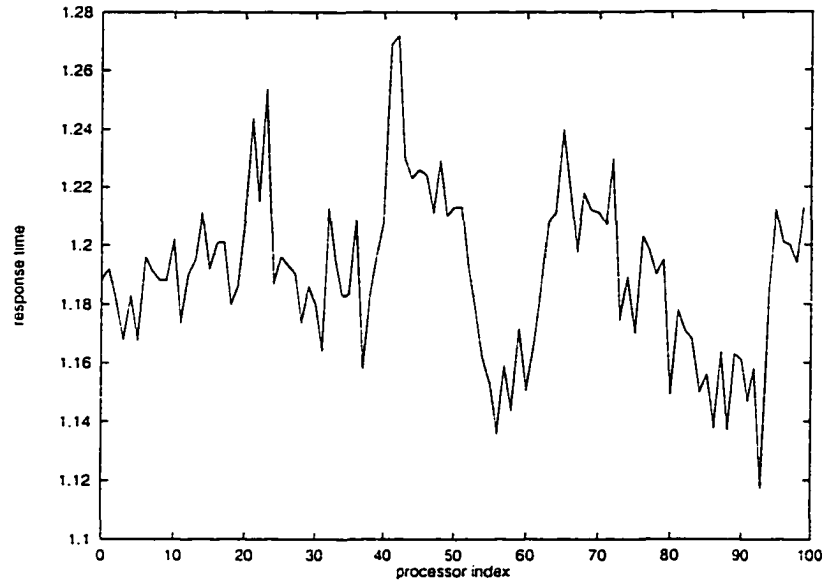


Figure 4.13: The unfairness of the DA-TDM bus with the round-robin priority scheme.

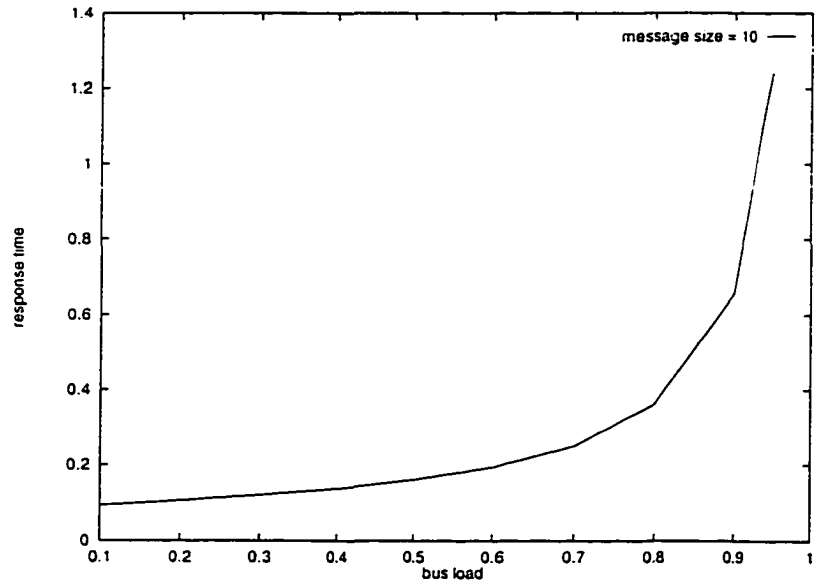


Figure 4.14: The AMRT of the DA-TDM bus with the round-robin priority scheme.

4.4 Summary and Discussions

We have introduced a pipelined asynchronous TDM optical bus based on coincidence pulse technique. We also proposed a reconfigurable version of this bus to solve the unfairness problem. Our simulation results indicate that the performance of our TDM bus is much better than the performance of its FA-TDM counterpart. In this section, we conclude this paper by mentioning two possible generalizations of our reconfigurable DA-TDM bus.

Our bus architecture can be used to implement an $n \times n$ switch shown in Figure 4.15(a). There are n input channels I_k, and n output channels O_k, $0 \leq k \leq n-1$. For simplicity, the buffering queues of these channels are omitted in the figure. We can implement such a switch by a DA-TDM bus with round robin priority scheme in the way shown in Figure 4.15(b). For each input channel, there is a device responsible for injecting packets into the waveguides, and for each output channel, there is a device responsible for detecting the coincidence pulses and picking up packets. The performance of such a switch is expected to be good, as indicated by our simulation results given in the previous section.

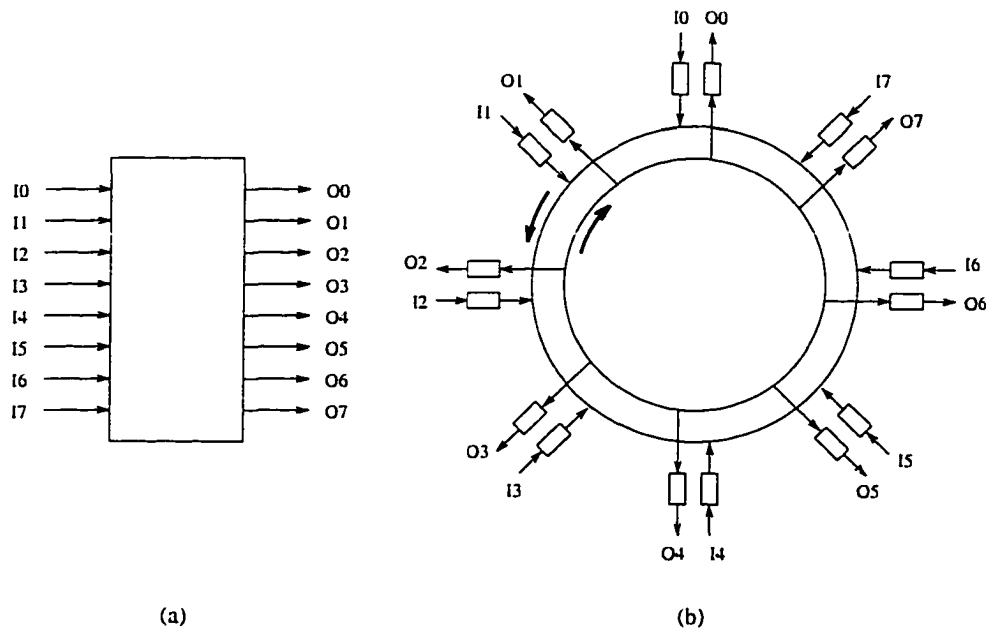


Figure 4.15: Implementing a switch using a DA-TDM bus. (a) A 8 × 8 switch. (b) A reconfigurable DA-TDM bus with 8 pairs of I/O devices.

There are several factors that constrain the size of a bus-connected system: the bus fan-out, power distribution problem, the length of unary addresses, and the increased latency when the number of processors is large. To improve the scalability, we can use our DA-TDM bus as a building block to construct processor arrays. We construct an $n \times n$ two-dimensional processor array as shown in Figure 4.16(a), where our DA-TDM buses with round-robin priority scheme are used to connect rows and columns. This structure is symmetric in the sense that the rows and columns of the array are connected by identical buses. The row communications and column communications are separated.

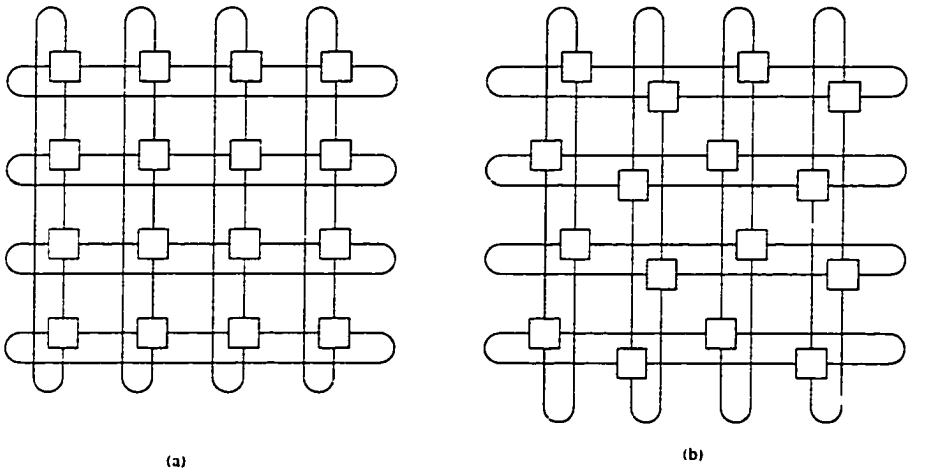


Figure 4.16: (a) A two-dimensional processor array. (b) A physical arrangement of the array.

This processor array has advantages in scalabilities over the ASOS architecture proposed in [65]. The structure of our processor array can be easily extended to higher dimensions. For example, a three-dimensional array can be constructed by introducing the same DA-TDM buses along the third dimension. It is hard, if possible, to extend the processor array ASOS with all-optical spanning buses to higher dimensions. For an $n \times n$ array using DA-TDM buses, the address frame contains n pulse slots. But for the ASOS of the same size, the address frame has $2n - 1$ pulse slots. Even for the $n \times n \times n$ 3-D array with DA-TDM buses, the size of address frame remains to be n. Since the row buses and column buses in our 2-D array are totally separated, the power distribution

for our array is simpler. Furthermore, a separate clock can be used to control each bus. Simplified clock distribution allows for a larger system to be built. We can even drop the SIMD assumption by letting each processor have its own clock. In such a situation, the flag signal on the reference waveguide has an additional role of acting as a synchronization signal, and the loops on the transmitting segment of each waveguide are lengthened to allow sufficient time for clock synchronizations. The structure of our 2-D array using reconfigurable DA-TDM buses is essentially a torus. The adjacent processors in a torus can be laid out with equal physical wiring separations as shown in Figure 4.16 (b) to avoid potential wiring problems.

The major disadvantage of our 2-D array is that when two processors on different rows and/or columns communicate, intermediate O/E and E/O conversions are required. For very sparse communications, the conversion overheads may contribute to a slowdown factor, compared with all-optical communications. We believe that the actual performance of our 2-D processor array can be much better than the ASOS architecture of [65]. There are several reasons: (1) Our buses use DA-TDM, rather than FA-TDM, so that the bandwidths of waveguides are fully utilized. (2) The pipelined row communications and column communications can be chained (for pipeline chaining, refer to [32]) so that the extra latency caused by conversions can be hidden (for latency hiding, refer to [34]). (3) Using the round-robin priority scheme, the fairness is enforced without losing the gain in average message response time over the FA-TDM. (4) The structure of ASOS is not symmetric, and it only allows X-Y dimension ordering routing paths. Due to the symmetry of row buses and column buses of our array, the routing paths can be more flexible. Both X-Y and Y-X dimension ordering routing paths can be used simultaneously for different source-destination pairs to avoid traffic congestions. Adaptive routing algorithms that route packets according to network conditions can be incorporated.

Chapter 5

Processor Arrays Connected by Segmented Buses

The realization of general-purpose, massively parallel computers hinges largely on being able to build scalable interprocessor networks. The full advantage of parallelism can be realized if processors are fully connected (as in a complete graph). In reality, a fully-connected system is either too costly or impossible to build. The communication cycle may far exceed the processor cycle, due to many hardware limitations. Interprocessor communication is the bottleneck in the overall system performance.

A Multiple-bus system as an interconnection network for shared memory multiprocessors has been extensively studied in the literature. However, due to bandwidth limitations, shared-buses are not suitable for exploiting large parallelism in applications. It has been proposed to augment low-dimensional point-to-point networks, such as meshes, by multiple buses to improve broadcasting performance [60, 74, 81], as shown in Figure 5.1. A number of multiprocessors connected by multiple reconfigurable buses have also been proposed. Examples include, among many, the reconfigurable multiple bus machine [79], the bus automaton [69], the reconfigurable mesh [51] (Figure 5.2), mesh with hyperbus [28] (Figure 5.3) and the polymorphic torus [42]. The common feature of these machine models is that the bus configurations can change under program control. Some of these models have been shown surprisingly powerful because dynamically reconfigurable communication paths are used to perform tasks that can be done only by processors in other parallel

architectures. For example, using bus control techniques to reconfigure communication paths as an integral part of computation, it is shown in [36, 46, 53, 54] that n numbers can be sorted in constant time on an $n \times n$ reconfigurable mesh.

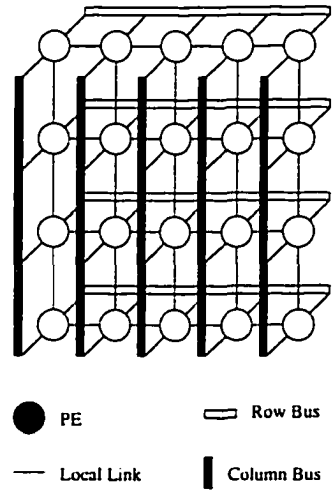


Figure 5.1: A 4x5 mesh with multiple broadcasting.

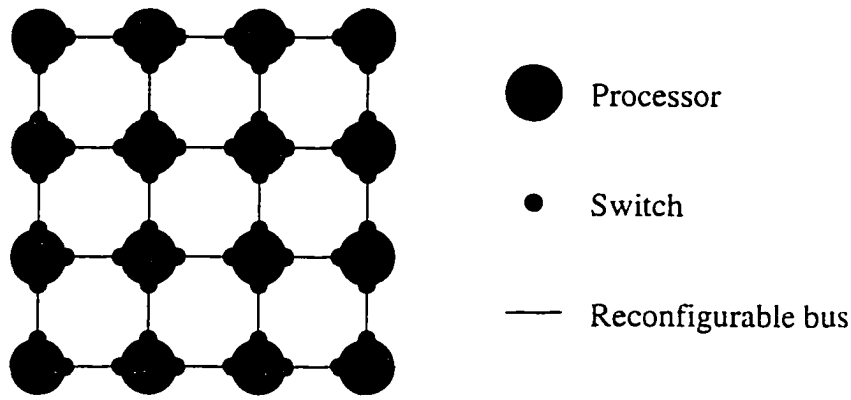


Figure 5.2: The reconfigurable mesh architecture.

Recent advances in optical interconnect technologies have drastically changed the landscape of interconnection schemes. Recently, massively parallel computing using optical interconnections has received considerable attention. Photons are non-charged particles, and do not naturally interact. Consequently, there are many desirable characteristics of optical interconnects: high speed (speed of light), increased fanout, high bandwidth, high reliability, longer interconnection lengths, low power requirements, and immunity to EMI with reduced crosstalk. The characteristics of optical interconnects have significant

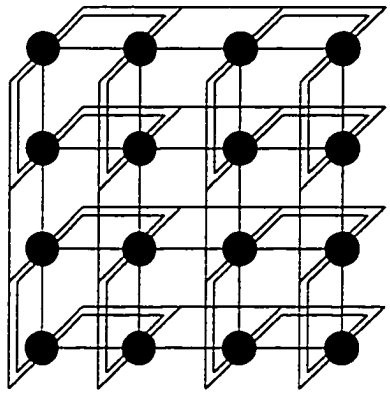


Figure 5.3: A 4x4 mesh with hyperbuses.

system configuration and complexity implications. Multiple-bus configurations with increased scalability are possible because of relaxed fanout and distance constraints. The optical fanout (which is the maximum number of processors that can be attached to an optical connecting device) is not bound by capacitance but by the power that must be delivered to each receiver to maintain a specified bit-error-rate. Processors can be arranged at increased physical distances. Several optical bus structures have been proposed to utilize the advantages of fiber optics (e.g. [27, 41, 49, 57, 59, 62, 80]).

In this work, we propose a class of reconfigurable buses, called *segmented buses*. A segmented bus connecting p processors, denoted by $B(p)$, is a bus that can be dynamically partitioned into several segments, each connecting a subset of processors, by switches. We also generalize the concept of segmented bus to obtain parallel architectures of higher dimensions, called k-dimensional mesh connected by segmented buses (k-D MCSB). We show that the segmented bus and the k-D MCSB are versatile parallel computing architectures by showing that they can simulate a wide variety of useful network structures. In particular, we show that $B(p)$ can simulate any linear array or ring of no more than p processors with a constant slowdown factor, and $B(p)$ can simulate a $(2p-1)$-processor complete binary tree, X-tree and one-dimensional multigrid with an $O(\log p)$ slowdown factor. Then, we use these results to show that a k-D MCSB can simulate a k-D mesh or torus with a constant slowdown factor, an $N \times N$ MCSB can simulate an $N \times N$ mesh-

of-trees, an $N \times N$ multigrid network and $N \times N$ pyramid network with an $O(\log N)$ slowdown factor. It would not be complete without considering the algorithmic aspect of the segmented bus based architecture. We demonstrate the advantages of parallel architectures based on segmented buses by giving a parallel algorithm for the prefix computation problem.

5.1 Segmented Buses

A *segmented bus* is a bus that can be dynamically partitioned into several segments by switches. A segmented bus $B(p)$ is obtained as follows. Given p processors P_i, $0 \leq i < p$, we connect them by a bus in the linear order of their indices. We use p to indicate the *size* of the bus. Considering a bus as a line, the p connection points on the line (bus) divide the line (bus) into $p+1$ intervals. Then, we select a subset of m intervals, and add a switch in each of these intervals. These switches divide the bus into $m+1$ segments, which are called *basic bus segments*. By dynamically setting the switches, the bus is partitioned into a set of disjoint segments, which are called *compound bus segments*. A compound bus segment consists of a series of basic bus segments. For simplicity, we also refer to a compound bus segment as a *sub-bus*. At any time instance, only the processors that are connected by a common sub-bus can communicate with each other. Parallel data communication among processors on a segmented bus is achieved by a sort of space division multiplexing.

We propose a class of segmented bus architectures called 2^i*-spacing segmented buses* that connects $p = 2^n$ processors, where $0 \leq i < n$. The 2^i-spacing segmented bus has $\frac{p}{2^i} - 1$ switches. The switches are inserted between every 2^i consecutive processors attached to the bus; that is, the 2^i-spacing has $\frac{p}{2^i} = 2^{n-i}$ basic segments of equal size. For $n = 4$, we illustrate 1-spacing, 2-spacing and 4-spacing segmented buses in Figure 5.4. For $i < j$ and same n, the 2^i-spacing segmented bus is more costly than the 2^j-spacing, but the former is more powerful than the latter. The class of 2^i-spacing segmented buses provide a wide range of cost/performance tradeoffs: linearly ordered processors may be connected

by one or more segmented buses, and/or the number of switches on a segmented bus may be selected from several alternatives.

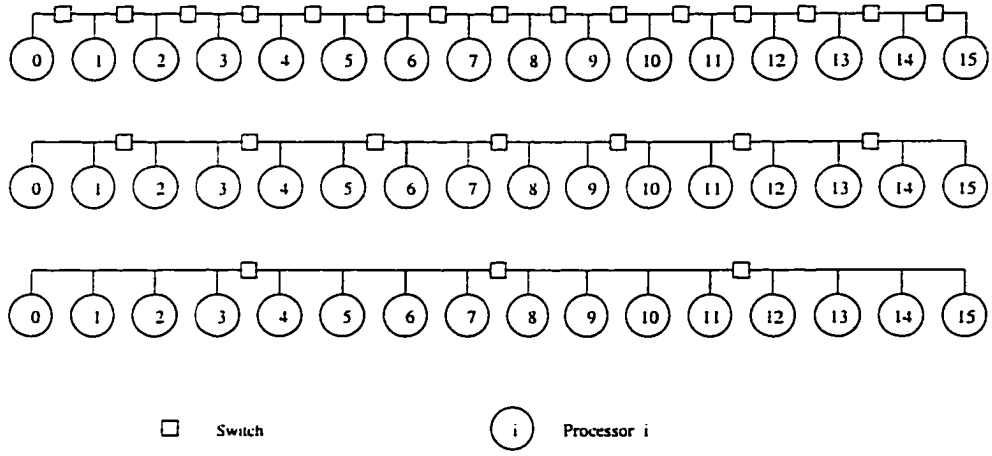


Figure 5.4: 1-spacing, 2-spacing and 4-spacing segmented buses of size 16.

The hardware implementations of a segmented bus can be in either the electronic domain or the optical domain. Segmented buses are more justifiable in the optical domain. An optical bus system is usually implemented as a folded bus. In such a configuration, each processor is attached to the bus twice, one attachment for reading (using a photo diode) and the other for writing (using a laser diode), as shown in Figure 5.5. In Figure 5.6, we show how to implement a 2-spacing segmented folded bus. Each switch is a 2×2 electronically controlled optical device [3, 6, 73], which can be in one of two states, straight and cross. Switches are grouped into pairs, both switches in a pair are set to one of the straight and cross states at the same time. These switches allow for all-optical paths without intermediate O/E and E/O conversions.

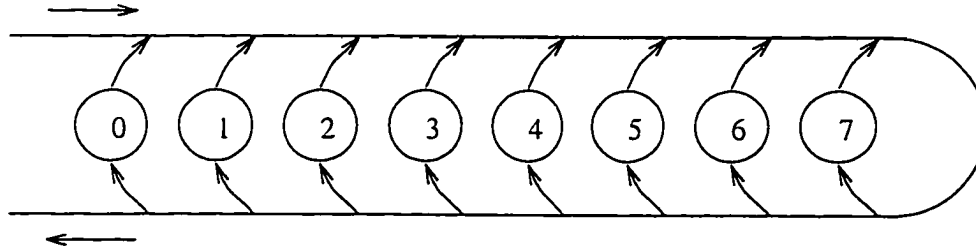


Figure 5.5: Folded bus configuration.

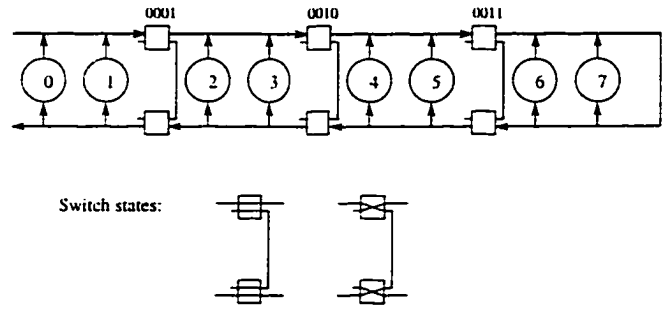


Figure 5.6: 2-spacing segmented folded bus.

Bus communications can be either synchronous and asynchronous. In asynchronous mode, arbiters are needed to allocate the bus to processors in an *on-line* fashion. Since there are $m = 2^{n-i} - 1$ switches on the bus, the total number of possible sub-bus partitions is $2^m = 2^{2^{n-i}-1}$. While dynamic bus partition, which requires one bit per switch, is easy to achieve, the complexity of bus access control poses major difficulties. One approach is to make the arbiters also partitionable. Segmented buses are more suitable for synchronous communication. We can equip each processor with an *off-line* circuitry so that both segment partitions and sub-bus allocations, although operated dynamically, are predetermined by an off-line scheduling algorithm. With off-line bus allocation assumption, the bus partitions can be "compiled" in advance. The advantage of this method is that the complexity of system design can be reduced because a handshaking mechanism, which is necessary in an on-line environment, and arbiters can be omitted.

Because of physical limitations, the number of processors and the number of switching elements attached to a segmented bus cannot be so large. To achieve higher scalability, we generalize the notion of segmented bus to obtain parallel architectures of higher dimensions. We define a 2^{nk}-processor *k-dimensional mesh connected by 2^i-spacing segmented buses* (k-D MCSB), denoted by $M_k(2^n, 2^i)$, as follows: the processors, which are denoted by $P_{i_0,i_1,\cdots,i_{k-1}}$, $0 \leq i_j < 2^n$, are connected by 2^i-spacing segmented buses of size 2^n, each connecting processors with the same $k-1$ processor indices. An $M_2(8,2)$, 8×8 mesh connected by 2-spacing segmented buses, is shown in Figure 5.7.

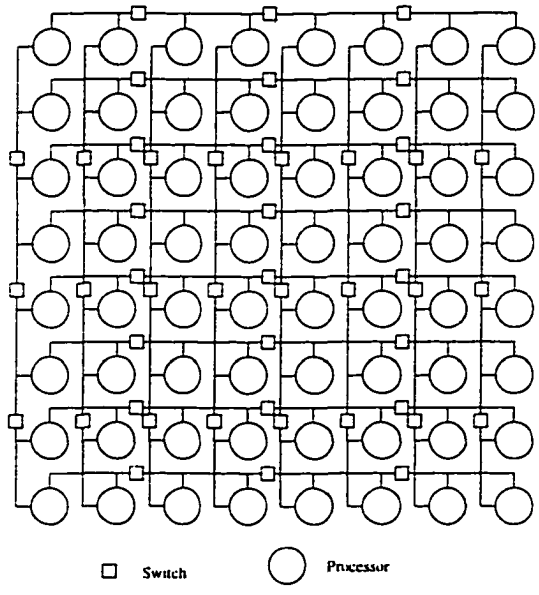


Figure 5.7: A 2-D MCSB $M_2(8,2)$.

5.2 Versatility of Parallel Architectures Based on Segmented Buses

Parallel architectures based on segmented buses, especially those of low-dimensions, are feasible for implementation. They have low wire densities, small diameters, and a processor in such a system has a small number of I/O ports. The control of switches in a such system is less complex than most other parallel models based on reconfigurable buses, such as reconfigurable meshes. To justify that this class of architectures is suitable for general-purpose parallel processing, we need to show that they perform well for a large range of applications. A parallel model is considered versatile if it can efficiently simulate many other useful parallel machine models. If machine M_1 can simulate machine M_2 efficiently, then any algorithm developed on machine M_2 is portable to machine M_1. In this section, we compare segmented-bus based architectures with several point-to-point network based architectures. We demonstrate that architectures based on segmented buses are versatile by showing that they can efficiently simulate many useful machine models.

For simplicity, we assume synchronous computation and communication modes. That is, a parallel computation process is partitioned into computation and communication

steps. In a computation step, a subset of processors execute an instruction. In a communication step, each processor communicates with a subset of processors that are directly connected to it. Let M be a parallel machine whose underlying interprocessor connection structure is a point-to-point network (a conventional graph), i.e., pairs of processors are connected by dedicated links. We define a *maximum communication step* of machine M as a communication step with all links of its interconnection network engaged in the communication. When we consider simulating a communication step of M, we always consider its maximum communication steps. This is a rather conservative approach, since not all realizations of point-to-point networks have facilities that support all-port communications. Similar techniques were used in [16].

Our discussions will be concentrated on 2-spacing segmented buses, because any claim on such buses can be easily generalized to other segmented buses. Let us label the $\frac{p}{2} - 1$ switches by numbers starting from 0 to $\frac{p}{2} - 1$ in order from left to right. The i-th switch is denoted as S_i.

5.2.1 Simulation of Linear Array

Consider the problem of simulating a p-processor linear array, where $p = 2^n$, by a 2-spacing segmented bus $B(p)$. In the first step, all switches are set *off* (by which we mean that the switch disconnects the two basic bus segments adjacent at the switch). All processors P_i such that i being even can communicate with P_{i+1} in parallel. In the second step, we set a switch S_i *on* (by which we mean that the two basic bus segments adjacent at S_i are connected) if and only if the rightmost bit of the binary representation of i is 0. Then, all processors P_i such that (i mod 4 = 1) can communicate with P_{i+1} in parallel. In the third step, we set a switch S_i on if and only if the rightmost bit of its binary label is 1. Then, all processors P_i such that (i mod 4 = 3) can communicate with P_{i+1} in parallel. Therefore, a 2-spacing segmented bus $B(p)$ can simulate any parallel communication step of a p-processor linear array in at most 3 parallel communication steps. To simulate a ring, we can add one more step to let P_0 and P_{p-1} communicate by setting all switches

on. Therefore, we have

Theorem 1 *A 2-spacing segmented bus $B(p)$, $p = 2^n$, can simulate any parallel communication step of a p-processor linear array (resp. ring) in 3 (resp. 4) parallel communication steps.* ■

Let $L' = (P_{i_0}, P_{i_1}, \cdots, P_{i_m})$, where $i_j < i_{j+1} \leq m$, be a linearly ordered subset of the linearly ordered processor set $L = (P_0, P_1, \cdots, P_{p-1})$. A linear array corresponding to L' is a point-to-point network such that P_{i_j} is connected to $P_{i_{j+1}}$, $0 \leq j < m \leq p-1$. A ring corresponding to L' is a linear array corresponding to L' with an additional link that connects P_{i_0} and P_{i_m}. We show that the 2-spacing segmented bus can simulate a linear array (resp. ring) corresponding to any L' efficiently. Let $S' = \{S_{\lfloor i_j/2 \rfloor} | j < m$ and j is even $\}$. If we let all switches that are not in S' be on, and ignore the processors that are not in L', then L' and S' define a 2-spacing segment bus (note: the last segment may have only one processor of L'). By controlling the switches in S' in the way that is described for the case of simulating a linear array of $p-1$ processors, the linear array (resp. ring) corresponding to L' can also be simulated efficiently. This generalization of Theorem 1 is stated in the following Corollary.

Corollary 1 *A 2-spacing segmented bus $B(p)$, $p = 2^n$, can simulate any parallel communication step of any linear array (resp. ring) corresponding to a linearly ordered processor subset of $B(p)$ in at most 3 (resp. 4) parallel communication steps.*

■

5.2.2 Simulation of Binary Tree

In addition to supporting efficient broadcasting and multicasting, and various linear array and ring communication patterns, a segmented bus can also simulate tree interconnection structures efficiently. Consider a complete tree T of $2p-1$ processors P'_i, $0 \leq i < 2p-1$. We map P'_i to $P_{f(i)}$ of a segmented bus $B(p)$ using function $f(i) = \lfloor \frac{i}{2} \rfloor$. Clearly, by f,

two processors of T are mapped to one processor of the segmented bus (except that one processor of T is mapped to the last processor of the bus). For $p = 16$, this mapping is shown in Figure 5.8. In this figure, processors bounded by a dashed box are mapped to the same processor of the segmented bus that appears in the same column. In simulating T by $B(p)$, some links of T can be ignored. Such situations occur when two processors, e.g., P'_0 and P'_1, connected by a link in T are mapped to the same processor of $B(p)$. We mark each of the remaining links of T by an integer pair (x, y) in a recursive manner as shown in Figure 5.8. Here, x indicates the switching pattern of $B(p)$ used for simulating the communication along the link, and y indicates the step number using pattern x. More specifically, we use the following switching scheme:

- All links in the tree with the same link label (x, y) are simulated by the same step.
- If $x = 1$, then all switches of the 2-spacing segmented bus $B(p)$ are off.
- For $x > 1$, a switch S_j is turned off if and only if the rightmost $x - 1$ bits of the binary value of j are all 1's.

For example, to simulate all links of T that are marked (2,1) and (2,2), we need to use the following switching pattern of $B(p)$: turn off switch S_j if and only if the rightmost bit of the binary value of j is 1. After setting this pattern, two communications steps are carried out. In the first step, $B(p)$ simulates all links marked (2,1), and in the second step $B(p)$ simulates all links marked (2,2).

Theorem 2 *A 2-spacing segmented bus $B(p)$, $p = 2^n$, can simulate any parallel computation step and communication step of a complete binary tree T of $2p - 1$ processors in at most two parallel computation steps and at most $2\log_2 p - 1$ parallel communication steps, respectively.*

Proof: Two parallel computation steps are needed to simulate one computation step of T because two processors in the tree are mapped to one bus processor. We prove the

theorem by showing that $2^n - 1$ steps are sufficient for simulating any communication step of T using the switching scheme given above.

For $n = 2$, our claim is obviously true. Suppose that our claim is true for $n = k$ and consider the case that $n = k + 1$. Now we use two segmented buses of size $p = 2^k$ to construct a larger segmented bus of size $q = 2^{k+1}$. Similarly, we use two complete binary trees of size $2p - 1$ to construct a larger tree of size $2q - 1$, as shown in Figure 5.9.

In the first $2k - 1$ steps, switch $S_{2^{k-1}-1}$ is set off because the rightmost $k - 1$ bits of the binary value of $S_{2^{k-1}-1}$ are all 1's (see Figure 5.9). By the hypothesis, at the end of $2k - 1$ simulation steps, the left and the right subtrees of the root are simulated by the inductive hypothesis. In step $2k$, the switch $2^{k-1} - 1$ is still off and the link labeled $(k, 2)$ is simulated. In step $2k+1$, the link labeled $(k+1, 1)$ is simulated. Notice that the largest label of the switches is $2^k - 2$ whose binary value has only $(k - 1)$ 1's, and a switch S_j is set off if and only if the rightmost $k + 1$ bits of the binary value of j are all 1's. Thus, in simulating link $(k + 1, 1)$, all switches are set on. This is exactly what is needed by our mapping. Therefore, $B(2^{k+1})$ can simulate a maximum parallel communication step of a complete binary tree of $2^{k+1} - 1$ processors in $2k + 1 = 2(k + 1) - 1$ steps. This completes the induction and the proof of the theorem. ■

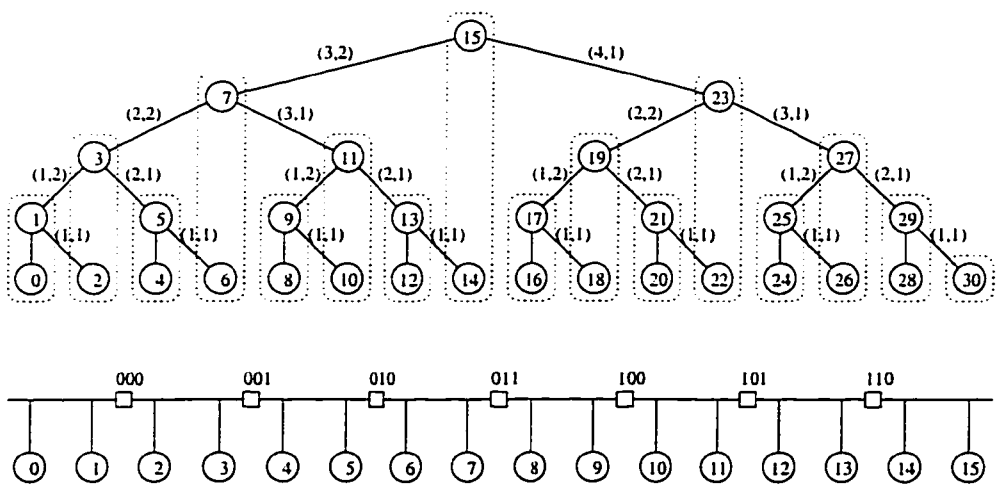


Figure 5.8: Simulation of a complete binary tree by a 2-spacing segmented bus.

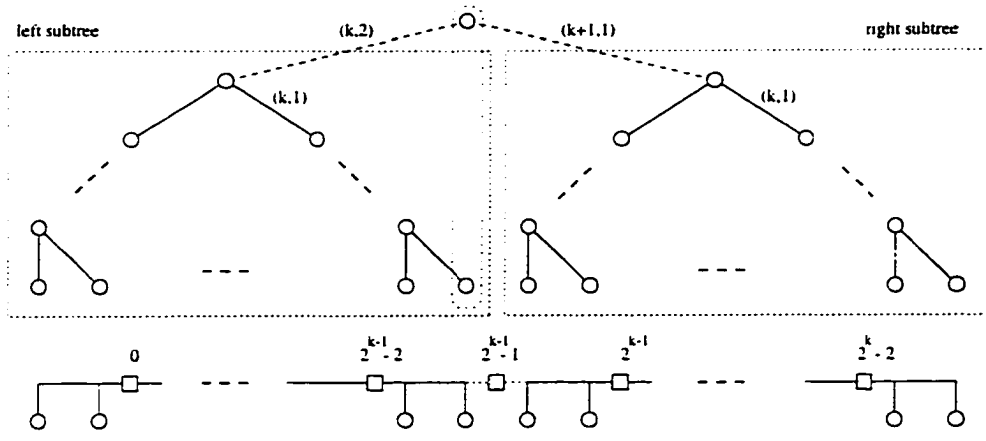


Figure 5.9: Recursive constructions of complete binary tree and 2-spacing segmented bus.

5.2.3 Simulation of X-tree

A useful variation of the tree structure is the X-tree. An X-tree is a supergraph of a complete binary tree with links added to connect consecutive processors on the same level of the tree. For example, a 16-leaf X-tree is shown in Figure 5.10. Using Corollary 1 and Theorem 2, it is easy to derive the following claim.

Corollary 2 *A 2-spacing segmented bus $B(p)$, $p = 2^n$, can simulate any parallel computation step and communication step of an X-tree of $2p-1$ processors in at most two parallel computation steps and at most $5(\log_2 p - 1) + 1$ parallel communication steps, respectively.*

Proof: Again, the simulation of the parallel computation steps is obvious. We only consider the simulation of the parallel communication steps. By Theorem 2, we need $2n-1$ parallel communication steps to simulate the tree links. Consider the simulation of horizontal links. From Corollary 1, it takes three parallel communication steps to simulate the horizontal links in each level of the X-tree except for the highest three levels. The highest level has one processor, so no step is needed for this level. The second highest level has two processors, and and one step is required for simulating the link connecting them. The third highest level has four processors, and two 2 steps are sufficient for simulating the three horizontal links. Therefore, simulating all horizontal links takes $3(n-2)+3$

steps. In summary, to a maximum parallel communication step of the whole X-tree,

$$2n - 1 + 3(n-2) + 3 = 5(n-1) + 1$$

parallel communication steps are sufficient. ■

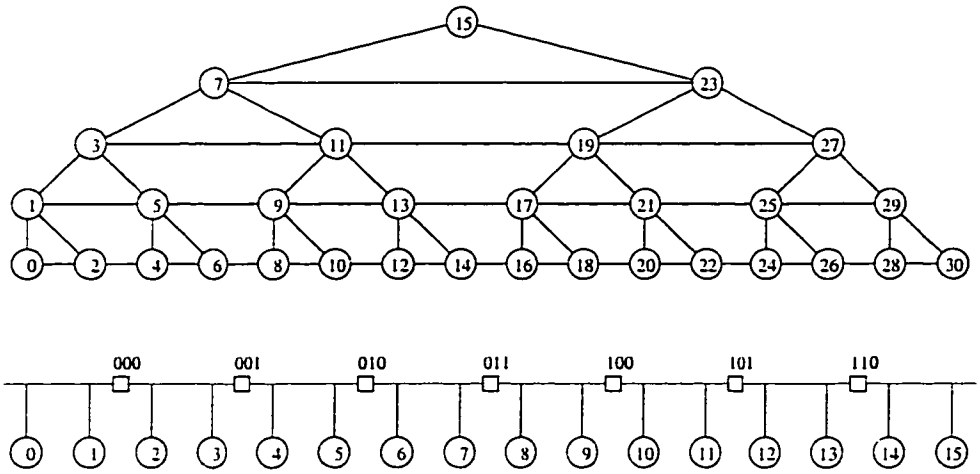


Figure 5.10: Simulation of an X-tree by a 2-spacing segmented bus.

5.2.4 Simulation of One-dimensional Multigrid

A subgraph of an X-tree, called a one-dimensional multigrid, has been proved useful for implementing efficient parallel matrix algorithms [40]. It has $2^{n+1} - 1$ processors, divided into $n + 1$ levels. Level m, $0 \leq m \leq n + 1$, is a linear array of 2^m processors. The j-th processor on level m is connected to the $2j$-th processor on level $m + 1$. A $(2^4 - 1)$-processor one-dimensional multigrid is shown in Figure 5.11 (a). We map the processors of a one-dimensional multigrid to the processors of $B(p)$ in the way shown in Figure 5.11 (b). Compare Figure 5.11 (b) with Figure 5.8. If we delete all horizontal links of 5.11 (b), we obtain a subgraph of Figure 5.8. The missing non-horizontal links are those that are either not labeled or labeled $(x, 2)$. Using the same switching scheme for simulating the tree, but without simulating the missing links, $\log_2 p$ steps are sufficient for $B(p)$ to simulate all non-horizontal links of the one-dimensional multigrid of $2^p - 1$ processors. By the proof of Corollary 2, it takes $3(\log_2 p - 2) + 3$ steps to simulate all the horizontal links. Therefore the total number of steps for simulating a maximum communication step of a

one-dimensional multigrid is $\log_2 p + 3(\log_2 p - 2) + 3 = 4\log_2 p - 3$. In summary, we have the following claim:

Theorem 3 *A 2-spacing segmented bus $B(p)$, $p = 2^n$, can simulate any parallel computation step and communication step of a one-dimensional multigrid $2p-1$ processors in at most two parallel computation steps and at most $4\log_2 p - 3$ parallel communication steps, respectively.*

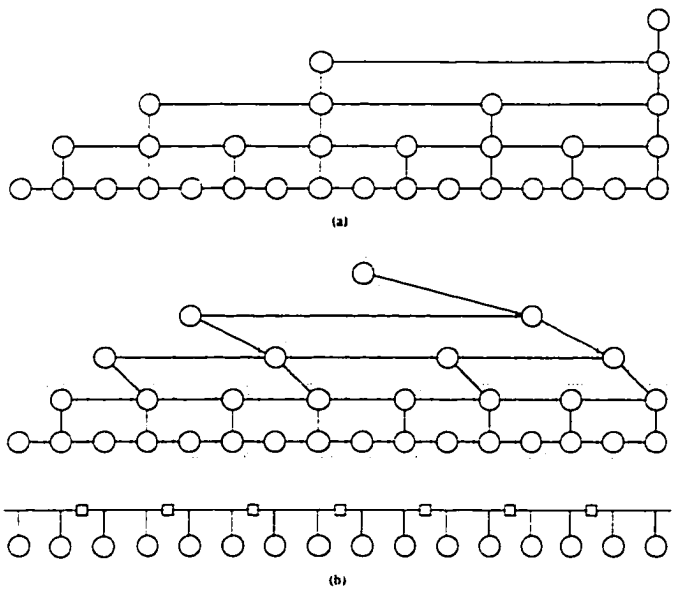


Figure 5.11: Simulation of a 1-D multigrid by a 2-spacing segmented bus. (a) 1-D multigrid of 31 processors. (b) Processor mapping to $B(16)$

Now, consider the k-D MCSB. By Theorem 1, we know that an $M_k(2^n, 2)$ can simulate a communication step of a k-dimensional 2^n-ary mesh and torus (which is also called a 2^n-ary k-cube) efficiently.

Corollary 3 *An MCSB $M_k(2^n, 2)$ can simulate any parallel computation step and any parallel communication step of a k-dimensional 2^n-ary mesh (resp. torus) in one parallel computation step and at most three (resp. four) parallel communication steps, respectively.*

Proof: Directly from Theorem 2. ■

5.2.5 Simulation of Mesh-of-tree

By adding a bus to each row and each column of a conventional two dimensional mesh, one can obtain a mesh with improved data broadcasting performance. Such a model has been considered for many applications (e.g. [60, 74, 81]). We refer to this interconnection structure as the mesh-connected computer with multiple broadcasting (MCCMB). Clearly, a k-D MCSB is more powerful than its corresponding k-D MCCMB. We demonstrate this by showing that the a k-D MCSB can efficiently simulate several useful machine models.

An important parallel computing model is the mesh-of-trees (MOT). A $2^n \times 2^n$ two-dimensional MOT is constructed from a $2^n \times 2^n$ two-dimensional grid of processors by adding processors and links to form a complete tree in each row and each column of the grid. The leaves of these trees are the original processors in the grid. Similarly, a three-dimensional MOT is constructed from a three-dimensional grid of processors by adding processors and links to form a complete tree whose leaves are the grid processors with the same two indices. There are $3 \cdot 2^{2n} - 2^{n+1}$ processors in the $2^n \times 2^n$ MOT, and there are $4 \cdot 2^{3n} - 3 \cdot 2^{2n}$ processors in the $2^n \times 2^n \times 2^n$ MOT. A 4×4 MOT is shown in Figure 5.12. In [40], it is shown that the mesh-of-trees is a versatile machine model. Many problems can be solved using the MOT in polylogarithmic time. The following claim is derived by using the processor mapping of the proof of Theorem 2 for all the trees of MOT.

Corollary 4 *An MCSB $M_2(2^n, 2)$ can simulate any parallel computation step and any parallel communication step of a $2^n \times 2^n$ 2-D MOT in at most 3 parallel computation steps and at most $2n-1$ parallel communication steps, respectively. An MCSB $M_3(2^n, 2)$ can simulate any parallel computation step and any parallel communication step of a $2^n \times 2^n \times 2^n$ 3-D MOT in at most 4 parallel computation steps and at most $2n-1$ parallel communication steps, respectively.* ■

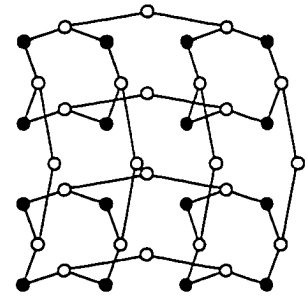


Figure 5.12: A 4×4 mesh-of-trees.

5.2.6 Simulation of Pyramid

Two related classes of point-to-point networks, multigrids and pyramids [40], have been proven useful for many applications. For example, a common approach to solving a system of partial differential equations is the use of finite difference methods. A variety of algorithms for finite difference problems have been devised for multigrid structures. Pyramids are also useful for parallel image processing. MCSB's can be used to simulate these two classes of interconnection networks efficiently. The $2^n \times 2^n$ multigrid network consists of $n+1$ levels of 2-D processor arrays; the array at level m, $0 \leq m \leq n$, is of size $2^m \times 2^m$. The processor with indices (i, j) on the $2^m \times 2^m$ array is connected to the processor with indices $(2i, 2j)$ on the $2^{m+1} \times 2^{m+1}$ array. The $2^n \times 2^n$ pyramid network consists of $n+1$ levels of 2-D processor arrays; the array at level m, $0 \leq m \leq n$, is of size $2^m \times 2^m$. The processor with indices (i, j) on the $2^m \times 2^m$ array is connected to the processors with indices $(2i-1, 2j-1)$, $(2i-1, 2j)$, $(2i, 2j-1)$, and $(2i, 2j)$ on the $2^{m+1} \times 2^{m+1}$ array. A 4×4 multigrid network and a 4×4 pyramid network are shown in Figures 5.13 and 5.14, respectively. The Multigrid network and pyramid network are the natural two-dimensional generalizations of the one-dimensional multigrid and X-tree, respectively; they are closely related. It is known that a $2^n \times 2^n$ multigrid network can simulate a $2^n \times 2^n$ pyramid network with a slowdown factor three. communication slowdown factor. Thus, if an MCSB $M_2(2^n, 2)$ can simulate a $2^n \times 2^n$ multigrid network with a slowdown factor c, then $M_2(2^n, 2)$ can simulate a $2^n \times 2^n$ pyramid network with a slowdown factor $3c$.

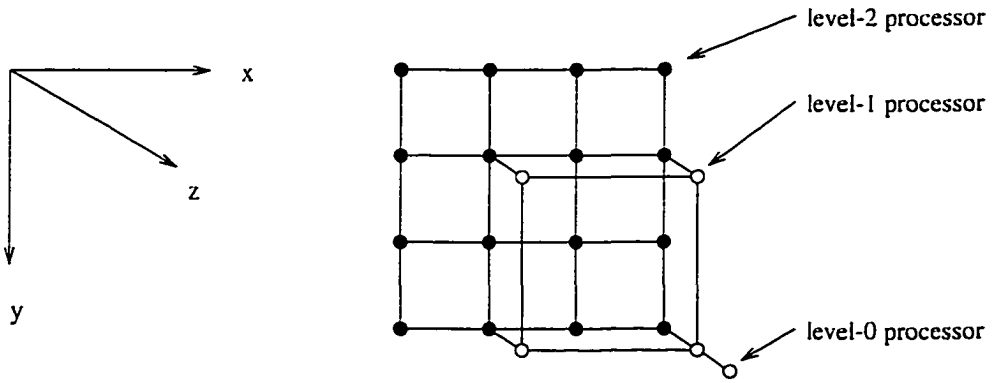


Figure 5.13: A 4 × 4 multigrid.

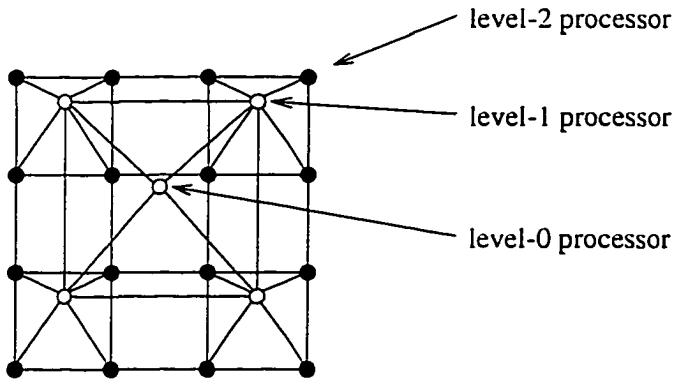


Figure 5.14: A 4 × 4 pyramid.

5.2.7 Simulation of High-dimensional Multigrid

Consider simulating a $2^n \times 2^n$ multigrid network by an MCSB $M_2(2^n, 2)$. Imagine that we cut a $2^n \times 2^n$ multigrid network into slices, then we obtain a set of subgraphs of one-dimensional multigrids. For each slice, we use the processor mapping method for simulating a one-dimensional multigrid by a 2-spacing segmented bus. Then, it is not difficult to derive the following claim.

Theorem 4 *An MCSB $M_2(2^n, 2)$ can simulate any parallel computation step and any parallel communication step of a $2^n \times 2^n$ multigrid network and a $2^n \times 2^n$ pyramid net-*

work in at most two parallel computation steps and $O(n)$ parallel communication steps, respectively.

Proof:

From above discussion, we only need the proof for the case of a multigrid. We use (x, y, z) to represent the indices (coordinates) of a processor in a $2^n \times 2^n$ multigrid (refer to Figure 5.13). We say that a link of the multigrid is a dimension x (resp. y and z) link if it connects two processors of the different x-coordinates (resp. y-coordinates and z-coordinates). Let L_x, L_y and L_z denote the links of dimension x, y and z, respectively. We partition the links of a $2^n \times 2^n$ multigrid into two subsets: $L_{xz} = L_x \cup L_z$ that contains all links of dimensions x and z, and the set L_y of links of dimension y. We cut the $2^n \times 2^n$ multigrid by xz planes to obtain a set of partial grids, each being a subgraph of a one-dimensional multigrid. Clearly, all links on these planes are in L_{xz}, and the remaining links are in L_y. We use the processor mapping method shown in Figure 5.11 to map the processors in each partial grid to a row bus of $M_2(2^n, 2)$ (note: at most two processors of the multigrid are mapped to one processor of $M_2(2^n, 2)$). Furthermore, a communication step on all links in L_{xz} can be simulated by row buses in $4n - 3$ steps (Theorem 3). The links in L_y are mapped to column buses of $M_2(2^n, 2)$. Since in the multigrid there are $n-1$ levels that contain links of L_y, a step of communications on links of L_x requires $3(n-2)+3$ steps (by Corollary 2). Since the simulations of links in L_{xz} and L_y are independent from each other, a maximum communication step on the $2^n \times 2^n$ multigrid can be simulated by $M_2(2^n, 2)$ in at most $\max\{4n - 3, 3n - 3\} = 4n - 3$ steps. The processor mapping used ensures that a computation step of the multigrid can be simulated by at most two steps by $M_2(2^n, 2)$.

■

All above discussions are based on 2-spacing segmented buses. If 2^i-spacing segmented buses, $i > 1$, are used, all the claimed simulation performances degrade as i increases. For example, a 2^i-spacing segmented bus can simulate any parallel communication of a

p-processor linear array (resp. ring) in $2^i + 1$ (resp. $2^i + 2$) parallel communication steps. Similar performance degradations occur when using segmented buses of less number of switches to simulate other interconnection structures. We would like to point out that $MCSB$ is much sparser than the point-to-point networks we considered. For example, a $2^n \times 2^n$ torus has $O(2^{2n})$ processors and $O(2^{2n})$ links, whereas a $2^n \times 2^n$ MCSB has 2^{2n} processors and $2n$ segmented buses. All the simulation results discussed in this section are asymptotically optimal.

5.3 Parallel Prefix Computation

Given a sequence $S = (a_0, a_1, \cdots, a_{N-1})$ of N elements in a domain D, and an associative operation $\otimes$ on D, the prefix problem is to compute $y_i = a_0 \otimes a_1 \otimes \cdots \otimes a_i$ for $0 \leq i < N$. The prefix computation is a fundamental problem in parallel computing. It has a wide range of applications such as processor allocation, data distribution and alignment, data compaction, job scheduling, sorting, packet routing, matrix computation, linear recurrence, polynomial evaluation, graph algorithms, general Horner expressions and general arithmetic formulae. Refer to [28, 40] for references of these applications.

There has been much research on parallel prefix computation (PPC), and efficient PPC algorithms have been proposed for various parallel computing models such as PRAM, tree-like machines, hypercube, mesh-like machines, and the shuffle-exchange machine. For a good survey of previous PPC results, refer to [28]. In this section, we want to show how PPC can be efficiently realized on the architecture based segmented bus.

5.3.1 Prefix on 1-D array

Recursive doubling

Horng [28] introduced a concept called recursive doubling for the parallel computation of prefix problem. The idea is to break the calculation of one term into two complex subterms, as shown in Fig. 5.15. He gave a proof that prefix can be correctly calculated by recursive doubling. But we give a more concise and direct proof in the following.

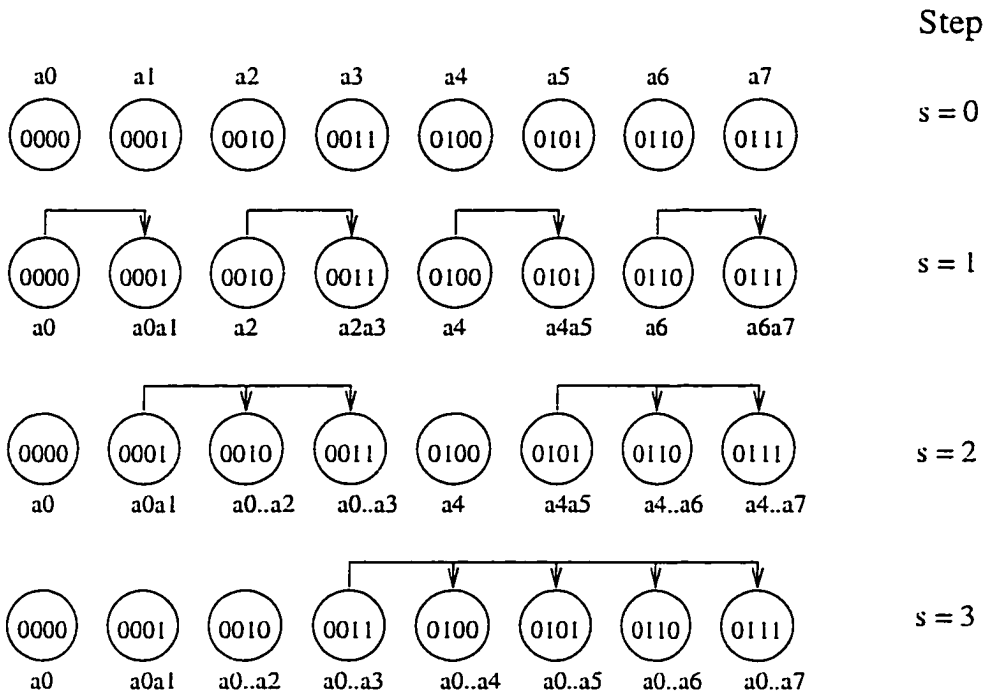


Figure 5.15: Parallel prefix computation using recursive doubling.

The following notation will be used in this paper.

(i) Let i_j denote the jth bit of the binary representation, $i_{d-1}i_{d-2}...i_j...i_1i_0$, of i.

(ii) Let P_i^s be a variable located in processor i that contains a segment of prefix computed at step s.

(iii) Let s denote the parallel computation step.

(iv) Let p denote the number of processors.

(iv) The following definition and properties are from Horng [28].

$sub(i, j, 0) = i_{d-1}i_{d-2}...i_j0_{j-1}0_{j-2}...0$

$sub(i, j, 0) = sub(i, j + 1, 0)$, if $i_j = 0$

$sub(sub(i, j, 0) - 1, j, 0) = sub(i, j + 1, 0))$, if $i_j = 1$

Lemma 1

$$P_i^s = P_i^{s-1} \quad if \ i_{s-1} = 0$$

$$P_i^s = P_{sub(i,s-1,0)-1}^{s-1} \otimes P_i^{s-1} \quad if \ i_{s-1} = 1$$

Lemma 1 can be justified directly from the observation of Figure 5.15.

Lemma 2 $P_i^s = x_{sub(i,s,0)} \otimes x_{sub(i,s,0)+1} \otimes \cdots \otimes x_{i-1} \otimes x_i$

Proof: The proof is done by induction on s.

Basics step: When $s = 0, i = sub(i, s, 0)$. Therefore, $P_i^0 = x_i$. That is, at the beginning, x_i is assigned to processor i.

Induction step: Assume that when $s = m$, the lemma is true. That is,

$P_i^m = x_{sub(i,m,0)} \otimes x_{sub(i,m,0)+1} \otimes \cdots \otimes x_{i-1} \otimes x_i$.

We need to prove that when $s = m + 1$, it is also true. Lemma 1 and the properties of *sub* will be used in the following steps.

If $i_m = 0$, then

$$\begin{aligned} P_i^{m+1} &= P_i^m \\ &= x_{sub(i,m,0)} \otimes x_{sub(i,m,0)+1} \otimes \cdots \otimes x_{i-1} \otimes x_i \\ &= x_{sub(i,m+1,0)} \otimes x_{sub(i,m+1,0)+1} \otimes \cdots \otimes x_{i-1} \otimes x_i. \end{aligned}$$

If $i_m = 1$, then

$$\begin{aligned} P_i^{m+1} &= P_{sub(i,m,0)-1}^m \otimes P_i^m \\ &= x_{sub(sub(i,m,0)-1,m,0)} \otimes x_{sub(sub(i,m,0)-1,m,0)+1} \otimes \cdots \otimes x_{sub(i,m,0)-1} \otimes \\ &\quad x_{sub(i,m,0)} \otimes x_{sub(i,m,0)+1} \otimes \cdots \otimes x_{i-1} \otimes x_i \\ &= x_{sub(i,m+1,0)} \otimes x_{sub(i,m+1,0)+1} \otimes \cdots \otimes x_{i-1} \otimes x_i. \end{aligned}$$

■

Theorem 5 $y_i = P_i^{\lceil \log(i+1) \rceil}$

Proof: In Lemma 2, let $sub(i, s, 0) = 0$, which can be guaranteed by $s = \lceil \log(i+1) \rceil$.

■

The algorithm

Use the recursive doubling and the above architecture, we can have the following algorithm to do the parallel prefix computing.

procedure parallel_prefix;

1. begin
2. parfor $0 \leq i < N$ do
3. begin
4. $y_i = a_i$ /* initialization */
5. for $s = 1$ to $\log N$ do
6. begin
7. if s=1 close all the switches
8. else close switch $w = \frac{i}{2}, i_0 = 0$ and $w_{s-2} = 1$
9. $x_i \longleftarrow y_{sub(i,s-1,0)-1}$, $i_{s-1} = 1$
10. $y_i = x_i \otimes y_i$
11. end
12. end
13. end

5.4 $k-D$ mesh with segmented buses

The segmented buses can be easily integrated into a higher dimensional mesh. Figure 5.16 shows a $3-D$ mesh enhanced with segmented buses. The corresponding bus enumerations are shown in Figure 5.17 and the notations will be explained later on. Note that in the figure we only show the segmented buses necessary for prefix computation. We do not show the buses in detail either, since each bus is the same as a $1-D$ bus.

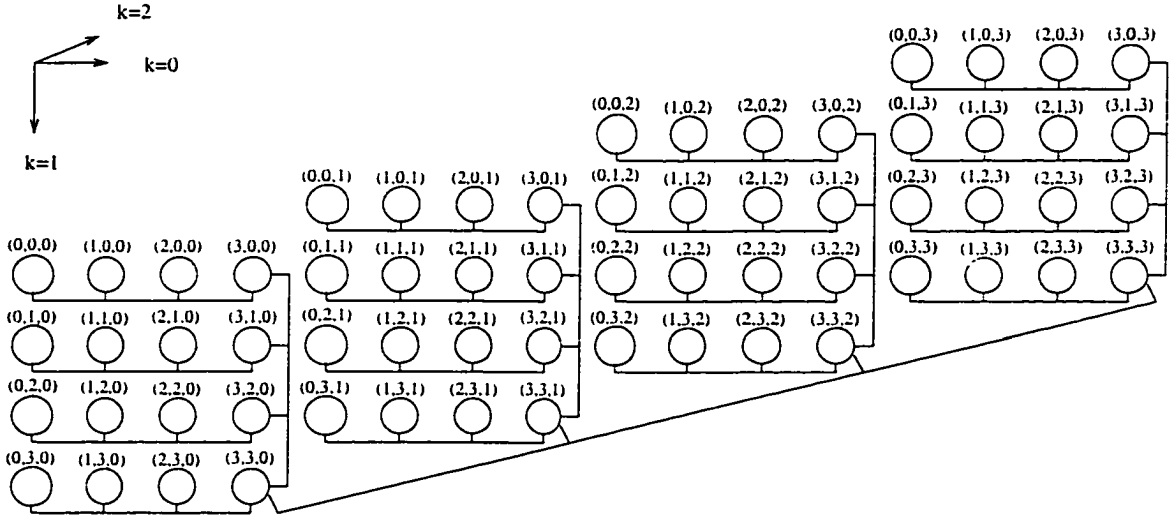


Figure 5.16: A $3-D$ mesh with segmented buses.

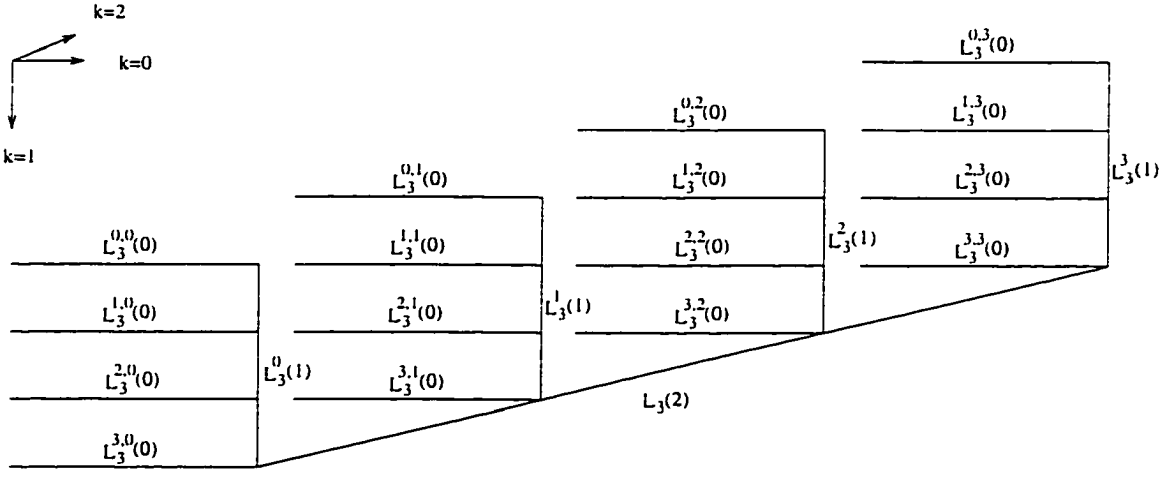


Figure 5.17: The bus notations for the $3-D$ mesh shown in Figure 5.16.

For the convenience of description, we introduce some more definitions and notations. Let M_k be a k-dimensional mesh with each dimension having n processors. There are a total of $p = n^k$ processors. Each processor is identified by $P(i_0, i_1, ..., i_{k-1})$. Given $m \leq k$, an $m-D$ submesh can be specified by $M_k(n-1, ... n-1, i_m, i_{m+1}, ..., i_{k-1})$ where $0 \leq i_m, i_{m+1}, ..., i_{k-1} < n$

In this section, a subscript is also used for denoting the dimension to which each index belongs. For example, in $P(i_0, i_1, ..., i_j, ..., i_{k-1})$, i_j is an index in dimension j. For the sake of simplicity, a subscript will be omitted if it can be infered from the context.

Consider a prefix $a_0 \otimes a_1 \otimes ... \otimes a_j$, $0 \leq j \leq n^k - 1$. Initially, we have the following mapping relationship between the elements a_j and the processors:

$$a_{i_{k-1}*n^{k-1}+i_{k-2}*n^{k-2}+\cdots+i_2*n^2+i_1*n+i_0} \longrightarrow P(i_0, i_1, ..., i_{k-1})$$

That is, we fill the processors using row major order.

We will also denote an element by the coordinate of the processor to which this element is initially assigned. For example, $a_0 = a_{0,0,0...0}, a_1 = a_{1,0,0...0}$.

A submesh head processor (SHP) is a processor with the maximum coordinates in a given submesh. For example, in Figure 5.16, $P(3,1,0)$ is a $1-D$ SHP, $P(3,3,0)$ is a $2-D$ SHP and $P(3,3,3)$ is a $3-D$ SHP. Generally, $P(n-1, ... n-1, i_m, i_{m+1}, ..., i_{k-1}), 0 \leq i_m, i_{m+1}, ..., i_{k-1} < n-1$, is an $m-D$ SHP. During the $k-D$ parallel prefix computing, the communication is mainly done through these SHPs. A processor may serve as an SHP for different submeshes. For example, $P(3,3,3)$ is a SHP of submeshes of dimension from 1 to 3. SHPs that are on the same dimension axis are *siblings* of each other. An SHP P is the *parent* of those SHPs, called *child* SHPs, which are within the submesh headed by P and are one dimension lower. Specifically, the head processor $P(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$ of an $m-D$ submesh $M(m)$ is the parent processors of those SHPs $P(n-1, n-1, ..., n-1, t, i_m, i_{m+1}, ..., i_{k-1})$, $0 \leq t < n$, which are in $M(m)$ and on the dimension $m-1$ axis. For example, $P(3,3,0)$ is the parent of $P(3,0,0)$, $P(3,1,0)$ and $P(3,2,0)$, and $P(3,3,3)$ is the parent of $P(3,3,0)$, $P(3,3,1)$ and $P(3,3,2)$.

$L_k(m)$, $0 \leq m < n$, denotes a set of segmented buses connecting:

$$\begin{array}{llll} P(n-1, n-1, ..., n-1, & 0, & i_{m+1}, i_{m+2}, ..., i_{k-1}) \\ P(n-1, n-1, ..., n-1, & 1, & i_{m+1}, i_{m+2}, ..., i_{k-1}) \\ ... & ... & ... \\ P(n-1, n-1, ..., n-1, & n-1, & i_{m+1}, i_{m+2}, ..., i_{k-1}) \end{array}$$

Each bus in $L_k(m)$ represents a permutation of $\{ i_{m+1}, i_{m+2}, ..., i_{k-1} \}$. A single bus will be denoted by $L_k^{i_{m+1}, i_{m+2}, ..., i_{k-1}}(m)$. Refer to Figure 5.17 for some example bus notations. Each bus in $L_k(m)$ connects a set of $m-D$ SHPs on a dimension m axis. Each of the buses $L_k(m)$ has a *head processor* that has the biggest coordinate in dimension m and a *tail processor* that has the smallest coordinate.

We define a *collection step* on a bus $L_k(m)$ as all the operations needed for a $1-D$ prefix computation on that bus. A *broadcasting step* on a bus $L_k(m)$ means that the bus head processor broadcasts a message or a subprefix to each of the processors on the bus.

SD_m denotes a Synthesized-from-Descendent subprefix which contains all the elements in an $m-D$ submesh. $SD_m(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$, where $0 < m \leq n$, denotes a subprefix for an $m-D$ submesh located on the SHP $P(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$. $P(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$ collects its SD_m subprefix from $P(n-1, n-1, ..., n-1, t, i_m, i_{m+1}, ..., i_{k-1})$, $0 \leq t < n$, using the bus $L_k^{(i_m, i_{m+1}, ..., i_{k-1})}(m-1)$. For example, $P(3,3,0)$ constructs its SD_2 from $P(3,0,0)$, $P(3,1,0)$ and $P(3,2,0)$ using the $L_3^0(1)$ bus as shown in Figure 5.16.

SS_m denotes a Synthesized-from-Sibling subprefix that an $m-D$ SHP receives from its younger siblings on the dimension m axis. That is, $P(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$ collects SS_m messages from processors $P(n-1, n-1, ..., n-1, t, i_{m+1}, ..., i_{k-1}), 0 \leq t < i_m$, through the bus $L_k^{i_{m+1}, ..., i_{k-1}}(m)$. Likely, we use $SS_m(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$ denotes the subprefix located on the processor $P(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$.

IP_m denotes an Inherited-from-Parent subprefix that an $m-D$ submesh head processor receives from its parent which broadcasts it. That is, $P(n-1, n-1, ..., n-1, t, i_{m+1}, i_{m+2}, ..., i_{k-1}), 0 \leq t < n$, receives an IP_m subprefix from $P(n-1, n-1, ..., n-1, n-1, i_{m+1}, i_{m+2}, ..., i_{k-1})$ which broadcasts IP_m on the bus $L_k^{i_{m+1}, ..., i_{k-1}}(m)$.

Now we use an example to clarify the above definitions and to help understand the following proof. We consider a $3-D$ prefix computation. Initially all the elements are assigned to its corresponding processors as described before. We divide the computation into two phases, collecting phase and broadcasting phase. Figure 5.18 shows the intermediate states of the collecting phase. The collecting phase is further divided into three collection steps, each for one dimension. Referring to Figure 5.18, at the end of the collection step on dimension 0 or on the buses $L_3(0)$, each processor on a bus $L_3(0)$ has a proper subprefix computed in the same way as with the one-dimensional case; each $1-D$ SHP has an SD_1 subprefix. Next, the computation is done in dimension 1 or on the buses $L_3(1)$. At the end of computation in this dimension, each processor except the tail processor on an $L_3(1)$ bus has an SS_1 subprefix. Each bus head processor, which is a $2-D$ submesh head processor, has an SD_2 subprefix. The same collection operations are done in dimension 2 and the collecting phase ends.

The broadcasting phase is done in a top-down fashion as shown in Figure 5.19. First, each $2-D$ SHP broadcasts its SS_2 subprefix along its corresponding $L_3(1)$. The receiving processors, which are $1-D$ submesh head processors, receive the subprefix and save it in $IP1$. Next each $1-D$ submesh head processor combines its IP_1 and SS_1 subprefixes and broadcasts the resulting subprefix on the its corresponding $L_3(0)$ bus. Each receiving processor combines the received subprefix and its previous subprefix to get a complete prefix. The $3-D$ prefix computation is finished. Note for $3-D$ prefix computation, we need two broadcasting step.

With the concept used in the above example, it is easy to prove the following theorem.

Theorem 6 *A prefix of length $p = n^k$ can be computed in time $O(\log p)$.*

Proof: The theorem is proved using two phase induction on dimension k.

Inductive hypothesis

We imagine that a $k-D$ mesh consists of n submeshes of $k-1$ dimension, referring to Figure 5.20. Note the simplified notations used in the figure. $SD_m^{i_m,i_{m+1},\ldots,i_{k-1}}$ is used

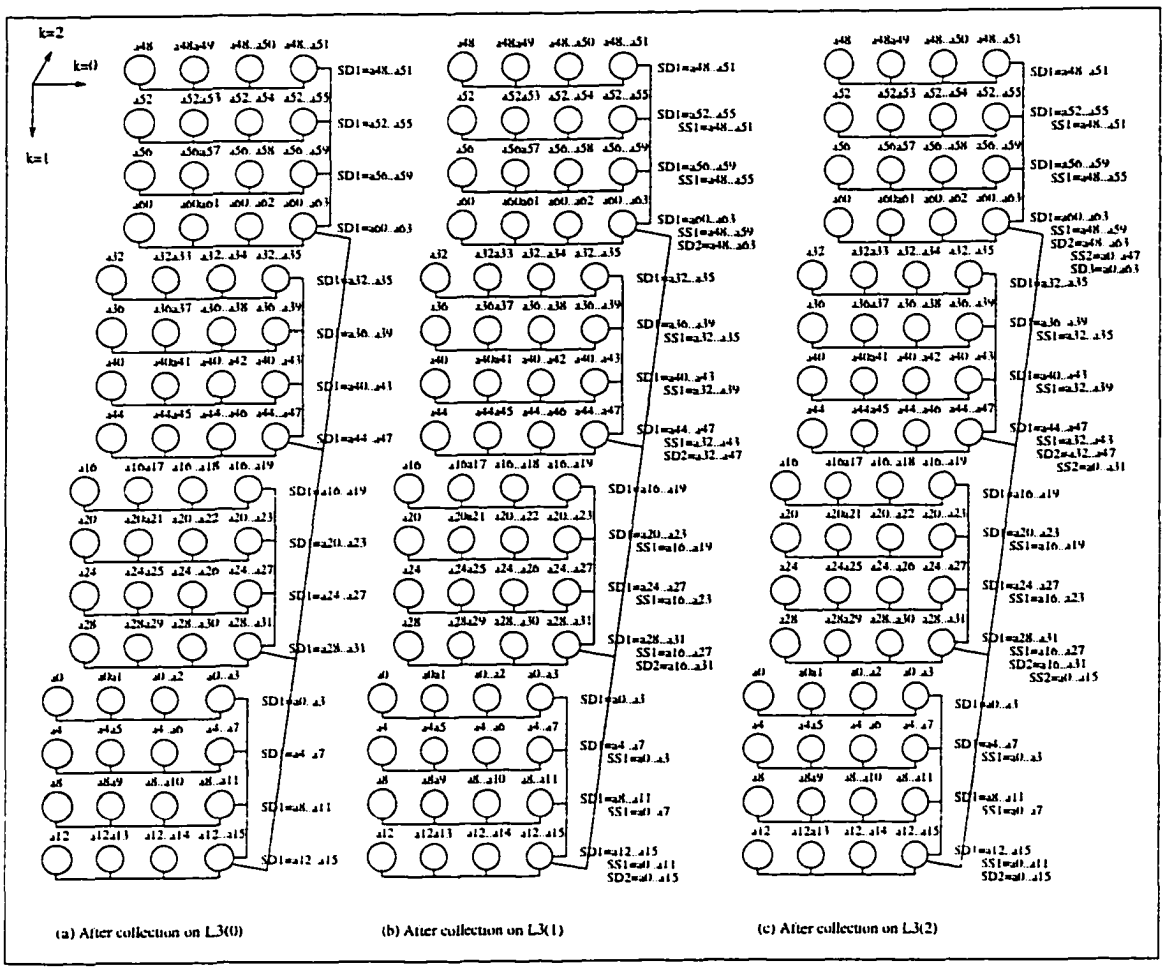


Figure 5.18: A The collecting phase of $3-D$ prefix computation.

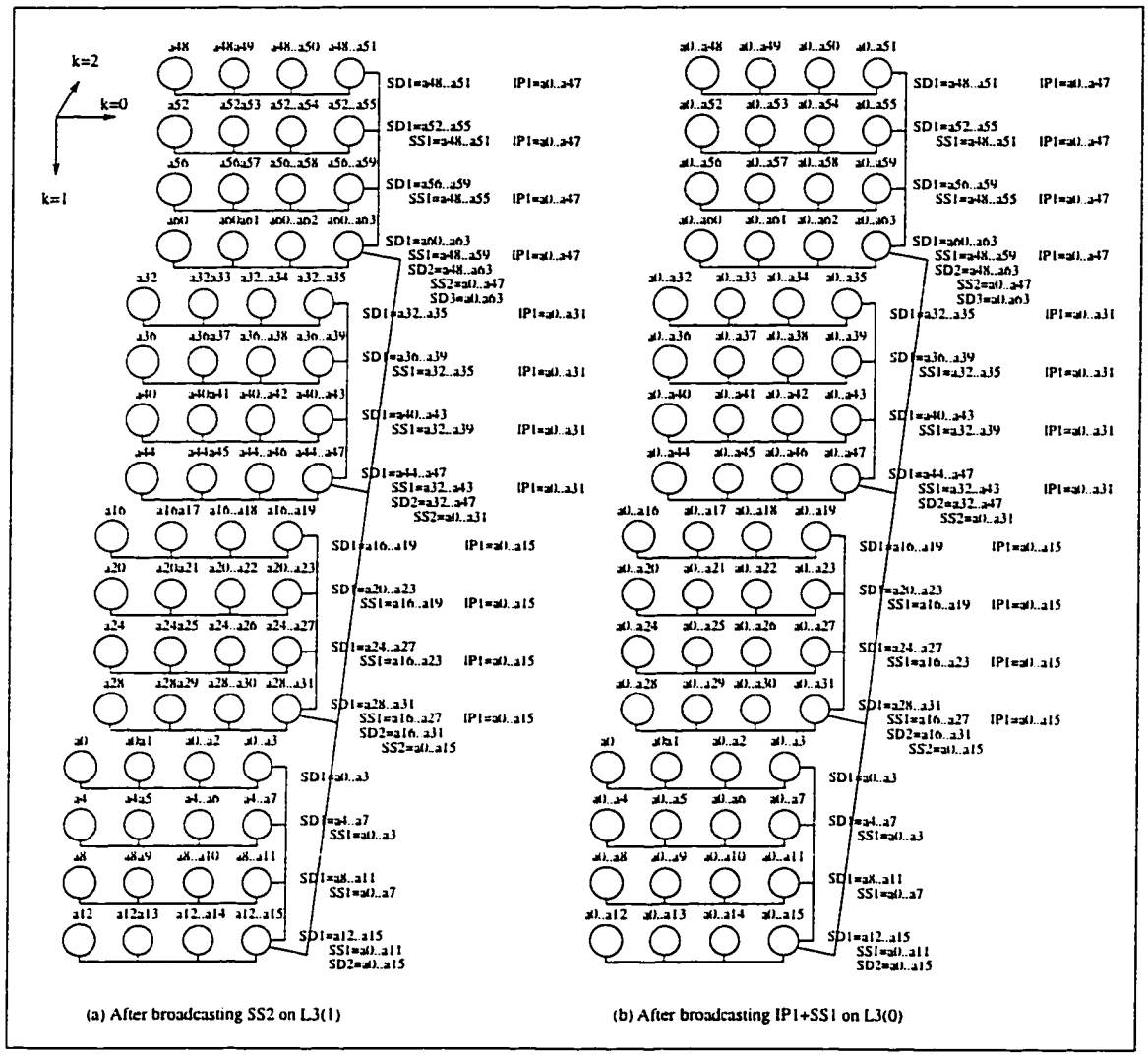


Figure 5.19: A The broadcasting phase of $3-D$ prefix computation.

for $SD_m(n-1, n-1, ..., n-1, i_m, i_{m+1}, ..., i_{k-1})$. We do the same with SS_m and M_m. First let us consider the collecting phase. In the collecting phase, each SHP sends its SD subprefix to its older sibling. The receiving sibling receives the subprefix as an SS subprefix, combines it with its own SD subprefix and sends the combined subprefix to its older siblings. The process is exactly the same as with the one-dimensional case. The collecting phase starts with dimension 0 towards higher dimensions. At the end of the collecting phase, assume that each SHP of $k-1$ dimensional submesh contains an SD_{k-1} subprefix and, except for the first SHP, also contains an SS_{k-1} subprefix. The time needed for the collecting phase is denoted by $T_c(k)$. Next we consider the broadcasting phase. In the broadcasting phase, which starts with the dimension $k-2$ towards dimension 0, each $k-1$ dimensional SHP broadcasts its SS subprefix to its children. Each child will combine the received subprefix, or IP, with its SS subprefix and broadcasts the resulting subprefix to its children. This process is done recursively until broadcasting is done in the dimension 0 buses. We assume that, at the end of the broadcasting phase, each processor contains a correct prefix. The time needed for broadcasting is denoted by $T_b(k)$. In brief, we have the following three inductive hypotheses:

1. At the end of the collecting phase, SD^j_{k-1} contains a subprefix of all the elements in the submesh M^j_{k-1}.

2. $SS^j_{k-1} = \otimes_{m=0}^{j-1} SD^m_{k-1}$, $0 \leq j < n$. Note SS^{n-1}_{k-1} contains a prefix of all the elements in the $k-D$ mesh.

3. After the SHP of M^j_{k-1} broadcasts its SS^j_{k-1} to its children, eventually each processor in M^j_{k-1} will contain a correct prefix.

Inductive proof

We need to show that the three inductive hypotheses still holds for a $k+1$ dimensional mesh. Again, we construct a $k+1$ dimensional mesh from n $k-D$ meshes, referring to Figure 5.21. Notice the notational differences between Figure 5.20 and Figure 5.21. The

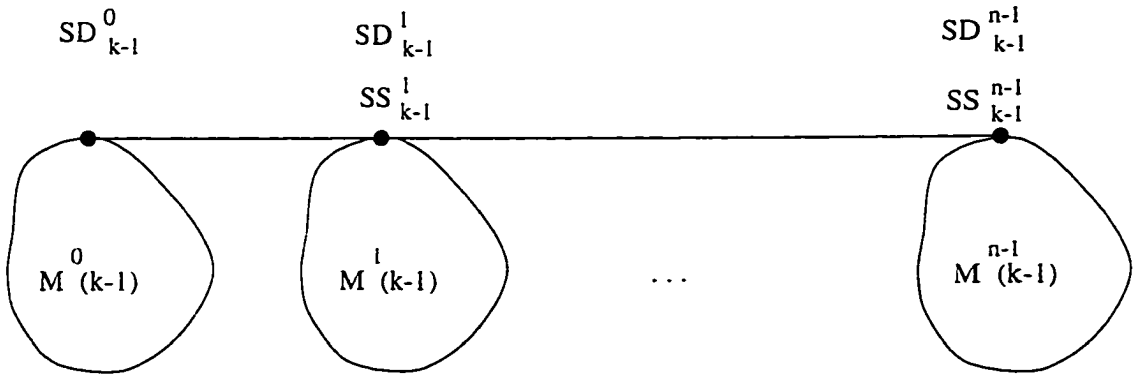


Figure 5.20: A $k-D$ mesh is imagined to consist of n $k-1$ dimensional meshes.

notations in Figure 5.20 are based on a $k-D$ mesh while the notations in Figure 5.21 are based on a $k+1$ dimensional mesh.

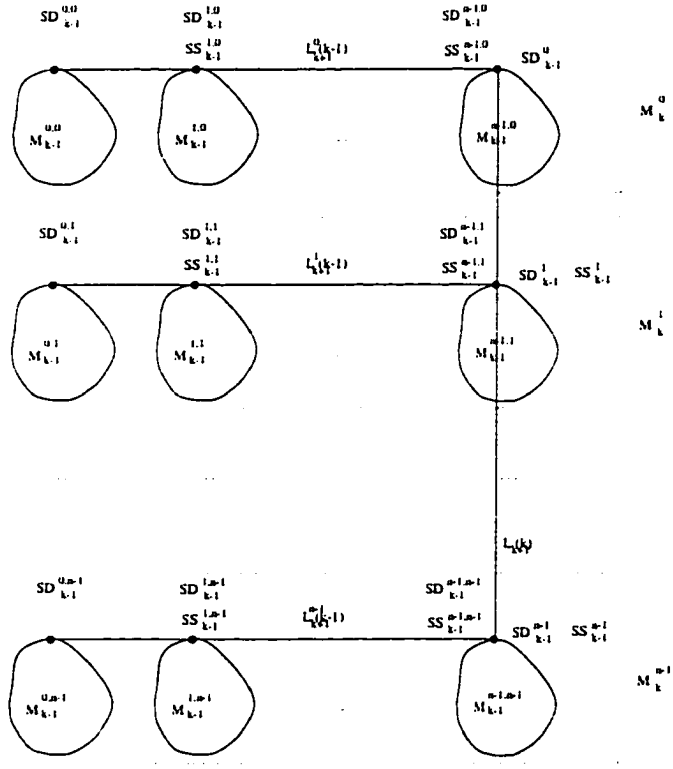


Figure 5.21: A $k+1$ dimensional mesh is imagined to consist of n $k-D$ meshes.

For the hypothesis (1), we simply let $SD_k^j = SS_{k-1}^{n-1,j}$. For hypothesis (2), we do a $1-D$ prefix computation on the bus $L_{k+1}(k)$. It is obvious that $SS_k^i = \otimes_{m=0}^{i-1} SD_k^m$, $0 < i < n$. For hypothesis (3), we let the SHP of M_k^i broadcasts SS_k^i along the bus $L_{k+1}^i(k-1)$. Each

of the receiving processors combines the received subprefix with its own SS subprefix and broadcasts the combined subprefix to its own children. According to the hypothesis (3), at the end of the broadcasting phase, each processor will contain a correct prefix.

For $(k+1)-D$ mesh, we use $T_c(k+1)$ to denote the time needed for the collecting phase and $T_b(k+1)$ for the broadcasting phase. From the above analysis, we have the following iterations:

$T_c(k+1) = T_c(k) + O(\log n)$ with $T_c(1) = \log n$ and

$T_b(k+1) = T_b(k) + 1$ with $T_b(1) = 1$.

Solving the iterations yields $T_c(k) = O(k \log n)$ and $T_b(k) = k$. The total time complexity is:

$T(k) = T_c(k) + T_b(k) = O(k \log n) + k = O(k \log n) = O(\log p)$ ■

Using the above concept, we can come out with the following parallel algorithm for a $k-D$ mesh enhanced with global segmented buses. In the following algorithm, x and y are used as temporary variables to hold a substring. SD_0 is introduced to store the original elements just for the expressing purpose; that is, it can be eliminated.

procedure k_D-parallel_prefix;

1. begin /* collection phase */
2. parfor $0 \leq i_0, i_1, ..., i_{k-1} < n$ do
3. $SD_0(i_0, i_1, ..., i_{k-1}) = a(i_0, i_1, ..., i_{k-1})$/*initialization*/
4. for $j = 0$ to $k-1$ do
5. begin
6. parfor $0 \leq i_{j+1}, i_{j+2}, ..., i_{k-1} < n$ /*for each of the buses $L_k(j)$ */
7. begin
8. parfor $0 \leq i_j < n$ /* 1-D parallel prefix on each bus */

9. begin

10. $y(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) = SD_j(n-1,...,n-1,i_j,i_{j+1}...,i_{k-1})$

11. $SS_j(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) = 0$ /* initialized to empty */

12. Turn off all the switches

13. for $s = 1$ to $\log n$ do

14. begin

15. if $s > 1$, turn on w, $w_{s-2} = 1$

16. $x(n-1,...n-1,i_j,i_{j+1},...i_{k-1}) = SD_j(n-1,...,n-1,sub(i_j,s-1,0)-1,i_{j+1},...,i_{k-1}),(i_j)_{s-1} = 1$

17. $SS_j(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) = x(n-1,...n-1,i_j,i_{j+1},...i_{k-1}) \otimes SS_j(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1})$

18. $y(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) = x(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) \otimes y(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1})$

19. end

20. if(i_j=n-1) $SD_{j+1}(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) = y(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1})$

21. end

22. end

23. end

24. end

25. begin /* broadcasting phase */

26. parfor $0 \leq i_{k-1} < n$

27. do $IP(n-1,...,n-1,i_{k-1}) = 0$ /* initialized to empty */

28. for $j = k-2$ downto 0

29. begin

30. parfor $0 \leq i_{j+1}, i_{j+2}, ..., i_{k-1} < n$

31. begin

32. parfor $0 \leq i_j < n$ do

33. $IP(n-1,...,n-1,i_j,i_{j+1},...,i_{k-1}) = IP(n-1,...n-1,i_{j+1},...,i_{k-1}) \otimes SS_j(n-1,...n-1,i_{j+1},...,i_{k-1})$ /* $P(n-1,...n-1,i_{j+1},...,i_{k-1})$ broadcasts $IP(n-1,...n-1,i_{j+1},...,i_{k-1}) \otimes SS_j(n-1,...n-1,i_{j+1},...,i_{k-1})$ on the buses $L_k(j)$ */

34. end

35. end

36. parfor $0 \leq i_0, i_1, ..., i_{k-1} < n$ do

37. $y(i_0,i_1,...,i_{k-1}) = IP(i_0,i_1,...,i_{k-1}) \otimes a(i_0,i_1,...,i_{k-1}) \otimes SS_0(i_0,i_1,...,i_{k-1})$

38. end

5.5 Summary and Discussions

We proposed a class of reconfigurable buses, the segmented buses, and constructed multidimensional interconnection structures using such buses. We showed that these interconnection structures can be used to build versatile general-purpose parallel machines. To improve bandwidth, reliability, and versatility, one may connect linearly arranged processors by several segmented buses instead of one. Interconnection patterns that are more complex than multidimensional meshes are possible. Segmented buses may appear in a hierarchical system design, providing chip-to-chip, module-to-module, board-to-board or

node-to-node communication. We believe that segmented buses are more promising in hardware implementation than most other reconfigurable buses, such as the ones used in reconfigurable meshes, because of their relatively simpler control schemes.

We would like to mention that an optical bus system, called linear array with a reconfigurable pipelined bus system (LARPBS), has been introduced recently. The paper by Pan and Li [57] gives an excellent survey on this subject. By allowing packets to be transmitted in a pipelined fashion using the time-division multiplexing (TDM) method, an LARPBS can simulate a complete point-to-point network (complete graph) with a constant factor slowdown. In addition, it can perform several operations with performance better than a PRAM of the same size. Since a segmented bus does not assume pipelined the TDM transmission mode, it can be implemented in either electronic or optical domain. Though less powerful than an LARPBS, a segmented bus is easier to be implemented.

Chapter 6

Hypernetworks

Designing high bandwidth, low latency and scalable interconnection networks is a great challenge in the construction of high-performance parallel computer systems. Traditionally, interconnection networks are characterized by graphs. Network topologies under graph models have been extensively investigated. Many network structures have been proposed, and some have been implemented. Observed the improving electrical bus and switching technologies and maturing optical interconnection technologies, Zheng pointed out that conventional graph structure is no longer adequate for the design and analysis of the new generation interconnection structures and proposed a new class of interconnection networks, the hypernetworks [84].

The class of hypernetworks is a generalization of point-to-point networks, and it contains point-to-point networks as a subclass. In a hypernetwork, the physical communication medium (a hyperlink) is accessible to multiple (usually, more than two) processors. The relaxation on the number of processors that can be connected by a link provides more design alternatives so that greater flexibilities in trade-offs of contradicting design goals are possible. The underlying graph theoretic tool for investigating hypernetworks is hypergraph theory [8]. Hypergraphs are used to model hypernetworks. Hypernetwork designs have been formulated as an optimization problem of constructing constrained hypergraphs. Interested readers may refer to [84, 85, 87, 88] for more justifications, design issues and implementation aspects of hypernetworks.

Existing results in hypergraph theory and combinatorial block design theory, which is closely related to hypergraph theory, can be used to design hypernetworks. For example, in [87], Zheng introduced several low diameter hypernetworks, referred as to $GMSH$, based on the concept of Steiner Triple System, as shown in Figure 6.1. The comparison between the $GMSH$ and point-to-point hypercube is shown in Table 6.1. From the table, one can see clearly the above claimed advantages of hypernetworks over point-to-point networks. In [88], Zheng and Wu proposed a scheme for constructing a new hypernetwork from an existing one using the concept of dual graph in hypergraph theory. They showed that the dual H^* of any given hypergraph H is a hypergraph that has some properties related to the properties of H so that one can investigate the properties of H^* based on the properties of H. Since the structure of H and its dual H^* can be drastically different, finding hypergraph duals can be considered as a general approach to the design of new hypernetworks. They investigated the structure of the dual K_n^* of an n-vertex complete point-to-point network K_n.

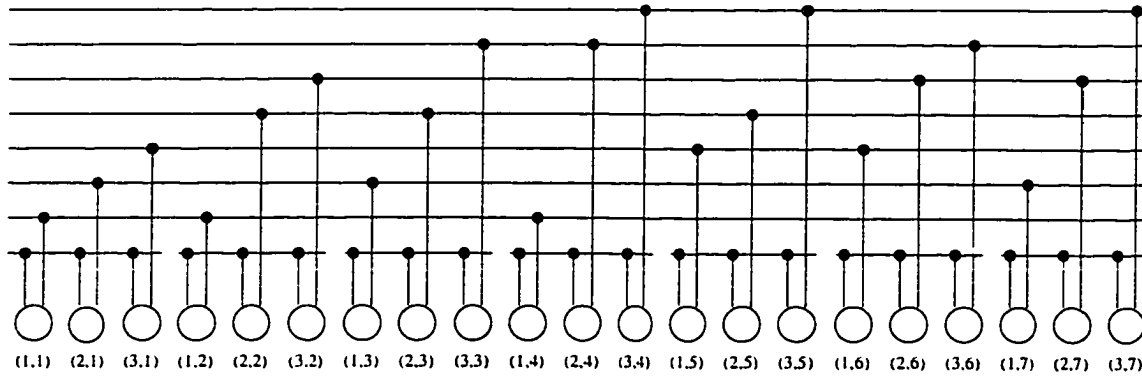


Figure 6.1: Bus implementation of GMSH(3,2).

Hypercube is a popular point-to-point network which has many desirable features such as small diameter, symmetry, and supporting a large class of efficient parallel algorithms. In this paper, we propose a class of hypernetworks, the Q_n^* (read as Q_n star) hypernetworks. The Q_n^* hypernetwork is the dual of the n-dimensional hypercube Q_n. We discuss the topological and fault tolerance aspects of Q_n^*, and present a set of parallel data

Table 6.1: Comparison between $GMSH(3,d)$ and point-to-point hypercubes

	n,number of vertices	m,number of (hyper)links	Δ degree	δ diameter	$\frac{m}{n}$ ratio
$GMSH(3,1)$	3	1	1	1	0.3
Q_2	4	4	2	2	1
$GMSH(3,2)$	21	14	2	3	0.6
Q_4	16	32	4	4	2
$GMSH(3,3)$	903	903	3	7	1
Q_{10}	1024	5120	10	10	5
$GMSH(3,4)$	1,631,721	2,175,628	4	15	1.3
Q_{21}	2,097,152	22,020,096	21	21	15.5
$GMSH(3,5)$	5.3×10^{12}	8.9×10^{12}	5	31	1.6
Q_{42}	4.4×10^{12}	92.4×10^{12}	2	2	1

communication algorithms for Q_n^*. Our results indicate that the Q_n^* hypernetwork is a useful and promising interconnection network for high-performance parallel and distributed computing systems.

6.1 Background

Hypergraphs are used as underlying graph models of hypernetworks. A *hypergraph* [8] $H = (V, E)$ consists of a set $V = \{v_1, v_2, \cdots, v_n\}$ of vertices, and a set $E = \{e_1, e_2, \cdots, e_m\}$ of hyperedges such that each e_i is a non-empty subset of V and $\{v | v \in e_i, 1 \leq i \leq m\} = V$. An edge e contains a vertex v if $v \in e$. If $e_i \subseteq e_j$ implies that $i = j$, then H is a *simple hypergraph*. In this article, we only consider simple hypergraphs. When the cardinality of an edge e, denoted as $|e|$, is 1, it corresponds to a self-loop edge. If all the edges have cardinality 2, then H is a graph that corresponds to a point-to-point network. A hypergraph of n vertices and m hyperedges can also be defined by its $n \times m$ incidence matrix A with columns representing edges and rows representing vertices such that $a_{i,j} = 0$ if $v_i \notin e_j$, $a_{i,j} = 1$ if $v_i \in e_j$.

For a subset E' of E, we call the hypergraph $H'(V', E')$ such that $V' = \{v | v \in e, e \in E'\}$ the *partial hypergraph of H generated by the set E'*. For a subset U of V, we call the hypergraph $H''(V'', E'')$ such that $E'' = \{e_i \cap U | e_i \cap U \neq \phi, 1 \leq i \leq m\}$

and $V'' = \{v | v \in e, e \in E''\}$ the *sub-hypergraph induced by the set U.* Note that such an induced sub-hypergraph may or may not be a simple hypergraph.

The degree $d_H(v_i)$ of v_i in H is the number of edges in V that contain v_i. A hypergraph in which all the vertices have the same degree is said to be *regular.* The *degree of hypergraph* H, denoted by $\Delta(H)$, is defined as $\Delta(H) = \max_{v_i \in V} d_H(v_i)$. A regular hypergraph of degree k is called *k-regular hypergraph.* The *rank* $r(H)$ and *antirank* $s(H)$ of a hypergraph H is defined as $r(H) = \max_{1 \leq j \leq m} |e_j|$ and $s(H) = \min_{1 \leq j \leq m} |e_j|$, respectively. We say that H is a *uniform hypergraph* if $r(H) = s(H)$. Hypercube is a good example for being regular and uniform. A uniform hypergraph of rank k is called *k-uniform hypergraph.* A hypergraph is *vertex* (resp. *hyperedge*) *symmetric* if for any two vertices (resp. hyperedges) v_i and v_j (resp. e_i and e_j) there is an automorphism of the hypergraph that maps v_i to v_j (resp. e_i to e_j).

In a hypergraph H, a path of length q is defined as a sequence $(v_{i_1}, e_{j_1}, v_{i_2}, e_{j_2}, \cdots, e_{j_q}, v_{i_{q+1}})$ such that (1) $v_{i_1}, v_{i_2}, \cdots, v_{i_{q+1}}$ are all distinct vertices of H; (2) $e_{j_1}, e_{j_2}, \cdots, e_{j_q}$ are all distinct edges of H; and (3) $v_{i_k}, v_{i_{k+1}} \in e_{j_k}$ for $k = 1, 2, \cdots, q$. A path from v_i to v_j, $i \neq j$, is a path in H with its end vertices being v_i and v_j. A hypergraph is *connected* if there is a path connecting any two vertices. We only consider connected hypergraphs. A hypergraph is *linear* if $|e_i \cap e_j| \leq 1$ for $i \neq j$, i.e., two distinct buses share at most one common vertex. For any two distinct vertices v_i and v_j in a hypergraph H, the distance between them, denoted by $dis(v_i, v_j)$, is the length of the shortest path connecting them in H. Note that $dis(v_i, v_i) = 0$. The *diameter* of a hypergraph $H = (V, E)$, denoted by $\delta(H)$, is defined by $\delta(H) = \max_{v_i, v_j \in V} dis(v_i, v_j)$. More concepts in hypergraph theory can be found in [8].

A *hypernetwork* M is a network whose underlying structure is a hypergraph H, in which each vertex v_i corresponds to a unique processor P_i of M, and each hyperedge e_j corresponds to a *connector* that connects the processors represented by the vertices in e_j. A connector is loosely defined as an electronic or a photonic component through which

messages are transmitted between connected processors, not necessarily simultaneously. We call a connector a *hyperlink*.

Unlike a point-to-point network, in which a link is dedicated to a pair of processors, a hyperlink in a hypernetwork is shared by a set of processors. A hyperlink can be implemented by a bus or a crossbar switch. Current optical technologies allow a hyperlink to be implemented by optical waveguides in a folded-bus form operating in pipelined fashion using time-division multiplexing. Free-space optical or optoelectronic switching devices such as bulk lens, microlens array, spatial light modulator (SLM), and smart pixel arrays can also be used to implement hyperlinks. A star coupler, which uses wavelength-division multiplexing, can be considered either as a generalized bus structure or as a photonic switch, is another implementation of a hyperlink. Similarly, an ATM switch, which uses TDM, is also a hyperlink. In the rest of this chapter, the following pairs of terms are used interchangeably: (hyper)edges and (hyper)links, vertices and processors, point-to-point networks and graphs, and hypernetworks and hypergraphs.

6.2 Hypernetwork Design Issues

The problem of designing efficient interconnection networks can be considered as a constrained optimization problem. For example, the goal of designing point-to-point networks is to find well-structured graphs (whose ranks are fixed, as a constant 2) with small degrees and diameters. In hypernetwork design, the relaxation on the number of processors that can be connected by a hyperlink (i.e. the rank of the hyperlink) provides more design alternatives so that greater flexibilities in trade-offs of contradicting design goals are possible.

6.3 Dual Hypernetworks and Q_n^* Hypernetworks

6.3.1 Dual Graph

The *dual* of a hypergraph $H = (V, E)$ with vertex set $V = \{v_1, v_2, \cdots, v_n\}$ and hyperedge set $E = \{e_1, e_2, \cdots, e_m\}$ is a hypergraph $H^* = (V^*, E^*)$ with vertex set $V^* =$

$\{v_1^*, v_2^*, \cdots, v_m^*\}$ and hyperedge set $E^* = \{e_1^*, e_2^*, \cdots, e_n^*\}$ such that v_j^* corresponds to e_j with hyperedges $e_i^* = \{v_j^* | v_i \in e_j \text{ and } e_j \in E\}$. In other words, H^* is obtained from H by interchanging of vertices and hyperedges in H. The incidence matrix of H^* is the transpose of the incidence matrix of H. Thus, $(H^*)^* = H$. The following relations between a hypergraph and its dual are apparent [88].

Proposition 1 *H is r-uniform if and only if H^* is r-regular.*

Proposition 2 *The dual of a linear hypergraph is also linear.*

Proposition 3 *A hypergraph H is vertex symmetric if and only if H^* is hyperedge symmetric .*

Proposition 4 *The dual of a sub-hypergraph of H is a partial hypergraph of the dual hypergraph H^*.*

Since $(H^*)^* = H$, all the above propositions still hold after interchanging H with H^*.

Proposition 5 $\delta(H) - 1 \leq \delta(H^*) \leq \delta(H) + 1$.

Propositions 1 - 5 show that some properties of the dual hypergraph H^* of a given hypergraph H can be derived from properties of H. For example, if H is a ring, then H^* is isomorphic to H. However, in general, the structures of H and its dual H^* can be drastically different. Finding hypergraph duals can be considered as a general approach to the design of new hypernetworks.

6.3.2 The Hypernetwork Q_n^*

We consider using the dual Q_n^* of the hypercube Q_n as a hypernetwork. An n-dimensional hypercube Q_n consists of 2^n vertices, each being labeled by a unique n-bit binary number. Two vertices are connected by an edge if and only if their binary labels are distinct in one bit position. Properly labeling the vertices and hyperedges in Q_n^* can greatly simplify its use as a communication network. Vertex labels are used as processor addresses. Similarly,

hyperedge labels are used as the unique names of hyperlinks. There are many ways to label the vertices and hyperedges of Q_n^*.

Let I_n be the set of non-negative integers that can be represented by n-bit binary numbers. For $l, u \in I_n$, we use $d(l,u)$ to denote the number of different bits in the binary representations of l and u, i.e., $d(l,u)$ is the Hamming distance between the binary representations of l and u.

Definition 1 Let $N_n = n2^{n-1}$ for $n \geq 2$. The Q_n^* hypernetwork is a hypergraph with vertex set $\{\langle l,u\rangle | l,u \in I_n,\ l < u, \text{ and } d(l,u) = 1\}$ of N_n vertices and 2^n hyperlinks, $e_0, e_1, \cdots, e_{2^n-1}$. Each vertex $\langle l,u\rangle$ is connected to exactly two hyperedges e_l and e_u.

Example 1 *The incidence matrix A of Q_3^* is*

$$A = \begin{array}{c} \\ \langle 0,1\rangle \\ \langle 0,2\rangle \\ \langle 1,3\rangle \\ \langle 2,3\rangle \\ \langle 0,4\rangle \\ \langle 1,5\rangle \\ \langle 2,6\rangle \\ \langle 3,7\rangle \\ \langle 4,5\rangle \\ \langle 4,6\rangle \\ \langle 5,7\rangle \\ \langle 6,7\rangle \end{array} \begin{array}{c} \begin{array}{cccccccc} e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}.$$

The transpose of A is

$$A^T = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} e_{0,1} & e_{0,2} & e_{1,3} & e_{2,3} & e_{0,4} & e_{1,5} & e_{2,6} & e_{3,7} & e_{4,5} & e_{4,6} & e_{5,7} & e_{6,7} \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Clearly, A^T is the incidence matrix of the hypercube Q_3. Figure 6.2 shows the bus implementation of the Q_3^* hypernetwork, whose incidence matrix A is given above. Its corresponding hypercube, whose incidence matrix is A^T, is shown in Figure 6.3, where each edge is labeled by its two end vertices.

□

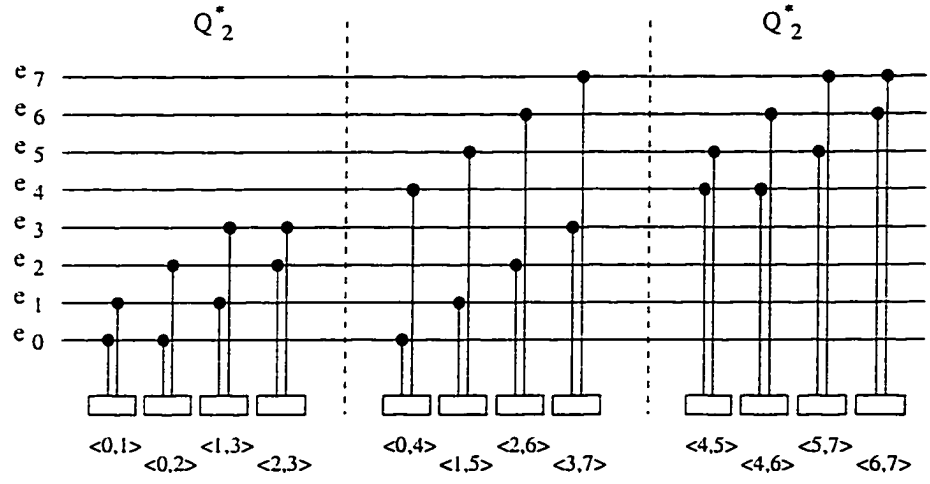


Figure 6.2: Bus implementation of Q_3^*.

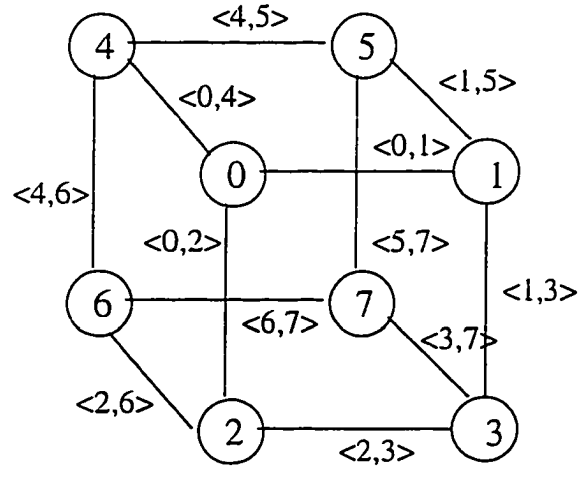


Figure 6.3: Hypercube Q_3 corresponding to Q_3^*.

The Q_n^* hypernetwork can also be defined in a recursive way. One can easily observe that Q_3^* can be constructed with two copies of Q_2^* (see Figure 6.2), and Q_4^* can be constructed with two copies of Q_3^* (see Figure 6.4). For brevity, we omit the recursive definition of Q_n^*.

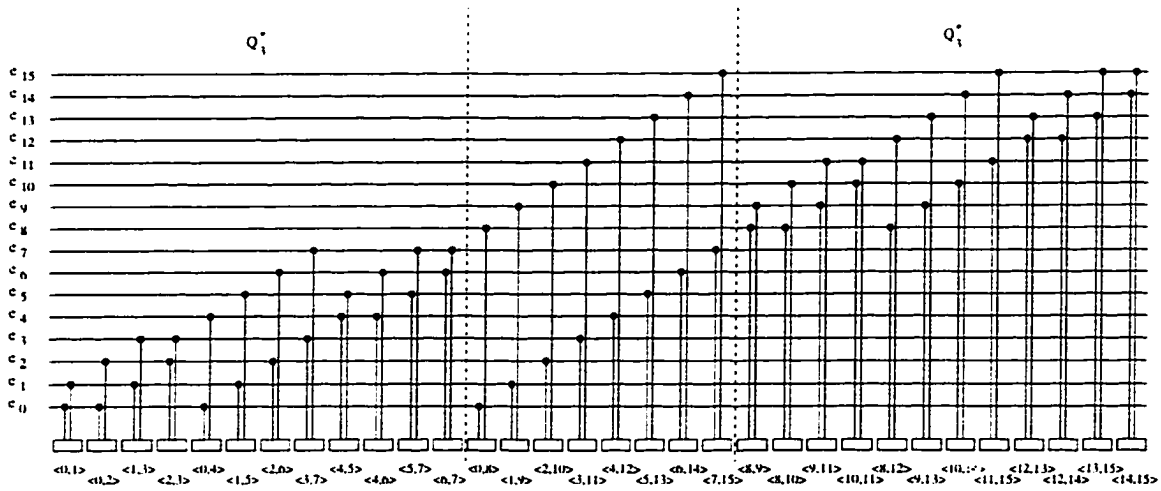


Figure 6.4: Bus implementation of Q_4^*.

6.3.3 The Properties of Q_n^*

Based on the properties of hypercube Q_n and Propositions 1 to 4, we have the following fact:

Fact 1: Q_n^* is 2-regular, n-uniform, linear, and vertex and hyperedge symmetric.

Let $dis(\langle l,u\rangle, \langle l',u'\rangle)$ be the distance between vertices $\langle l,u\rangle$ and $\langle l',u'\rangle$ in Q_n^*. We want to derive the diameter of Q_n^*.

Lemma 3 *For any two vertices* $\langle l,u\rangle$ *and* $\langle l',u'\rangle$ *in* Q_n^*, $dis(\langle l,u\rangle, \langle l',u'\rangle) = \min\{\ d(l,l'),\ d(l,u'),\ d(u,l'), d(u,u')\} + 1$.

Proof. We view $\langle l,u\rangle$ and $\langle l',u'\rangle$ as two edges in Q_n. The minimal path connecting these two edges is one from one end node of $\langle l,u\rangle$ to one end node of $\langle l',u'\rangle$. Therefore, the distance is the length of a minimal path between two end nodes plus one. □

Lemma 4 *For any two vertices* $\langle l,u\rangle$ *and* $\langle l',u'\rangle$ *in* Q_n^*, $dis(\langle l,u\rangle, \langle l',u'\rangle) \leq n$.

Proof. For two hyperedges $\langle l, u\rangle$ and $\langle l', u'\rangle$ in Q_n^*, if $d(l, l') = n$ then $d(l, u') < n$. Therefore, $d(\langle l, u\rangle, \langle l', u'\rangle) = \min\{d(l, l'), d(l, u'), d(u, l'), d(u, u')\} + 1 \leq \min\{d(l, l'), d(l, u')\} + 1 \leq (n-1) + 1 \leq n$.

Theorem 7 *The diameter of Q_n^* is n.*

Proof. From Lemma 4, we know that the diameter is less than or equal to n. All we need to do is to prove that there are two vertices $\langle l, u\rangle$ and $\langle l', u'\rangle$ such that $dis(\langle l, u\rangle, \langle l', u'\rangle) = n - 1$. Actually, it is easy to see that vertices $\langle 000...0, 100...0\rangle$ and $\langle 011...1, 111...1\rangle$ meet the above condition. □

In Q_n^*, the number of processors is $n/2$ times the number of the hyperlinks. Each processor is attached to exactly two hyperlinks, and this simplifies the processor interface circuit design. Each hyperlink connects n processors. Suppose that a 10×10 crossbar switch is implementable and cost effective, then Q_{10}^* of 5,120 processors can be implemented using 1,024 such switches as hyperlinks.

Consider the fault tolerance aspect of the Q_n^* hypernetwork. We say that a hypernetwork H is *x-processor fault-tolerant* (resp. *y-hyperlink fault-tolerant*) if it remains connected when no more than any x processors (resp. y-hyperlinks) are removed. We have the following claim.

Theorem 8 *Q_n^* is $(2n-3)$-processor fault-tolerant and 1-hyperlink fault-tolerant.*

Proof. Consider the hypercube Q_n. Take an edge e and delete all edges that share a vertex with e, then the graph becomes disconnected. This implies that it is possible to disconnect the Q_n^* hypernetwork by removing $2(n-1)$ processors. However, removing any less than $2(n-1)$ processors from Q_n will not make the remaining part of Q_n disconnected. Hence, Q_n^* is $(2n-3)$-processor fault-tolerant. Since Q_n^* is 2-regular, removing any two hyperlinks that share a vertex v will disconnect processor v. □

6.4 Data Communication Algorithms for Q_n^*

In this section, we use the vertex and hyperedge labels to design data communication algorithms for the Q_n^* hypernetwork. For simplicity, we assume bus implementation of hyperlinks. In the electronic domain, the bus load, i.e., the number of processors that can be connected by a bus, is limited. Using optical fibers to implement a bus, the bus load can be increased significantly. Since a bus is shared by all its connected processors, the performance of a bus depends on the way it is accessed by processors. For example, one way for processors to share a bus is to use time-division multiplexing (TDM), which allocates time slots to processors so that they can only access the bus during their slots. Another way is to let processors compete for bus tenure, and use an arbiter to grant the bus tenure in an on-line fashion.

We assume a synchronous mode communication. Bus allocations, although operated dynamically, are predetermined by an off-line scheduling algorithm. This bus operational mode has been used in [16] for analyzing a multiple-bus interprocessor connection structure. Assume that all messages are of the same length. The communication performance is measured in terms of parallel message steps. We adopt these assumptions for two reasons. First, under these assumptions, it is easier to assess the capability and limitation of the proposed hypernetwork structure. Secondly, the performance results obtained can be easily used to measure other bus communication method by either adding additional overheads, which may be incurred in TDM transmission and asynchronous bus allocation, or deducting transmission latency saving due to pipelining effect of a pipelined optical bus.

We discuss four types of communication operations: one-to-one communications, one-to-many communications, many-to-one communications and many-to-many communications. For each type, we present an algorithm for a representative communication operation. These communication algorithms constitute a useful set of tools for designing parallel algorithms on the Q_n^* hypernetwork.

6.4.1 One-to-One Communication

We consider shortest path routing between two processors, where $\langle l, u\rangle$ and $\langle l', u'\rangle$ represent the source processor and the destination processor, respectively. The idea of the shortest path routing is as follows. If the two processors share one hyperlink, the message is transmitted through that hyperlink. Otherwise, the transmission is done from hyperlink e_a toward hyperlink e_b such that $a \in \{l, u\}$, $b \in \{l', u'\}$, and $d(a, b)$ is the minimal. The source processor sends the message to the processor $\langle a, c\rangle$ through hyperlink e_a such that $d(c, b) = d(a, b) - 1$. This process is recursive; that is, the processor $\langle a, c\rangle$ will then relay the message toward the processor $\langle l', u'\rangle$. The following is the shortest path routing algorithm.

procedure $ROUTE(\langle l, u\rangle, \langle l', u'\rangle)$
begin
  Let $a \in \{l, u\}$, $b \in \{l', u'\}$ such that $d(a, b) = dis(\langle l, u\rangle, \langle l', u'\rangle) - 1$;
  if $a = b$ **then**
    Processor $\langle l, u\rangle$ sends the message to $\langle l', u'\rangle$ using e_a
  else begin
    Select c such that $d(c, b) = d(a, b) - 1$;
    Processor $\langle l, u\rangle$ sends the message to $\langle a, c\rangle$ using e_a;
    $ROUTE(\langle a, c\rangle, \langle l', u'\rangle)$
    end
end

Theorem 9 *For any given pair of processors $\langle l, u\rangle$ and $\langle l', u'\rangle$ in the Q_n^* hypernetwork, algorithm ROUTE routes a message from $\langle l, u\rangle$ to $\langle l', u'\rangle$ along a minimal path in $dis(\langle l,u\rangle, \langle l', u'\rangle)$ message steps.*

Proof. The theorem directly follows from Lemmas 3 and 4. □

6.4.2 One-to-Many Communication

We consider broadcasting a message from any processor $\langle l, u \rangle$ to all other processors in Q_n^*. Given $\langle l, u \rangle$, procedure *TRANSFORM* is used to transform $\langle l, u \rangle$ to $\langle 0, 1 \rangle$ and all the other $\langle a, b \rangle$ in Q_n^* to $\langle a', b' \rangle$ in $O(n)$ time.

```
procedure TRANSFORM(⟨l, u⟩)
begin
   for all ⟨a, b⟩ do in parallel
      a' = a;
      b'= b;
      for (k=0; k<n; k++)
       if l(k) ≠ u(k) then break;
      Shift bit k of l,u,a',b' to 0th bit
      for (k=0; k<n; k++)
       if l(k) ≠ 0 then
          complement l(k), u(k), a'(k), b'(k)
end
```

A more complicated program can do the same transform in constant time. We call this program as TRANSFORM1 as shown below.

```
procedure TRANSFORM1(⟨l, u⟩)
begin
   for all ⟨a, b⟩ do in parallel
```

if $a = 0$ and $b = 1$ **then** $\langle a', b' \rangle = \langle l, u \rangle$;

else if $a = l$ and $b = u$ **then** $\langle a', b' \rangle = \langle 0, 1 \rangle$;

else $\langle a', b' \rangle = \langle TRANSLINK(a, l, u),$

$TRANSLINK(b, l, u) \rangle$

end

procedure *TRANSLINK*(e,l,u)

begin

l *band* u → x1

(l *bxor* u) | 1 → x2

e *band* x2 → x3

e *band* (*bcom* x2) → x4

if x3=1 or (x3 mod 2) =0 **then**

x3 *bxor* F → x5

x1 *bxor* x4 → x6

x5 *bor* x6 → e'

return e'

end

By the symmetry of the Q_n^* hypernetwork, we know that the new identities $\langle a', b' \rangle$ assigned to processors of Q_n^* satisfy the connectivities of Q_n^*. We only need to describe an algorithm that broadcasts a message from $\langle 0, 1 \rangle$ to all processors in Q_n^*.

We use 0^k to represent k consecutive 0's, e.g., $0^3 = 000$. Let $Q_k = 0^{n-k-1}0*^k$ and $Q_k' = 0^{n-k-1}1*^k$ denote the k-dimensional subcube of Q_n induced by all vertices whose left $n-k$ bits are $0^{n-k-1}0$ and $0^{n-k-1}1$, respectively. Here, an $*$ in a bit position stands for "don't care", and $*^k$ represents k consecutive $*$'s. We use $Q_{k+1} = Q_k + Q_k'$ to denote

the $(k+1)$-dimensional subcube of Q_n induced by vertices in Q_k and Q'_k. We explain the broadcasting algorithm in Q_n^* using an n-dimensional hypercube (Q_n) by interchanging the role of vertices and edges. The idea behind our broadcasting algorithm is as follows. Assuming that initially the edge connecting vertices $0^{n-1}0$ and $0^{n-1}1$ in Q_n is colored, and all other edges in Q_n are not colored. We want an edge traversing algorithm A that systematically traverses all edges in Q_n in n steps. In the k-th step, algorithm A selects a subset E_k of edges in Q_n that satisfies the following conditions: (1) all edges in E_k are not previously traversed, (2) each edge in E_k has at least one end vertex that is an end vertex of a previously colored edge, and (3) for each edge e in in E_k assign a direction it is traversed: let u and v be the two end vertices of e, and suppose that u is an end vertex of a previously traversed edge, then traverse e from u to v. In the following, we provide a selection procedure for E_k so that all edges of Q_n are guaranteed to be traversed in n parallel steps. Obviously, such an algorithm A corresponds to a broadcasting algorithm A^* for Q_n^*.

Now, let us describe our algorithm A. Starting from Q_1 (which corresponds to the edge connecting vertices $0^{n-1}0$ and $0^{n-1}1$ in Q_n, and the source vertex $\langle 0^{n-1}0, 0^{n-1}1\rangle$ in Q_n^*), increase the dimension of the cube by one in each step. In the first step, we consider $Q_2 = Q_1 + Q'_1$, the two edges connecting vertices in Q_1 and Q'_1 are colored. In the second step, the edge connecting $0^{n-2}10$ and $0^{n-2}11$ is traversed in the direction from $0^{n-2}10$ and $0^{n-2}11$ and the edges connecting Q_2 and Q'_2 are traversed in the direction from Q_2 and Q'_2. Assume that after k, $1 \leq k \leq n-2$, steps, all the edges in $Q_k = 0^{n-k-1}0*^k$ and the ones connecting Q_k and $Q'_k = 0^{n-k-1}1*^k$ have been traversed, but all the edges in Q'_k have not been traversed. In step $k+1$, we traverse all the edges in Q'_k and the edges connecting Q_{k+1} and Q'_{k+1}, where $Q_{k+1} = 0^{n-k-2}0*^{k+1}$ and $Q'_{k+1} = 0^{n-k-2}1*^{k+1}$, in the direction from Q_{k+1} to Q'_{k+1}. To traverse all the edges in Q'_k, we randomly pick two connected vertices v and u, if $v < u$ then the link (v,u) is traversed from v to u. In step n, we only need to traverse all the edges in Q'_{n-1} in the direction from $Q'_{n-1,0}$ to $Q'_{n-1,1}$.

Let $b_{n-1}b_{n-2}\cdots b_0$ be the binary representation of b. We use $b^{(i)}$ to represent the binary number (and its corresponding decimal value) obtained by complementing the ith bit, b_i, of the binary representation of b. Translating the above hypercube edge traverse algorithm into an algorithm for traversing vertices in Q_n^*, we obtain the following algorithm.

procedure *BROADCAST*$(\langle 0,1\rangle)$
begin
  for $k = 1$ **to** $n-1$ **do**
    for all $\langle a,b\rangle$ where $a \in 0^{n-k}0*^{k-1}$ and $b \in 0^{n-k}1*^{k-1}$ **do in parallel**
      Processor $\langle a,b\rangle$ sends the message to $\langle a,a^{(k+1)}\rangle$ using e_a;
      Processor $\langle a,b\rangle$ sends the message to $\langle b,b^{(k+1)}\rangle$ using e_b;
      Processor $\langle a,b\rangle$ sends the message to $\langle b,b^{(i)}\rangle$ using e_b,
       if $b < b^{(i)}$ for $i \in \{0,1,...,k-1\}$
    endfor
  endfor
  for all $\langle a,b\rangle$ **do in parallel**
    if $b < b^{(i)}$ for $i \in \{0,1,...,n-1\}$
    then Processor $\langle a,b\rangle$ sends the message to $\langle b,b^{(i)}\rangle$ using e_b
  endfor
end

In Figure 6.5, we show the broadcasting tree for Q_4^*, whose bus implementation is shown in Figure 6.4. In Figure 6.5, a circle labeled by a pair of integers a and b represents a processor $\langle a,b\rangle$. A directed edge labeled by an integer c from $\langle a,b\rangle$ to $\langle a',b'\rangle$ indicates that the message is transmitted from $\langle a,b\rangle$ to $\langle a',b'\rangle$ using hyperlink e_c.

Theorem 10 *Assuming bus hyperlinks of Q_n^*, algorithm BROADCAST broadcasts a message from any processor to all other processors in n parallel message steps.*

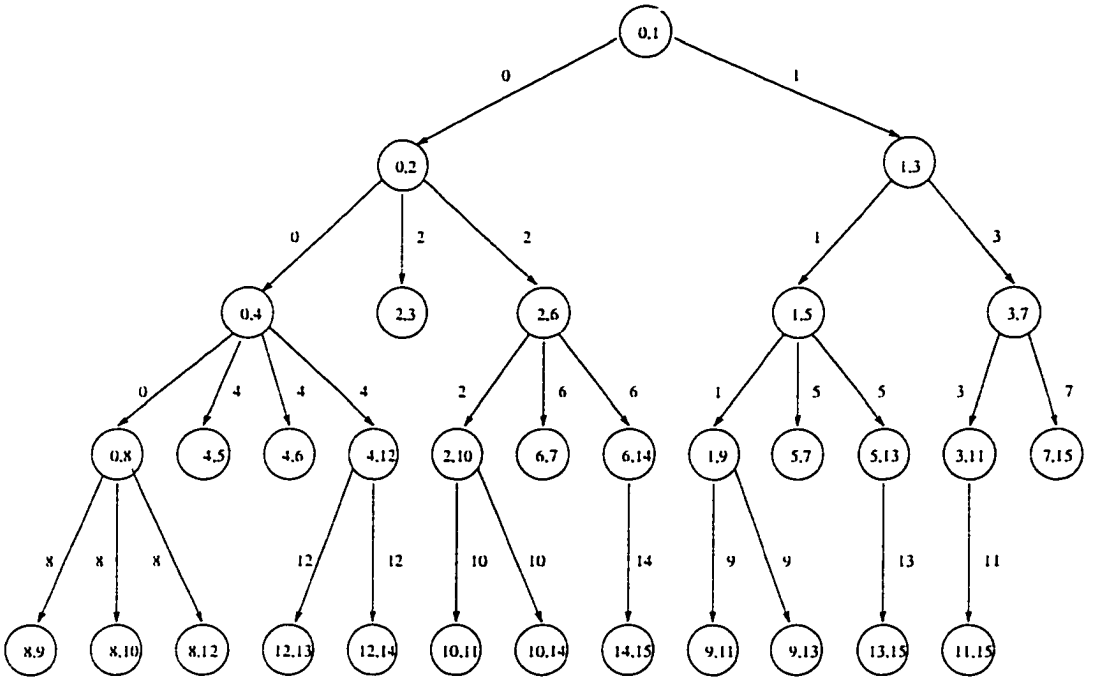


Figure 6.5: Data communication pattern for broadcasting from $\langle 0, 1\rangle$ in Q_4^*.

Proof. The theorem follows directly from a simple induction based on the discussion preceding *BROADCAST*. □

A genral case broadcast can also be done in the same complexity. This algorithm, called GBROADCAST, is shown as follows.

procedure *GBROADCAST*($\langle l, u\rangle$)

begin

$\langle l, u\rangle$ broadcasts on l and u

for $k = \frac{u-l}{2}$ **to** $k \geq 1$ **step** $k = \frac{k}{2}$**do**

for all $\langle a, b\rangle$ such as $b - a = k$ and $l \leq a$, $b \leq u$ **do in parallel**

Processor $\langle a, b\rangle$ broadcasts on a and b

endfor

endfor

for $k = 2 * (u - l)$ **to** $k \leq 2^{n-1}$ **step** $k = k * 2$ **do**

for all $\langle a, b\rangle$ such as $b - a = k$ **do in parallel**

Processor $\langle a, b\rangle$ broadcasts on a and b

endfor

endfor

end

6.4.3 Many-to-One Communication

A reduction (or census, or fan-in) function is defined as a commutative and associative operation on a set of values, such as finding maximum, addition, logic or, etc. It can be carried out using a many-to-one communication operation. We only consider the case that the specified reduction operation is addition. The same algorithm can be slightly modified to perform other reduction operations.

We present an algorithm that can be used to perform a summation on a set of N_n values stores in the A registers of processors, one per processor, and putting the final result in processor $\langle l, u\rangle$. That is, the algorithm computes $\sum_{\langle a,b\rangle \in Q_n^*} A_{\langle a,b\rangle}$, and putting the final result in $A_{\langle l,u\rangle}$ of processor $\langle l, u\rangle$. We assume that each processor $\langle a, b\rangle$ has a working register $B_{\langle a,b\rangle}$. Given any processor $\langle l, u\rangle$, procedure *TRANSFORM* discussed in the previous section can be used to transform $\langle l, u\rangle$ to $\langle 0^{n-1}0, 0^{n-1}1\rangle$ (or $\langle 0, 1\rangle$) and all other $\langle a, b\rangle$ in Q_n^* to $\langle a', b'\rangle$. Then, we only need to consider the summation algorithm that stores the final result in processor $\langle 0, 1\rangle$.

Summation is done in two phases. In the first phase, a set of 2^{n-1} partial sums are obtained and stores in processors $\langle z, 2^{n-1} + z\rangle$, $0 \leq z \leq 2^{n-1} - 1$. The second phase computes the sum of these partial sums and stored the final result in $\langle 0, 1\rangle$.

procedure *REDUCTION*($\langle 0, 1\rangle$)

begin

/ phase 1 */*

for $i = 1$ **to** $n-1$ **do**

for all $\langle a, b\rangle$ **do in parallel**

if $a_i a_{i-1} = 00$ and $b_i b_{i-1} = 01$ **then**

Processor $\langle a, b\rangle$ sends $A_{\langle a,b\rangle}$ from $\langle a, b\rangle$ to $\langle a, (b^{(i)})^{(i-1)}\rangle$ using e_a;

if $a_i a_{i-1} = 10$ and $b_i b_{i-1} = 11$ **then**

Processor $\langle a, b\rangle$ sends $A_{\langle a,b\rangle}$ from $\langle a, b\rangle$ to $\langle (a^{(i)})^{(i-1)}, b\rangle$ using e_b;

if $\langle a, b\rangle$ received a value **then**

store this value in $B_{\langle a,b\rangle}$ and perform $A_{\langle a,b\rangle} := A_{\langle a,b\rangle} + B_{\langle a,b\rangle}$

endfor

endfor

/* phase 2 */

for $i = n-1$ **down to** 1 **do**

for all $\langle a, b\rangle$ **do in parallel**

if $a = 0^{n-i-1}00*^{i-1}$ and $b = 0^{n-i-1}10*^{i-1}$ **then**

Processor $\langle a, b\rangle$ sends $A_{\langle a,b\rangle}$ from $\langle a, b\rangle$ to $\langle a, (b^{(i)})^{(i-1)}\rangle$ using e_a;

if $a = 0^{n-i-1}01*^{i-1}$ and $b = 0^{n-i-1}11*^{i-1}$ **then**

Processor $\langle a, b\rangle$ sends $A_{\langle a,b\rangle}$ from $\langle a, b\rangle$ to $\langle (b^{(i)})^{(i-1)}, a\rangle$ using e_a;

if $\langle a, b\rangle$ received two values **then**

store one value in $A_{\langle a,b\rangle}$ and the other in $B_{\langle a,b\rangle}$, and perform

$A_{\langle a,b\rangle} := A_{\langle a,b\rangle} + B_{\langle a,b\rangle}$

endfor

endfor

end

Theorem 11 *Assuming bus hyperlinks of* Q_n^**, algorithm REDUCTION carries out a reduction operation in* $2(n-1)$ *parallel message transmission steps.*

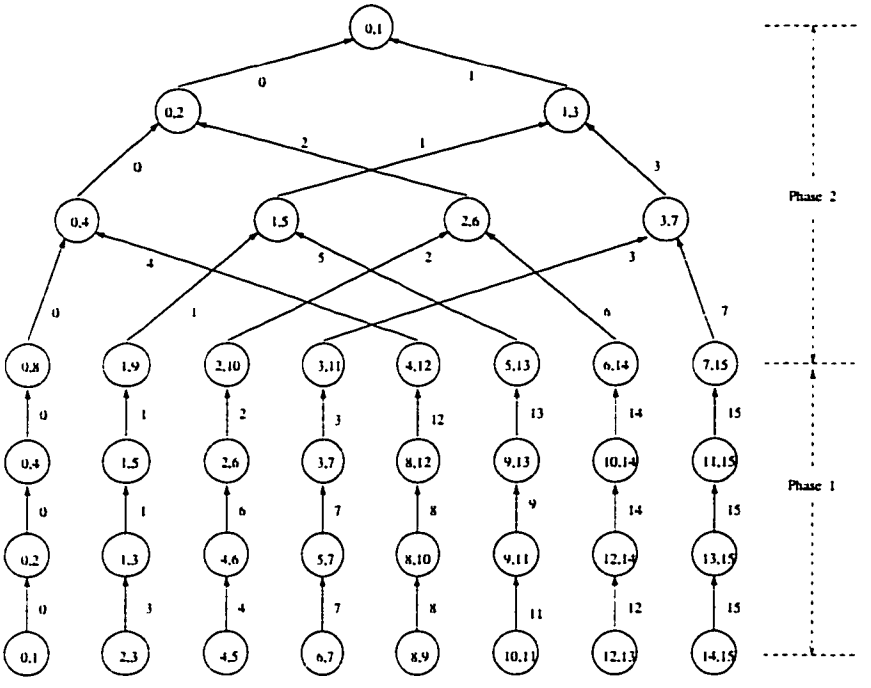


Figure 6.6: Data communication pattern for reduction in Q_4^*.

Proof. First, we claim that at the end of the first phase the sum of the partial sums stored in processors $\langle z, 2^{n-1}+z\rangle$, $0 \le z \le 2^{n-1}-1$, is the final sum. It is easy to verify that the claim is true for $n = 2$ and $n = 3$. Suppose that the claim is true for $n = k$, and consider the case $n = k+1$. By the algorithm, any processor $\langle a, b\rangle$ such that $a_k a_{k-1} = 00$ and $b_k b_{k-1} = 01$ has not sent and receive any value before the k-th step (iteration). Furthermore, by the hypothesis, the sum of the partial sums stored in processors $\langle z, 2^{k-1}+z\rangle$, $0 \le z \le 2^{k-1}-1$, is the sum of the values originally stored in sub-hypernetwork Q_k^* induced by all vertices $\langle a, b\rangle$ in Q_{k+1}^* such that $a < 2^k$ and $b < 2^k$. Consider one more step (i.e. the k-th iteration) of the first phase. In this step the partial sum stored in $\langle z, 2^{k-1}+z\rangle$, $0 \le z \le 2^{k-1}-1$, is sent to processor $\langle z, 2^k+z\rangle$, $0 \le z \le 2^{k-1}-1$, using hyperlink e_z. By the symmetry, the sum of the partial sums stored in processors $\langle 2^k+z, 2^k+2^{k-1}+z\rangle$, $0 \le z \le 2^{k-1}-1$, is the sum of the values originally stored in sub-hypernetwork Q_k^* induced by all vertices $\langle a, b\rangle$ in Q_{k+1}^* such that $a \ge 2^k$ and $b \ge 2^k$ and after the k-th step, the partial sum stored in $\langle 2^k+z, 2^k+2^{k-1}+z\rangle$, $0 \le z \le 2^{k-1}-1$, is sent to processor $\langle 2^{k-1}+z, 2^k+2^{k-1}+z\rangle$, $0 \le z \le 2^{k-1}-1$, using hyperlink $e_{2^k 2^{k-1}+z}$.

Therefore, after performing parallel additions in processors $\langle z, 2^k + z\rangle$, $0 \leq z \leq 2^k - 1$, we obtain the claimed 2^k partial sums for Q^*_{k+1}. This completes the induction.

Now, we claim that the second phase computes the sum of the partial sums stored in processors $\langle z, 2^{n-1} + z\rangle$, $0 \leq z \leq 2^{n-1} - 1$, and store the final result in $\langle 0, 1\rangle$ of Q^*_n. This claim is true for $n = 2$ and $n = 3$. Suppose that the claim is true for $n = k$, and consider the case $n = k + 1$. In the first iteration ($i = k$), each processor $\langle z, 2^{k-1} + z\rangle$, $0 \leq z \leq 2^{k-1} - 1$, receives two partial sums, one from $\langle z, 2^k + z\rangle$ via e_z, and the other from $\langle 2^{k-1} + z, 2^k + 2^{k-1} + z\rangle$ via $e_{2^{k-1}+z}$. After additions, 2^{k-1} partial sums are obtained and stored in processors $\langle z, 2^{k-1} + z\rangle$, where $0 \leq z \leq 2^{k-1} - 1$. Then, the induction hypothesis guarantees that after $k - 1$ more steps the final result will be stored in processor $\langle 0, 1\rangle$. This completes the proof of the claim for phase 2, and the proof of the theorem. □

In Figure 6.6, we show the communication pattern used by *REDUCTION* on Q^*_4, whose bus implementation is shown in Figure 6.4. As Figure 6.5, in this figure, a circle labeled by a pair of integers a and b represents a processor $\langle a, b\rangle$. A directed edge labeled by an integer c from $\langle a, b\rangle$ to $\langle a', b'\rangle$ indicates that the message is transmitted from $\langle a, b\rangle$ to $\langle a', b'\rangle$ using hyperlink e_c.

6.4.4 Many-to-Many Communication

We consider a general case, the all-to-all communication. In an all-to-all communication, each processor sends a message to all the other processors. It is also called the *total exchange* operation.

We can obtain a total exchange communication algorithm by modifying algorithm *REDUCTION*. The operator used is set union instead of addition. After applying *REDUCTION*, all messages are collected at processor $\langle 0, 1\rangle$. Then, by applying *BROADCAST*, processor $\langle 0, 1\rangle$ broadcasts the N_n messages to all remaining processors of Q^*_n. By Theorem 5, the second phase alone takes nN_n parallel message steps. Note that the lower bound for the time of a total exchange operation on Q^*_n is $\Omega(N_n)$, and clearly, this algorithm is not efficient.

In what follows, we present an algorithm *TOTAL-EXCHANGE* which takes $O(N_n)$ message steps to perform the total exchange operation on Q_n^*. Algorithm *TOTAL-EXCHANGE* is an all-port algorithm, i.e., the two I/O ports of each processor may participate in a message transmission step. However, each port performs either a send operation or receive operation, but not both. This algorithm can be easily converted to a single-port algorithm with the same communication complexity.

For convenience, we define that a processor $\langle i, j \rangle$ is of dimension k, $0 \leq k \leq n-1$, if $j - i = 2^k$. It is easy to verify the following facts: (i) There are exactly 2^{n-1} processors of dimension k. $0 \leq k \leq n-1$, in Q_n^*; (ii) There is exactly one processor of dimension k, $0 \leq k \leq n-1$, attached to each hyperlink in Q_n^*; and (iii) Any two processors of the same dimension are not attached to the same hyperlink in Q_n^*.

procedure *TOTAL-EXCHANGE*
begin
  for all hyperlinks e_h **do in parallel**
    All processors attached to e_h sends its message to the processor of dimension 0 that is attach to e_h using e_h;
  endfor
  Let the set of messages received by each processor $\langle i, j \rangle$ be denoted by $M_{\langle i,j \rangle}$;
  for $k = 0$ **to** $n-1$ **do**
    for all processors $\langle i, j \rangle$ of dimension k **do in parallel**
      Broadcast $M_{\langle i,j \rangle}$ to all processors attached to hyperlink e_i and e_j
    endfor
    Let the set of messages received by each processor $\langle i, j \rangle$ be denoted by $M_{\langle i,j \rangle}$;
  endfor
end

The correctness of this algorithm can be verified by the induction. It is easy to see the correctness for Q_1^*. Suppose that the algorithm is correct for $n = m$, and consider the case of $n = m + 1$. After m iterations of the **for** loop, the total exchange operations are performed with respect to the subhypergraph of Q_{m+1}^* induced by vertices $\langle i, j \rangle$ such that $i < 2^m$ and $j < 2^m$, and the subhypergraph of Q_{m+1}^* induced by vertices $\langle i', j' \rangle$ such that $i' \geq 2^m$ and $j' \geq 2^m$. In addition, dimension m processors have received all messages in Q_{m+1}^*. In one additional iteration, each processor $\langle a, b \rangle$ of dimension m broadcasts all its received messages to processors attached to hyperlinks e_a and e_b. Then, by (i), (ii) and (iii), the total exchange operation is performed with respect to Q_{m+1}^*.

Now, let use analyze the performance of *TOTAL-EXCHANGE*. In our algorithm, we assume that when a processor broadcasts a set of messages, it broadcasts all messages it received in the previous step. As a consequence, duplicated messages are broadcast. We show that even with duplicated messages, the performance of *TOTAL-EXCHANGE* is within a constant factor of the optimal. In the first **for** statement, $2(n-1)$ messages are collected by each dimension 0 processor $\langle a, b \rangle$ using two hyperlinks e_a and e_b, and this takes $(n-1)$ message steps. Consider the **for** loop. It has n iterations. In the first iteration, $2n$ messages are broadcast from each dimension 0 processor $\langle a, b \rangle$ to all processors attached to e_a and e_b. In the second iteration, $4n$ messages are broadcast from each dimension 1 processor $\langle a, b \rangle$ to all processors attached to e_a and e_b. In general, in the iteration with $k = m$, $2^{m+1}n$ messages need to be broadcast by each dimension m processor $\langle a, b \rangle$ to all processors attached to e_a and e_b. By (ii) and (iii) above, the iteration with $k = m$ takes no more than $2^{m+1}n$ message steps. Therefore, *TOTAL-EXCHANGE* requires no more than $(n-1) + (2 + 4 + 8 + \cdots + 2^n)n = (2^{n+1} - 1)n - 1$ parallel message steps. We summarize this analysis by the following claim.

Theorem 12 *Assuming bus hyperlinks of Q_n^*, algorithm TOTAL-EXCHANGE carries out a total exchange operation in $4N_n - n - 1$ parallel message steps.*

□

6.5 Summary and Discussions

We proposed a new class of hypernetworks based on the duals of hypercubes. The structures of Q_n^* and Q_n are quite different, but as we showed, many properties of Q_n^* can be directly derived from the properties of Q_n. The Q_n^* hypernetwork is suitable for exploiting the high bandwidths provided by new interconnection technologies such as optical fiber or devices. We presented a set of basic data communication algorithms for Q_n^* based on bus implementation of hyperlinks. Algorithms *ROUTE* are *BROADCAST* are optimal. The algorithms *REDUCTION* and *TOTAL-EXCHANGE* are optimal within a constant factor. A trivial lower bound for the communication complexity of the permutation operation in n. We believe that the performance of algorithm *PERMUTATION* can be improved. However, an improved algorithm can be much more complicated.

Except *PERMUTATION*, our algorithms are closely related to the ideas behind their corresponding algorithms on the hypercube network. This leads us to a pose an open problem: is there a simulation scheme that can be used to simulate Q_n by Q_n^* efficiently? If such a scheme can be found, then all previously known hypercube algorithms can be automatically translated to algorithms for a machine using Q_n^* as the interconnection network.

Using the hypergraph dual concept, one can obtain another class of hypernetworks that contains the duals of the star graphs. The n-star graph S_n (refer to [1] for its definition) is a point-to-point network that has $n!$ vertices, $n(n-1)!/2$ edges, and its degree and the diameter are $n-1$ and $\lfloor 3(n-1)/2 \rfloor$, respectively. S_n is vertex and edge symmetric. Therefore, S_n^* has $n!(n-1)/2$ vertices and $n!$ hyperedges; S_n^* is 2-regular, $(n-1)$-uniform, linear, and vertex and hyperedge symmetric; and the diameter of S_n^* is no greater than $\lfloor (3n-1)/2 \rfloor$, respectively. Both of diameter and degree of S_n^* are sub-logarithmic functions of the number of processors and the number of hyperlinks in S_n^*. Compared with Q_n^*, S_n^* has some advantages. The topological and communication aspects of the S_n^* hypernetworks deserve further investigations.

Chapter 7

Conclusions

In this doctoral research, we first proposed a linear array architecture based on a pipelined TDM optical bus. We showed that using the conditional-delay and coincidence pulse techniques, several fundamental operations can be carried out very efficiently on our bus system and we demonstrated how to design parallel algorithms on our linear architecture. For example, parallel selection can be done in optimal $O(\log(n))$ time and parallel sorting can be done in $O(\log(n))$ time where n is the number of data elements. Our bus structure is simpler and more powerful than the one proposed in [26], and much simpler than and as powerful as the bus of [57]. Our linear array can be used as building blocks to construct processor arrays of multiple dimensions to achieve better scalability and performance.

Second, we introduced a pipelined asynchronous TDM optical bus based on the coincident pulse technique. We also proposed a reconfigurable version of this bus to solve the unfairness problem. Our simulation results indicate that the performance of our TDM bus is much better than the performance of its FA-TDM counterpart. It is interesting that our bus architecture can be used to implement an $n \times n$ switch. Again our bus structure can be used as building blocks to construct processor arrays of multiple dimensions.

We then proposed a class of reconfigurable buses, the segmented buses, and constructed multidimensional interconnection structures using such buses. We showed that these interconnection structures can be used to build versatile general-purpose parallel machines. To improve bandwidth, reliability, versatility, one may connect linearly arranged proces-

sors by several segmented buses instead of one. Interconnection patterns that are more complex than multidimensional meshes are possible. Segmented buses may appear in a hierarchical system design, providing chip-to-chip, module-to-module, board-to-board or node-to-node communication. We believe that segmented buses are more promising in hardware implementation than most other reconfigurable buses, such as the ones used in reconfigurable meshes, because of their relatively simpler control schemes. Our segmented bus it can be implemented in either electronic or optical domain, and the implementation is simple.

At last, we proposed a new class of hypernetworks base on the duals of hypercubes. Although the structures of Q_n^* and Q_n are quite different, as we showed, many properties of Q_n^* can be directly derived from the properties of Q_n. The Q_n^* hypernetwork is suitable for exploiting the high bandwidths provided by new interconnection technologies such as optical fiber or devices. We presented a set of basic data communication algorithms for Q_n^* based on bus implementation of hyperlinks. Algorithms *ROUTE* and *BROADCAST* are optimal. The algorithms *REDUCTION* and *TOTAL_EXCHANGE* are optimal within a constant factor. A trivial lower bound for the communication complexity of the permutation operation is n. We think that the performance of algorithm *PERMUTATION* can be improved. However, an improved algorithm can be much more complex.

Except *PERMUTATION*, our algorithms are closely related to the ideas behind their corresponding algorithms on the hypercube network. This leads us to a pose an open problem: is there a simulation scheme that can be used to simulate Q_n by Q_n^* efficiently? If such a scheme can be found, then all previously known hypercube algorithms can be automatically translated to algorithms for a machine using Q_n^* as the interconnection network.

Using the hypergraph dual concept, one can obtain another class of hypernetworks that contains the duals of the star graphs. n-star graph S_n. The n-star graph S_n (refer to [1] for its definition) is a point-to-point network that has $n!$ vertices, $n(n-1)!/2$ edges,

and its degree and the diameter are $n-1$ and $\lfloor 3(n-1)/2 \rfloor$, respectively. S_n is vertex and edge symmetric. Therefore, S_n^* has $n!(n-1)/2$ vertices and $n!$ hyperedges; S_n^* is 2-regular, $(n-1)$-uniform, linear, and vertex and hyperedge symmetric; and the diameter of S_n^* is no greater than $\lfloor (3n-1)/2 \rfloor$, respectively. Both of diameter and degree of S_n^* are sub-logarithmic functions of the number of processors and the number of hyperlinks in S_n^*. Compared with Q_n^*, S_n^* has some advantages. The topological and communication aspects of the S_n^* hypernetworks deserve further investigations.

Bibliography

[1] S. Akers, D. Harel, and B. Krishnamurthy. The star graph: an attractive alternative to the n-cube. In *Proceedings of 1987 International Conference on Parallel Processing*, pages 393–400, 1987.

[2] Selim G. AKL. *The Design and Analysis of Parallel Algorithms*, pages 39–58. Prentice-Hall, 1989.

[3] R. Alferness, L. Buhl, S. Korotky, and R. Tucker. High-speed $\delta\beta$-reversal directional coupler switch. *Top. Meeting Photon. Switching, Tech. Dig. Series*, 13:77–78, 1987.

[4] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The tera computer system. In *Proc. ACM Int. Conf. Supercomputing*, 1990.

[5] S. Araki, M. Kajita, K. Kasahara, K. Kubota, K. Kurihara, I. Redmond, E. Schenfeld, and T. Suzaki. Experimental free-space optical network for massively parallel computers. *Applied Optics*, 35(8):1269–1281, 1996.

[6] A. Benner, H. Jordan, and V. Heuring. Optically switched lithiumni obate directional couplers for digital optical computing. *SPIE Proc. Digital Optical Computing II*, 1215:343–352, 1990.

[7] A. Benner, H. Jordan, and V. Heuring. Digital optical computing with optically switched directional couplers. *Optical Engineering*, 30(12):1936–1941, 1991.

[8] C. Berge. *Hypergraphs.* North-Holland, 1989.

[9] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[10] D.A. Carlson. Performing tree and prefix computations on modified mexh-connected parallel computers. In *Proceedings of the 1985 IEEE International Conference on Parallel Processing*, 1985.

[11] S. Chandran and A. Rosenfeld. *Order statistics on a hypercube.* Center for Automation Research, University of Maryland, College Park, Md., 1986.

[12] D.M. Chiarulli, R.G. Melhem, and S.P. Levitan. Using coincident optical pulses for parallel memory addressing. *IEEE Computer*, 20(12):48–58, 1987.

[13] K.L. Chung. Generalized mesh-connected computers with multiple buses. *Proc. Int. Conf. on Parallel and Distributed System*, 1993.

[14] W.J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. on Computers*, 39:775–785, 1990.

[15] D.Bhagavathi, P.H.Looges, S. Olariu, J.L. Schwing, and J. Zhang. A fast selection algorithm for meshes with multiple broadcasting. In *Proceedings of the International Conference on Parallel Processing, III-10*, 1992.

[16] O.M. Dighe, R. Vaidyanathan, and S.Q. Zheng. The bus-connected ringed tree: a versatile interconnection network. *Journal of Parallel and Distributed Computing, to appear*, 1997.

[17] Patrick W. Dowd. Wavelength division multiple access channel hypercube processor interconnection. *IEEE Trans. on Computers*, 41(10):1223–1241, 1992.

[18] R. Duncan. A survey of parallel computers architectures. *IEEE Trans. on Computers*, pages 5–16, 1990.

[19] Hossam ElGindy and Paulina Wegrowicz. Selection on the reconfigurable mesh. In *Proceedings of the International Conference on Parallel Processing, III-26*, 1991.

[20] R. Floren and et all. Optical interconnects in the touchstone supercomputer program. *SPIE Proc. Intergrated Optoelectronics for Communication and Processing*, 1582:46–54, 1989.

[21] Kanad Ghose, R. Kym Horsell, and Nitin K. Singhvi. Hybrid multiprocessing using wdm optical fiber interconnections. In *mppoi*, 1994.

[22] J. W. Goodman et al. Optical interconnections for vlsi systems. *Proc. IEEE*, 72(7):850–866, 1984.

[23] J.R. Goodman and P. Woest. The wisconsin multicube: A new large-scale cache-coherent muptiprocessor. In *Proc. 15th Int. Symp. Computer Arch.*, 1988.

[24] Mathew S. Goodman et al. The lambdanet multiwavelength network: Architecture, applications, and demonstrations. *IEEE J. on Selected Areas in Communications*, 8(6), 1990.

[25] A. Guha, J. Bristow, C. Sullivan, and A. Husain. Optical interconnections for massively parallel architectures. *Applied Optics*, 29(8), 1980.

[26] Z. Guo. Sorting on array processors with pipelined buses. In *Proceedings of 1992 International Conference on Parallel Processing*, pages 289–292, 1992.

[27] Z. Guo et al. Array processors with pipelined optical buses. *J. Parallel Distributed Comput.*, 12(3):269–282, 1991.

[28] S.H. Horng. Performing prefix computation and its applications on modified mesh-connected computers with hyperbus broadcasting. *ICCI*, 1994.

[29] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[30] D. K. Hunter and D. G. Smith. New architectures for optical tdm switching. *Journal Lightwave Technology*, 11:495–511, 1993.

[31] K. Hwang. Omp: A risc-based multiprocessor using orthogonal access memories and multiple spanning buses. In *Proc. ACM Int. Conf. Supercomputing*, 1990.

[32] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing*, pages 87–88. McGraw-Hill, 1984.

[33] K. Hwang, P.S. Tseng, and D. Kim. An orthogonal multiprocessor for parallel scientific computations. *IEEE Trans. Computers*, 38, 1989.

[34] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, pages 87–88. McGraw-Hill, 1993.

[35] J. Jahns and S. H. Lee. *Optical Computing Hardware*. Academic Press, 1994.

[36] J. Jang and V.K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. In *Proc. 6th Int. Parallel Processing Symposium*, pages 130–137, 1992.

[37] Harry F. Jordan, Daeshik Lee, Kyungsook Y. Lee, and Srinivasan V. Ramanan. Serial array time slot interchangers and optical implementations. *IEEE Trans. on Computers*, 43(11), 1994.

[38] B. Kahle, E. Parish, T. Lane, and J. Quam. Optical interconnects for interprocessor communications in the connection machine. In *IEEE Conference on Computer Design*, 1989.

[39] P. Lalwaney, L. Zenou, A. Ganz, and I. Koren. Optical interconnects for multiprocessors: cost performance analysis. In *Proc. of the Smposium on Frontiers of Massively Parallel Computation*, 1992.

[40] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercube*, pages 78–82, 239–244. Morgan Kaufmann Publishers, Inc., 1992.

[41] S.P. Levitan, D.M. Chiarulli, and R.G. Melhem. Coincident pulse techniques for multiprocessor interconnection sturctures. *Applied Optics*, 29(14):2024–2039, 1990.

[42] H. Li and M. Maresca. Polymorphic-torus network. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(3):233–243, 1989.

[43] Y. Li, A.W. Lohmann, Z.G. Pan, S.B. Rao, I. Redmond, and T. Wang. Optical multiple-access mesh-connected bus interconnects. *Proc. of IEEE*, 82(11):1690–1700, 1994.

[44] Y. Li and S.Q. Zheng. Prefix computation using a segmented bus. In *Proc. of the 28th IEEE Southeastern Symposium on System Theory*, 1996.

[45] Y. Li and S.Q. Zheng. Asynchronous optical tdm communication for parallel computers. *Proceedings of SPIE's Photonics West'97*, 1997.

[46] R. Lin, S. Olariu, J. Schwing, and J. Zhang. Sorting in o(1) time on a reconfigurable mesh of size $n \times n$. In *Proceedings of EWPC'92, Plenary Address, IOS Press*, pages 16–27, 1992.

[47] A. Louri and L. Sung. An optical multiple-mesh hypercube - a scalable optical interconnection network for massively-parallel computing. *Journal of Lightwave Technology*, 12(4):704–716, 1994.

[48] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[49] R. Melhem, D. Chiarulli, and S. Levitan. Space multiplexing of waveguides in optically interconnected multiprocessor systems. *Computer Journal*, 32(4):362–369, 1989.

[50] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, and Q.F. Stout. Data movement operations and applications on reconfigurable vlsi arrays. In *Proceedings of the International Conference on Parallel Processing, 1, 205-208*, 1988.

[51] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, and Q.F. Stout. Meshes with reconfigurable buses. In *Proc. of the International Conf. on Parallel Processing*, volume 1, pages 205–208, 1988.

[52] John A. Neff. Optical interconnects based on two-dimensional vcsel arrays. In *mppoi*, 1994.

[53] M. Nigam and Sahni. Sorting n numbers on $n \times n$ mesh with buses. In *Technical Report #92-5, University of Florida, Gainsville*, 1992.

[54] S. Olariu and J. Schwing. A new deterministic sampling scheme with applications to broadcast efficient sorting on the reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 1996.

[55] Y. Pan. Hough transform on arrays with an optical bus. In *Proceedings of the 5th International Conference on Parallel and Distributed Computing and Systems*, pages 161–166, 1992.

[56] Y. Pan. Order statistics on optically interconnected multiprocessor systems. In *mppoi*, 1994.

[57] Y. Pan and K. Li. Linear array with a reconfigurable pipelined bus system – concepts and applications. In *Proceedings of 1996 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1431–1442, 1996.

[58] Craig Partridge. *Gigabit Networking*, pages 20–21. Addison-Wesley Publishing Company, 1993.

[59] S. Pavel and S.G. Akl. On the power of arrays with reconfigurable optical buses. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1443–1454, 1996.

[60] V.K. Prasanna-Kumar and C.S. Raghavendra. Array processor with multiple broadcasting. *Journal of Parallel and Distributed Computing*, 2(4):173–190, 1987.

[61] P.R.Prucnal, I. Glesk, and J.P. Sokoloff. Demonstration of all-optical self-clocked demultiplexing of tdm data at 250 gb/s. In *mppoi*, 1995.

[62] C. Qiao, R. Melhem, D. Chiarulli, and S. Levitan. Optical multicasting in linear arrays. *International Journal of Optical Computing*, 2(1), 1991.

[63] Chunming Qiao. On designing communication-intensive algorithms for a spanning optical bus based array. *Parallel Processing Letters(to appear)*, 1995.

[64] Chunming Qiao and Rami Melhem. Reconfigureation with time division multiplexed min's for multiprocessor communications. *IEEE Trans. on Parallel and Distributed Systems*, 5(4), 1995.

[65] Chunming Qiao and Rami G. Melhem. Time-division optical communications in multiprocessor arrays. *IEEE Trans. on Computers*, 42(5):557–590, 1995.

[66] S. V. Ramanan, H. F. Jordan, and J. R. Sauer. A new time domain, multi-stage permutation algorithm. *IEEE Trans. Inform. Theory*, 36:171–173, 1990.

[67] I. Redmond and E. Schenfeld. Experimental results of a 64-channel, free-space optical interconnection network for massively parallel processing. *Institute of Physics Conference Series*, 1995.

[68] John H. Reif and Akitoshi Yoshida. Free space optical message routing for high performance parallel computers. In *mppoi*, 1994.

[69] J. Rothstein. Bus automation, brains, and mental models. *IEEE Trans. on Systems Man Cybernetics*, 18, 1988.

[70] C.L. Seitz. Concurrent vlsi architectures. *IEEE Trans. on Computers*, 33:1247–1265, 1984.

[71] H.J. Siegel. *Interconneciton Networks for Large Scale Parallel Processing.* McGraw Hill Publishing Co., 1990.

[72] D.M. Spirit, A.D. Ellis, and P.E. Barnsley. Optical-time division multiplexing - systems and networks. *IEEE Commnunications Magazine*, 32(12):56–62, 1995.

[73] F. Stone, J. Watson, D. Moser, and W. Minford. Performance and yield of pilot-line quantities of lithium niobate switches. In *SPIE Conf. Proc. OE/Fibers*, 1989.

[74] Q.F. Stout. Meshes with multiple buses. In *27th IEEE Symp. Found. Comput. Sci.*, 1986.

[75] S. Suzuki and K. Nagashima. Optical broadband communications network architecture utilizing wavelength-division switching technologies. In *Technical Digest, Topical Meeting on Photonic Switching (Optical Society of America*, Washington, DC, 1987.

[76] T. Szymanski. Hypermeshes - optical interconnection networks for parallel computing. *Journal of Parallel and Distributed Computing*, 26(1):1–23, 1995.

[77] Andrew S. Tanenbaum. *Computer Networks*, pages 57–59. Prentice-Hall, 1988.

[78] R. A. Thompson and P. P. Giordano. An experimental photonic time slot interchanger using optical fibers as reentrant delay-line memories. *Journal Lightwave Technology*, 5:154–162, 1987.

[79] J.L. Trahan, R. Vaidyanathan, and C.P. Subbaraman. Constant time graph algorithms on the reconfigurable multiple bus machine. *Parallel and Distributed Computing*, 1996.

[80] Y. Pan Y. Li and S.Q. Zheng. A pipelined tdm optical bus with conditional delays. In *Optical Engineering, to appear*, 1997.

[81] G.H.Chen Y.C.Chen, W.T.Chen and J.P.Sheu. Designing efficient parallel algorithms on mesh-connected computers with multiple broadcasting. *IEEE Trans. on Parallel and Distributed Systems*, 1(2):241–246, 1990.

[82] J.G. Zhang and G. Picchi. Self-synchronized all-optical time-division multiple-access broadcast network. *Electronics Letter*, 29(21):1871–1873, 1993.

[83] Yi-Mo Zhang, Xiao-Qing He, Ge Zhou, Wen-Yao Liu, Yong Wang, and Zhan-Ping Yin. Optical fiber interconnection system for massively parallel processor arrays. In *mppoi*, 1995.

[84] S.Q. Zheng. Hypernetworks - a class of interconnection networks with increased wire sharing. In *Part I - Part IV, Technical Reports, Department of Compute Science, Louisiana State University*, 1994.

[85] S.Q. Zheng. Hypercube hypernetworks : Implementations of hypercube with increased wire sharing. In *Proc. of the 8th International Conf. on Parallel Processing*, pages 452–457, 1995.

[86] S.Q. Zheng. Hypernetworks: A class of interconnection networks for new generation parallel computers. In *Proc. of International Conf. on Parallel Processing*, 1995.

[87] S.Q. Zheng. Sparse hypernetworks based on steiner triple systems. In *Proc. of International Conf. on Parallel Processing*, 1995.

[88] S.Q. Zheng and J. Wu. Dual of a complete graph as an interconnection network. In *Proc. of IEEE SPDP*, 1996.

Vita

Yueming Li was born at Shi Village, Xia County of Shanxi Province, People's Republic of China in 1961. He went through the village's elementary school and then Miaoqian High School in a neighboring town. After graduating from high school, he worked as a farmer.

In the spring of 1978, he entered the Beijing Institute of Iron and Steel, now named as University of Science and Technology Beijing. He studied Mining Machinery in Department of the Mining and Mineral Engineering. He obtained his Bachelor of Engineering and Master of Engineering in 1982 and 1984 respectively. After graduating from college, he worked as a mechanical engineer at Beijing Metallurgical Research Institute.

In the fall of 1991, he came to the United States to study western technology. He first enrolled at South Dakota State University to study in the Department of Agricultural Engineering and the Department of Computer Science. In the fall of 1993, after graduating with dual master degrees, he transfered to Louisiana State University to study for a doctoral degree in the Department of Computer Science. He is expecting the degree in the December of 1997.

During the studies of his doctoral degree, he has published numerous journal and conference papers. His major research interests include Parallel and Distributed Computing, Image Processing, Neural Networks and High Performance Software.

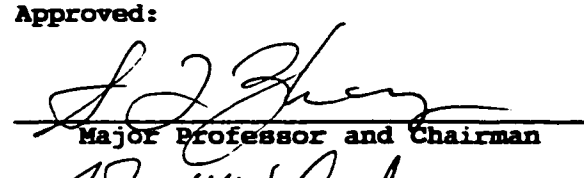
DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Yueming Li

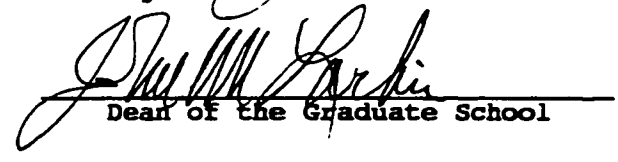
Major Field: Computer Science

Title of Dissertation: Design and Analysis of Optical Interconnection Networks for Parallel Computation

Approved:

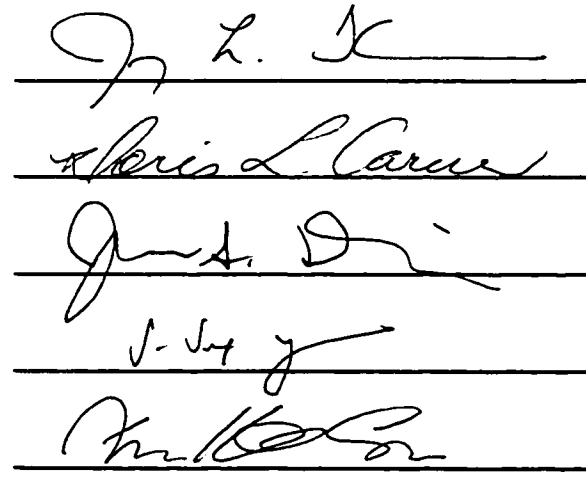


Major Professor and Chairman



Dean of the Graduate School

EXAMINING COMMITTEE:



Date of Examination:

August 27, 1997

IMAGE EVALUATION TEST TARGET (QA-3)

150mm

6"

APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989