Simulations and Algorithms on Reconfigurable Meshes With Pipelined Optical Buses.

Anu Goel Bourgeois

Louisiana State University and Agricultural & Mechanical College

SIMULATIONS AND ALGORITHMS ON RECONFIGURABLE MESHES WITH PIPELINED OPTICAL BUSES

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 Electrical and Computer Engineering

by
Anu Goel Bourgeois
B.S., Louisiana State University, 1991
M.S., Louisiana State University, 1997
May 2000

Acknowledgments

"True research is like fumbling in the dark for the right switches.
Once you've turned the light on everyone can see.........."

— author unknown

There are many people that have assisted me in some way and enabled me to complete this research. If it were not for the people mentioned below, I would still be "fumbling in the dark."

I would like to express my sincere appreciation to my major professor, Dr. Jerry Trahan, for his dedication, guidance, patience, and encouragement throughout this research. Working with such a remarkable person has been an enjoyable experience for me. He has been an amazing role model and I am truly grateful.

I would like to thank my committee members for their insightful suggestions and their time and effort. A special thanks to Dr. Vaidyanathan for his wisdom and collaboration during the course of this research.

Support for this research was partially funded by the Louisiana Board of Regents and Louisiana Engineering Society. The Department of Electrical and Computer Engineering and the College of Engineering also provided support enabling me to attend conferences and present this research to others in the field. Within the department, I am thankful for the support I received from the staff and faculty. Their concern and assistance provided constant encouragement and motivation. I am especially grateful for the time, advice, and motivation that Dr. Aravena has generously given me.

Finally, I want to extend my sincere gratitude to my friends and family for their unfailing confidence and reinforcement during the past few years. During my time as a graduate student I have made what I hope to be life-long friends and am grateful for their patience, advice, and support. In particular, I would like to thank Alberto Fernández for his friendship and collaboration. I would like to thank my friends outside of school and my family for their understanding and encouragement. A special thanks to my parents for their support and unconditional love. I am thankful that my brother made me realize that it is most important to enjoy what you do and follow your heart. Lastly, I am indebted to my husband for being so supportive while I followed my heart and allowing me to be myself. He has been a constant source of encouragement and support, mentally, physically, and intellectually.

Table of Contents

List of Figures

Abstract

Recently, many models using reconfigurable optically pipelined buses have been proposed in the literature. A system with an optically pipelined bus uses optical waveguides, with unidirectional propagation and predictable delays, instead of electrical buses to transfer information among processors. These two properties enable synchronized concurrent access to an optical bus in a pipelined fashion. Combined with the abilities of the bus structure to broadcast and multicast, this architecture suits many communication-intensive applications.

We establish the equivalence of three such one-dimensional optical models, namely the LARPBS, LPB, and POB. This implies an automatic translation of algorithms (without loss of speed or efficiency) among these models. In particular, since the LPB is the same as an LARPBS without the ability to segment its buses, their equivalence establishes reconfigurable delays (rather than segmenting ability) as the key to the power of optically pipelined models.

We also present simulations for a number of two-dimensional optical models and establish that they possess the same complexity, so that any of these models can simulate a step of one of the other models in constant time with a polynomial increase in size. Specifically, we determine the complexity of three two-dimensional optical models (the PR-Mesh, APPBS, and AROB) to be the same as the well known LR-Mesh and the cycle-free LR-Mesh.

We develop algorithms for the LARPBS and PR-Mesh that are more efficient than existing algorithms in part by exploiting the pipelining, segmenting, and multicasting characteristics of these models. We also consider the implications of certain physical constraints placed on the system by restricting the distance over which two processors are able to communicate.

All algorithms developed for these models assume that a healthy system is available. We present some fundamental algorithms that are able to tolerate up to $N/2$ faults on an N-processor LARPBS. We then extend these results to apply to other algorithms in the areas of image processing and matrix operations.

Chapter 1

Introduction

Advances in optoelectronic technologies have catapulted optical interconnects and optical computing to the forefront; this has opened up possibilities previously not considered in conventional electrical and electronic interconnection environments. An optically pipelined bus is one such example. It differs from an electronic bus in that it employs optical waveguides to transmit information. In such a model, many messages can be in transit simultaneously, pipelined in sequence on an optical bus, while the time delay between the furthest processors is only the end-to-end propagation delay of light over a waveguided bus. Currently, optical fiber is the preferred medium for telecommunication networks of long distances, due in part to its high bandwidth, reliability, low distortion, and low attenuation [38]. In parallel processing systems, communication efficiency determines the effectiveness of processor utilization, which, in turn, determines performance.

As a result, researchers have proposed several models based on pipelined optical buses as practical parallel computing platforms including the *Linear Array with a Reconfigurable Pipelined Bus System* (LARPBS) [38, 56, 73], the *Linear Pipelined Bus* (LPB) [54], the *Pipelined Optical Bus* (POB) [42, 79], the *Linear Array with Pipelined Optical Buses* (LAPOB) [18], the *Pipelined Reconfigurable Mesh* (PR-Mesh)

[72], the *Array with Reconfigurable Optical Buses* (AROB) [62, 63], the *Array Processors with Pipelined Buses* (APPB) [47], the *Array Processors with Pipelined Buses using Switches* (APPBS) [24], the *Array with Synchronous Optical Switches* (ASOS) [66], and the *Reconfigurable Array with Spanning Optical Buses* (RASOB) [65].

Many parallel algorithms, such as sorting [23], selection [54], matrix operations [38, 39, 62], Hough transform [53], singular value decomposition [55], nearest neighbor [57], and some numerical algorithms [26], exist for arrays with pipelined buses, indicating that such systems are very efficient for parallel computation due to the high bandwidth available by pipelining messages.

This dissertation focuses on two of the proposed optical models, specifically, the one-dimensional LARPBS and the multi-dimensional PR-Mesh. We present simulations for these models relating them to other similar optical models. We first relate the LARPBS to two other one-dimensional optical models, proving that the three models are equivalent. Next, we relate the PR-Mesh to other two-dimensional models, two with optical buses and two with electrical buses. We relate these two-dimensional models in the context of their computational power and prove that they belong to the same complexity class. These relations allow us to unify existing research on optical models and also to relate them to other well-established traditional models. This is the first work to determine relations between varying optical models.

We develop algorithms that are more efficient on these models than on other reconfigurable models that do not use optical buses. This is achieved by exploiting key features of optical models, such as pipelining and constant propagation delays. All existing algorithms for optical models assume that a healthy system is available, that is, all processors and switches are in working condition. This is not a reasonable assumption, therefore, we develop fault tolerant algorithms that are able to tolerate up to $N/2$ faults for an N-processor LARPBS. This provides the latitude of being able to develop algorithms without being concerned with the status of the available system. Fault-tolerant algorithms have been developed for other parallel architectures, however, this is the first work to address the issue for reconfigurable optical models.

The remainder of this chapter is organized as follows. Section 1.1 describes the main features of reconfigurable models with and without pipelined buses. Section 1.2 details the scope of the dissertation and the contributions of this work. Finally, Section 1.3 presents the organization of the dissertation.

1.1 Reconfiguration and Pipelining

Recently, researchers have proposed many reconfigurable models such as the *Reconfigurable Mesh* (R-Mesh) [5, 7, 45], *Linear Reconfigurable Mesh* (LR-Mesh) [5], *Fusing Reconfigurable Mesh* (FR-Mesh) [20, 22], *Processor Array with Reconfigurable Bus System* (PARBS) [77], *Reconfigurable Multiple Bus Machine* (RMBM) [74], and *Reconfigurable Buses with Shift Switching* (RESBIS) [44]. Nakano presented a bibliography of published research on reconfigurable models [48]. Chapter 2 describes some of these models in more detail.

Processors can fuse together the edges of a reconfigurable model to form buses (either electrical or optical buses) [6]. The main characteristics of these models are as follows.

- Each processor can locally determine its internal port connections and/or switch settings at each step to create or segment buses.
- The model assumes constant propagation delays on the buses.
- The model uses the bus as a computational tool.

The following examples demonstrate how reconfigurable models utilize these characteristics. Consider the OR operation on N bits, where each processor of an N-processor array holds an input. It is possible to perform this operation in constant time on an N-processor LR-Mesh. Assume $N = 8$ and that each processor holds one input bit and fuses its ports to form a single bus as shown in Figure 1.1(a).

Each processor that holds a value of '1' internally disconnects the bus and writes on the bus through its left port. The leftmost processor, R_0, reads the value on the bus; this value corresponds to the result of the OR operation (Figure 1.1(b)). If one or more processors hold a '1', then R_0 reads a '1' from the leftmost processor (R_2 in Figure 1.1(b)) holding a '1'. The processors between R_0 and R_2 all hold a '0', so they keep the bus intact and allow the value written by R_2 to reach R_0. If all processors hold a '0', then no value is written on the bus and the result is '0'. All processors then fuse their ports to connect the bus and processor R_0 broadcasts the result to all processors as in Figure 1.1(c).

The time required to perform this computation on a Parallel Random Access Machine (PRAM) with exclusive writes is $O(\log N)$ steps for N input bits. The demonstrated example performs the computation in a constant number of steps using only exclusive writes on a one-dimensional R-Mesh. In the second step, although both R_2 and R_6 are writing simultaneously, the two processors are writing on separate buses, maintaining an exclusive write.

The example demonstrates some of the key features of reconfigurable models. First, processors determine their internal port configurations based only upon the local variable held; those with a '1' disconnect their ports and those with a '0' connect their ports. Second, broadcasting a value on a bus takes a single step due to the assumption of constant propagation delay on a bus.

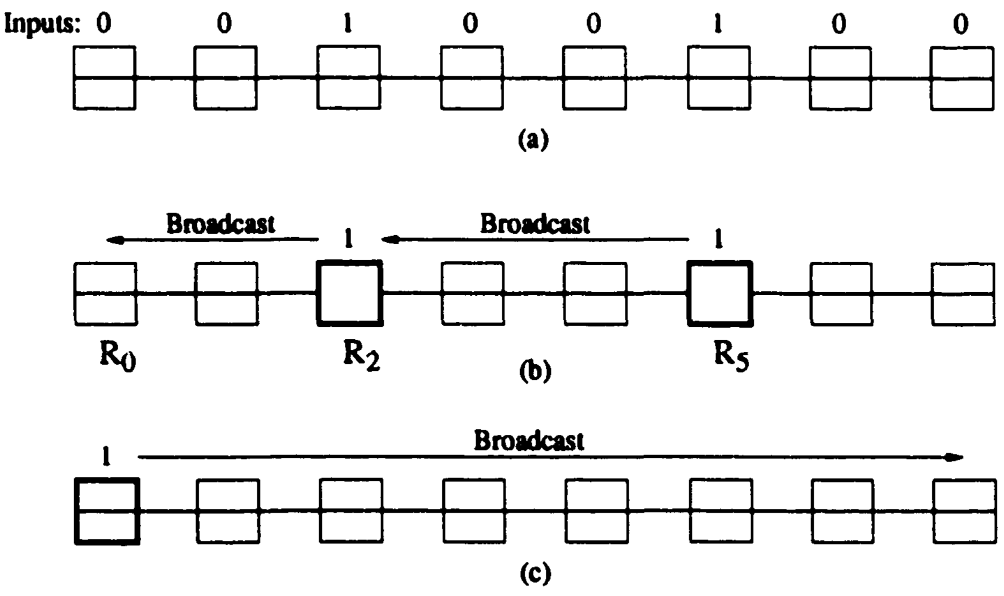


Figure 1.1: Computing the OR function on an LR-Mesh: (a) initial configuration; (b) disconnect bus and broadcast toward R_0; (c) broadcast result.

Next consider computing a binary sum on an R-Mesh. This is a two-dimensional model in which each processor has four ports (North, South, East, and West). The processors on the bottom row hold the input bit values.

First, all processors form vertical buses by fusing their North and South ports. Each processor on the bottom row broadcasts its input value to all processors on its vertical bus. A processor that reads a '0' on its vertical bus fuses its East and West ports together. A processor that reads a '1' on its vertical bus fuses its North and West ports together and its South and East ports. (Refer to Figure 1.2. The figure only shows the first four rows of the R-Mesh.)

The processor at the bottom left corner writes a signal at its West port. The internal port connections form staircase buses allowing a signal to step up a row for each '1' in the input. Figure 1.2 shows in bold the bus on which the signal propagates. The processors in the rightmost column read their East port. The

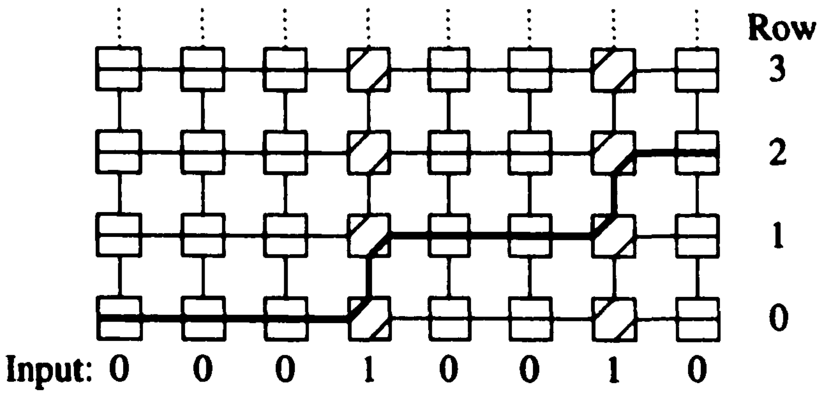


Figure 1.2: Summation of eight bits on an R-Mesh

processor that detects a signal determines the sum to be the same as its row value. This technique uses an $(N + 1) \times N$ R-Mesh to sum N bits in constant time. This example demonstrates the method of using the bus as a computational tool. In Section 4.2.1 we develop a binary prefix sums algorithm that runs on an N-processor LARPBS in constant time for N input bits.

The examples that we have considered thus far all can be executed on systems with either optical or electrical buses. Using optical waveguides provides us with the advantage of being able to pipeline messages on a bus. This is the ability of having multiple messages on a single bus concurrently. Chapter 2 provides more detail on how it is possible to pipeline messages on an optical bus.

We will use a general permutation routing example to illustrate the benefit pipelining provides. Let $\mathcal{N} = \{0, 1, \ldots, N-1\}$ and let $\pi : \mathcal{N} \longrightarrow \mathcal{N}$ be a bijection. Permutation routing of N elements on an N-processor system refers to sending information from processor i to processor $\pi(i)$, for each $i \in \mathcal{N}$. We will first describe how to implement this on an R-Mesh and then contrast this with how the LARPBS can perform a general permutation routing step more efficiently by using pipelining.

Consider a 4×4 R-Mesh in which each processor in column i on the bottom row holds $\pi(i)$ as shown in Figure 1.3. Assume each processor is to send value v_i to $\pi(i)$ on the bottom row.

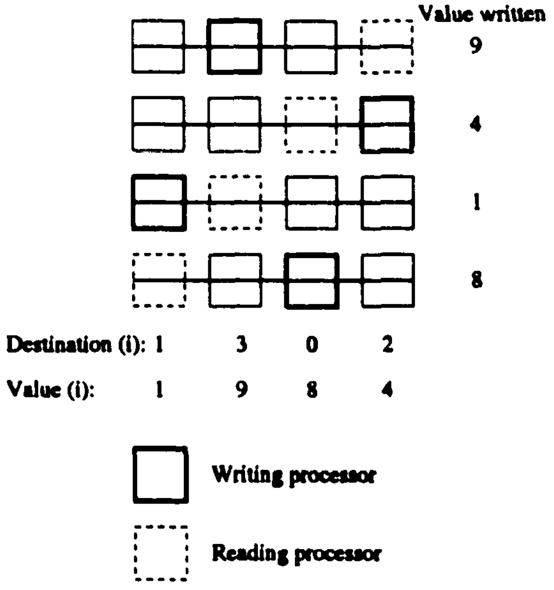


Figure 1.3: Permutation routing on an R-Mesh

First, each processor fuses its North and South ports forming vertical buses. Each processor on the bottom row broadcasts $\pi(i)$ and v_i along the vertical buses to all processors on the column. Next, all processors fuse their East and West ports forming horizontal buses. The processor with column index i and row index $\pi(i)$ writes v_i on the row bus as shown in Figure 1.3. Each processor with column index j and row index j reads from the bus with row index j. The processors then fuse their North and South ports forming vertical buses again. Each processor that read a value in the previous step writes on the bus so that the processors in the bottom row can read the value from the permutation.

If there are N inputs, then an $N \times N$ R-Mesh is required to execute a permutation routing in $O(1)$ steps. If an N-processor, one-dimensional R-Mesh is all that is available, then, by a simple bisection width argument, it would require N communication steps to route the permutation.

Pipelining enables an N-processor one-dimensional LARPBS to perform this general permutation routing in a single step. The properties of an optical waveguide support the propagation of multiple messages on a single bus during one communication step. (We discuss the details of pipelining messages in Chapter 2.) All processors of an LARPBS can concurrently select distinct destinations and each sends a message to its chosen destination in one bus cycle. To perform the permutation routing, each processor i selects $\pi(i)$ as its destination and sends its value v_i on the data waveguide. This ability of optical buses provides a savings in size and/or time.

1.2 Scope and Contributions of the Dissertation

The aim of this dissertation is to further demonstrate the claim that pipelined optical models are powerful parallel architectures and to show how these models fit into the well established hierarchy of complexity classes. We accomplish this by proceeding in two directions:

- Development of simulations relating models to one another, and
- Algorithm development.

We first develop a cycle of simulations between three one-dimensional optically pipelined models. This establishes the equivalence of these models in the sense that any step of one model can be simulated by either of the other two in a constant number of steps using the same number of processors. This result implies that any of these models can efficiently execute any algorithm designed for any of these models regardless of their structure differences.

Expanding these results to relate two-dimensional models is not straightforward: two-dimensional models have many different bus configurations that can be formed at any given step and the models that are considered have considerable differences in their features and capabilities. For instance, two of the models are able to change their switch configurations multiple times within a bus cycle. Another has additional hardware such as a relative delay counter and a rotate shift register and is also able to insert multiple delays at each processor within a bus cycle. As a result, we relate these models in a different context. Rather than focusing on equivalence as defined above, we relate models to within a constant factor of time while allowing a polynomial increase in the number of processors. The motivation for associating models in this way is that this relates time and processor-bounded complexity classes for these models. (Such a complexity class is the class of problems that can be solved by the model with the given time and processor resources.) Furthermore, this setting permits relating complexity classes based on these models to established complexity classes, firmly locating the abilities of these models relative to more widely studied, traditional models. Other reconfigurable models have been placed within established complexity classes, however, no effort had been given to place reconfigurable optical models within these classes.

We establish that the PR-Mesh has the same complexity as the cycle-free *Linear Reconfigurable Network* (LR-Mesh). In other words, any step of the PR-Mesh can be simulated by the cycle-free LR-Mesh or vice versa within constant time allowing a polynomial increase in processors. We also prove that in constant time using a polynomial number of processors the cycle-free LR-Mesh can solve the same class of problems as the LR-Mesh. This result implies that the PR-Mesh can solve the same class of problems within the same order of steps using polynomial processors. We extend this complexity class to include two other optical models, namely the *Array with Reconfigurable Optical Buses* (AROB) [62, 63] and the *Array of Processors with Pipelined Buses using Switches* (APPBS) [24].

Once the relations between different models are established, algorithms can be designed for one model and translated to the others appropriately. We therefore focus our attention to the LARPBS and PR-Mesh and develop algorithms for these two models.

We have developed algorithms in the areas of computational geometry, arithmetic operations, and image analysis. These algorithms modify existing algorithms to exploit pipelining and reconfiguration abilities, thereby providing savings in time and/or size, and improving efficiency.

Most algorithm development for reconfigurable models assumes availability of a healthy system with an unrestricted number of processors. Some of these assumptions are unrealistic and unfeasible for implementation. To accommodate this, we first considered limiting the communication distance between processors. With this approach, the length of the bus is unrestricted, however, the distance that a message is able to travel in a single step is limited. We present algorithms to compute binary prefix sums and perform compression on an N-processor LARPBS with the communication length restricted to L, where $L \leq N$. This results in a slowdown factor of N/L, which is optimal.

It is impractical to design an algorithm for a healthy system, only to have it be unusable due to a single faulty processor. Therefore, the next assumption that we tighten is allowing some processors to fail. We present several basic fault tolerant algorithms for the LARPBS. Specifically, we have developed algorithms to calculate binary prefix sums, perform compression, sort, and perform a general permutation routing step on an N-processor array that can have up to $N/2$ static faults. We then extend these results to other fault tolerant algorithms in the areas of image processing and matrix operations.

The relational results obtained (for both the one-dimensional models and the two-dimensional models) are some of the first to unify reconfigurable optical models to each other and relate them to other more widely known models. This is also the first work to consider physical restrictions and develop fault-tolerant algorithms for optically pipelined models.

1.3 Organization of the Dissertation

The dissertation is organized as follows. Chapter 2 describes the structure and addressing techniques of the LARPBS and PR-Mesh. The chapter also presents some fundamental algorithms that highlight the features of these models. This sets the framework for the remaining chapters of the dissertation.

Chapter 3 is a literature review that surveys other similar models and describes their differences from the LARPBS and PR-Mesh. The chapter provides an overview of algorithms that have been developed for optically pipelined models. The overview illustrates the key techniques utilized and the wide range of applications.

Chapter 4 presents a new algorithm to perform a binary prefix sums operation without using the segmenting ability of the LARPBS. This algorithm was presented at the *International Conference on Parallel and Distributed Computing Systems*, in New Orleans, Louisiana, in 1997 [73]. This algorithm provides the tool necessary to establish the equivalence of three one-dimensional optical models, namely the LARPBS, LPB, and POB. The work of this chapter was presented at the *International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, in

San Juan, Puerto Rico, in 1999 [70]. The work was also submitted to the *Journal of Parallel and Distributed Computing* [71].

Chapter 5 relates the PR-Mesh to other reconfigurable models with and without optical buses and establishes its complexity. Portions of this work appeared in *Parallel Processing Letters*, in 1998 [72]. This work will be presented at the *International Parallel and Distributed Processing Symposium*, in Cancun, Mexico [10]. It will also be published in the *International Journal on the Foundations of Computer Science* [9].

Chapter 6 develops algorithms for the LARPBS and PR-Mesh that are more efficient than existing algorithms. These algorithms are in the areas of computational geometry, arithmetic operations, and image analysis. The chapter also considers the implications of certain physical constraints and details the method to overcome these restrictions for performing binary prefix sums and compression.

Chapter 7 presents algorithms that can tolerate up to N faults for an N-processor LARPBS. We first present four fundamental fault-tolerant algorithms that can be used as building blocks for more extensive algorithms. We also describe how to use these building blocks to develop fault-tolerant algorithms for some matrix operations and image analysis. This work will be presented at the *Workshop on Optics and Computer Science*, in Cancun, Mexico [8].

Finally, Chapter 8 provides a summary of the dissertation and possible future work and extensions of the results.

Chapter 2

Model Description

A system with an optically pipelined bus uses optical waveguides instead of electrical buses to transfer information among processors. Signal (pulse) transmission on an optical bus possesses two advantageous properties: unidirectional propagation and predictable propagation delay per unit length. These two properties enable synchronized concurrent access to an optical bus in a pipelined fashion [25, 46, 66, 67]. Combined with the abilities of a bus structure to broadcast and multicast, this architecture suits many communication-intensive applications.

We adapt the following framework from Qiao and Melhem [66]. Organize data into fixed-length *data frames*, each comprising a train of optical pulses. The presence of an optical pulse represents a binary bit with value 1. The absence of an optical pulse represents a binary bit with value 0. Let ω denote the pulse duration. Define a unit pulse length Δ to be the spatial length of a single pulse; this is equivalent to the distance traveled by a pulse in ω units of time. The bus has the same length of fiber between consecutive processors, so propagation delays between consecutive processors are the same. Let τ denote the time for a signal to traverse the optical distance on the bus between two consecutive processors with spatial distance D_0; time τ is also referred to as a *petit cycle*.

As mentioned, the properties of an optical bus allow multiple processors to concurrently write on the bus by pipelining messages. This is possible provided that the following condition to assure no collisions is satisfied:

$$D_0 > b\omega c_g,$$

where b is the number of bits in each message and c_g is the velocity of light in the waveguide [25]. The assurance that all processors start writing their messages on the bus at the same time is another condition that must be satisfied to guarantee that no two messages will collide. Let a *bus cycle* be the end-to-end propagation delay on the bus. We specify time complexity in terms of a step comprising one bus cycle and one local computation.

The next section describes the structure of the *Linear Array with a Reconfigurable Pipelined Bus System* (LARPBS). This model will serve as a representative for linear arrays with optical buses in this work. Section 2.2 explains the addressing techniques of this model. Section 2.3 briefly describes two fundamental algorithms utilized by the LARPBS, namely binary prefix sums and compression. These algorithms highlight the key techniques of the LARPBS. Section 2.4 extends the one-dimensional model to a multi-dimensional optical model, called the *Pipelined Reconfigurable Mesh* (PR-Mesh). This model will serve as a representative for two-dimensional optical models in this work.

2.1 LARPBS Structure

In the LARPBS, as described by Pan and Li [56], the optical bus is composed of three waveguides, one for carrying data (the *data waveguide*) and the other two (the *reference* and *select waveguides*) for carrying address information (see Figure 2.1). (For simplicity, the figure omits the data waveguide, as it resembles the reference waveguide.) Each processor connects to the bus through two directional couplers, one for transmitting and the other for receiving [25, 66]. Note that optical signals propagate unidirectionally from left to right on the upper segment (transmitting segment) and from right to left on the lower segment (receiving segment), with a U-turn connecting the two segments. Referring to Figure 2.1, the processor furthest from the U-turn, R_0, is the *tail* of the bus, and the processor at the U-turn, R_4, is the *head*.

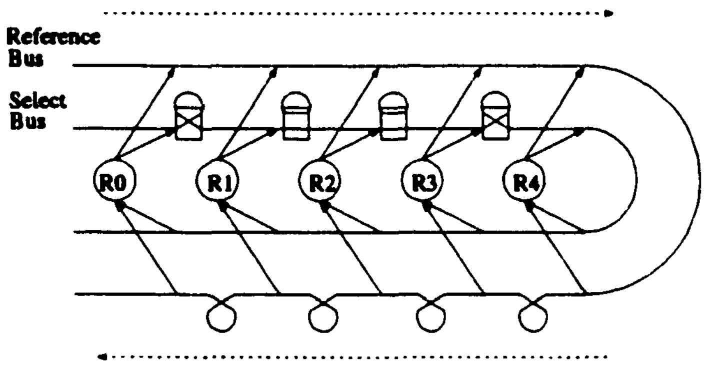


Figure 2.1: Structure of an LARPBS

The receiving segments of the reference and data waveguides contain an extra segment of fiber of one unit pulse-length, Δ, between each pair of consecutive processors (shown as a delay loop in Figure 2.1). The transmitting segment of the select waveguide has a switch-controlled conditional delay loop of length Δ between processors R_i and R_{i+1}, for each $0 \leq i \leq N-2$ (Figure 2.1). Processor $i+1$ controls the switch between processors i and $i+1$. A processor can set a switch to the *straight* or *cross* states, as shown in Figure 2.2. The length of a bus cycle for a system with N processors is $2N\tau + (N-1)\omega$.

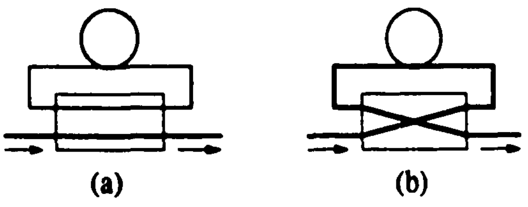


Figure 2.2: Conditional delay switch: (a) straight state; (b) cross state

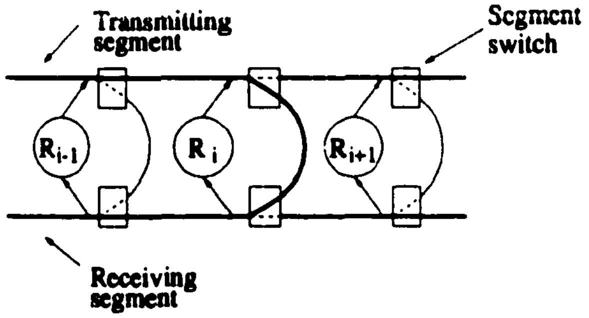


Figure 2.3: Segment switch

To allow segmenting, the LARPBS has optical switches on the transmitting and receiving segments of each bus for each processor. Let *trans*(i) and *recv*(i) denote these sets of switches on the transmitting and receiving segments, respectively, on the three buses between processors R_i and R_{i+1}. Switches on the transmitting segment are 1×2 optical switches, and on the receiving segment are 2×1 optical switches as shown in Figure 2.3. With all switches set to *straight*, the bus system operates as a regular pipelined bus system. Setting *trans*(i) and *recv*(i) to *cross* segments the whole bus system into two separate pipelined bus systems, one consisting of processors $R_0, R_1, \cdots, R_i$ and the other consisting of $R_{i+1}, R_{i+2}, \cdots, R_{N-1}$. Figure 2.4 shows an LARPBS with six processors, in which switches in *trans*(3) and *recv*(3) are set to cross, splitting the array into two subarrays with the first having four processors and the second having two processors. (For clarity, the figures show only one waveguide and omit conditional delay switches.)

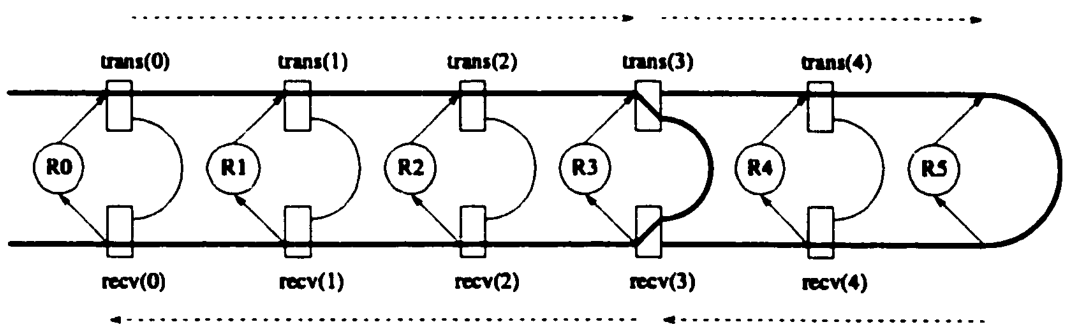


Figure 2.4: A six processor LARPBS model with two subarrays

2.2 Addressing Techniques

The LARPBS uses the *coincident pulse technique* [66] to route messages by manipulating the relative time delay of *select* and *reference* pulses on separate buses so that they will coincide only at the desired receiver. Each processor has a *select frame* of N bits (*slots*), of which it can inject a pulse into a subset of the N slots. For example, let all switches on the transmitting segment of the select waveguide be set straight to introduce no delay. Let source processor R_i send a reference pulse on the reference waveguide at time t_{ref} (the beginning of a bus cycle) and a select pulse on the select waveguide at time $t_{sel} = t_{ref} + (N - 1 - j)\omega$. Processor R_i also sends a data frame, on the data waveguide, that propagates synchronously with the reference pulse. After the reference pulse goes through $N - 1 - j$ fixed delay switches, the select pulse catches up to the reference pulse. As a result, processor R_j detects the double-height coincidence of reference and select pulses, then reads the data frame. Figure 2.5 shows a select frame relative to a reference pulse for addressing processor j. The coincident pulse technique admits broadcasting and multicasting of a single message by appropriately introducing multiple select pulses within a select frame.

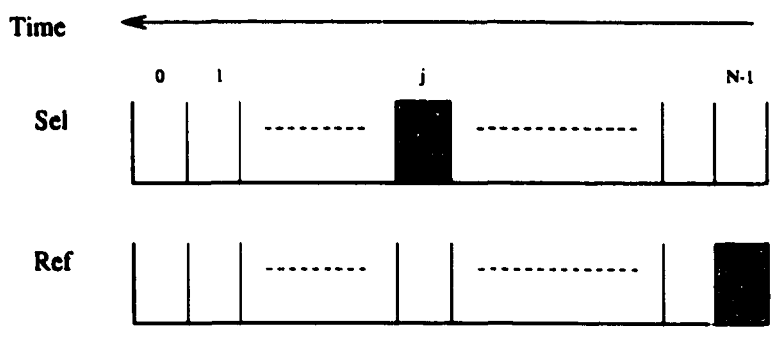


Figure 2.5: Select and reference frames

The conditional delay switches on the transmitting segment introduce delays to the select pulses and can alter the location at which the select and reference pulses will coincide. These switches are useful as a computing tool to calculate binary prefix sums and perform compression, for example (Section 2.3). The length of the bus between two processors provides enough space for two frames of N slots to fit, although there is only one such frame on each waveguide for each processor. This prevents a pulse in the select frame of processor R_i from being shifted to overlap the reference frame of R_{i-1}.

When multiple messages arrive at the same processor in the same bus cycle, it receives only the first message and disregards subsequent messages that have coinciding pulses at the processor. This corresponds to the PRIORITY concurrent write rule. The PRIORITY write rule has the processor with the highest priority (in this case, the processor with the highest index or nearest the U-turn) win a write conflict when multiple processors are attempting to write to the same destination.

We will refer to the processor that has a select pulse injected in its slot in a select frame for a particular message as the ***selected destination***. The ***actual destination*** will denote the processor that detects the coinciding reference and select pulses (the two may be different due to conditional delay loops and segmenting). The ***normal state*** *of operation* is when the actual destinations of messages are the selected destinations. For the LARPBS, the normal state of operation is when all conditional delay switches and segment switches are set to straight.

Consider the LARPBS shown in Figure 2.1. Suppose processor R_1 injects a select pulse so that R_3 is its selected destination, and R_0 attempts to broadcast. The message sent by R_1 encounters one conditional delay switch set to cross, and the message sent by R_0 encounters two. As a result, the actual destination of R_1 is R_2 instead of R_3. The actual destinations of the message broadcast by R_0 are R_2, R_1, and R_0, rather than all five processors. Even though R_2 is the actual destination of the message sent by R_0, processor R_2 will receive only the message sent by R_1 because this message arrives prior to the one sent by R_0.

2.3 Fundamental Algorithms

There are a few fundamental algorithms that find use as building blocks for other more extensive algorithms. Two that appear frequently are binary prefix sums and compression [56]. To demonstrate LARPBS operations, we will describe these in this section. The following chapters will use various forms of these algorithms. For instance, in Section 4.2.1, we describe a binary prefix sums algorithm that does not utilize the segmenting ability. Section 6.2 describes methods to perform binary prefix sums and compression on an array that has a restricted communication length. Section 7.3.1 provides fault tolerant algorithms to perform binary prefix sums and compression. These algorithms also play a role in relating different optical models to one another (Chapters 4 and 5).

2.3.1 Binary Prefix Sums

Consider an LARPBS with N processors such that each one holds a binary value, v_i, for $0 \leq i < N$. The i^{th} *binary prefix sum*, *psum*$_i$, is $v_0 + v_1 + \ldots + v_i$.

Lemma 2.1 [56] *Binary prefix sums of N elements can be computed on an N processor LARPBS in $O(1)$ steps.*

Proof: First, each processor R_i, $0 \leq i < N$, sets its conditional delay switch to straight if $v_i = 0$ and cross if $v_i = 1$. Referring to Figure 2.6(c), R_1 and R_4 both hold a value of '1'. Each processor sends a message containing its index addressed to processor R_{N-1}, that is, R_{N-1} is the selected destination for all messages. The conditional delay switches, however, will shift the pulses so that if $N - 1 - j$ is the number of switches set to cross after R_i, then the actual destination for processor R_i will be R_j. Processor R_j may receive multiple messages, however, it accepts only the first message to arrive in the bus cycle. Figure 2.6(c) shows the binary values held by processors that would induce switch settings as shown in Figure 2.1. Based on these values, R_3 receives a message from R_1, R_2, and R_3, but accepts only the message from R_3, as shown in Figure 2.6(a).

Next, processor R_j that received an index i then replies to R_i with a message containing its index. From the example, R_4 sends a message to itself, R_3 to itself, and R_2 to R_0 (Figure 2.6(b)). Since some messages may have been disregarded in the previous step, not all processors will receive a message in this step. To account for this, if R_i received a message from R_j during the second step, then it now segments the bus and broadcasts the index of j to its segment. The reason for this is that all processors within the same segment have the same prefix sums value. In our example,

R_0, R_3, and R_4 segment the bus and broadcast the values 2, 3, and 4, respectively (Figure 2.6(c)). Each processor stores the value it receives as x_i.

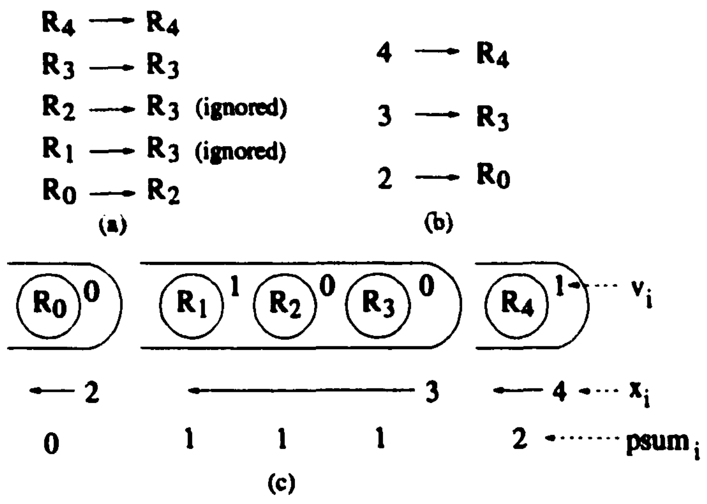


Figure 2.6: Binary prefix sums example: (a) actual destinations of first set of messages; (b) response to first message; (c) segmenting, broadcasting within segments, and computation steps.

Once R_0 receives the value x_0, it calculates the sum of all values in the array as $t = v_0 + (N - 1 - x_0) = 0 + (5 - 1 - 2) = 2$. Processor R_0 then broadcasts t to all processors, so that processor R_i can locally determine $psum_i = v_0 + v_1 + \ldots + v_i = t - (N - 1 - x_i)$. ■

The conditional delay switches are used to introduce unit delays, one unit delay for each input value of '1'. The effect of this is that select and reference pulses of all processors with the same prefix sum value coincide at the same processor, however, only one message from this group of processors is received. The segment switches enable the highest indexed processor of such a group to segment the bus and broadcast data relaying information necessary for each processor to locally compute its prefix sum. The ability to pipeline messages allows each processor to compute its prefix sum simultaneously on a single bus.

2.3.2 Compression

Consider an LARPBS with N processors, such that each processor holds one element and some of the elements are marked. Let there be x such marked elements. The *compression* algorithm compacts all marked elements to the lower end of the array, namely processors R_0 through R_{x-1}, maintaining their relative order. The algorithm also compacts all unmarked elements to the upper end of the array, namely processors R_x through R_{N-1}, maintaining their relative order.

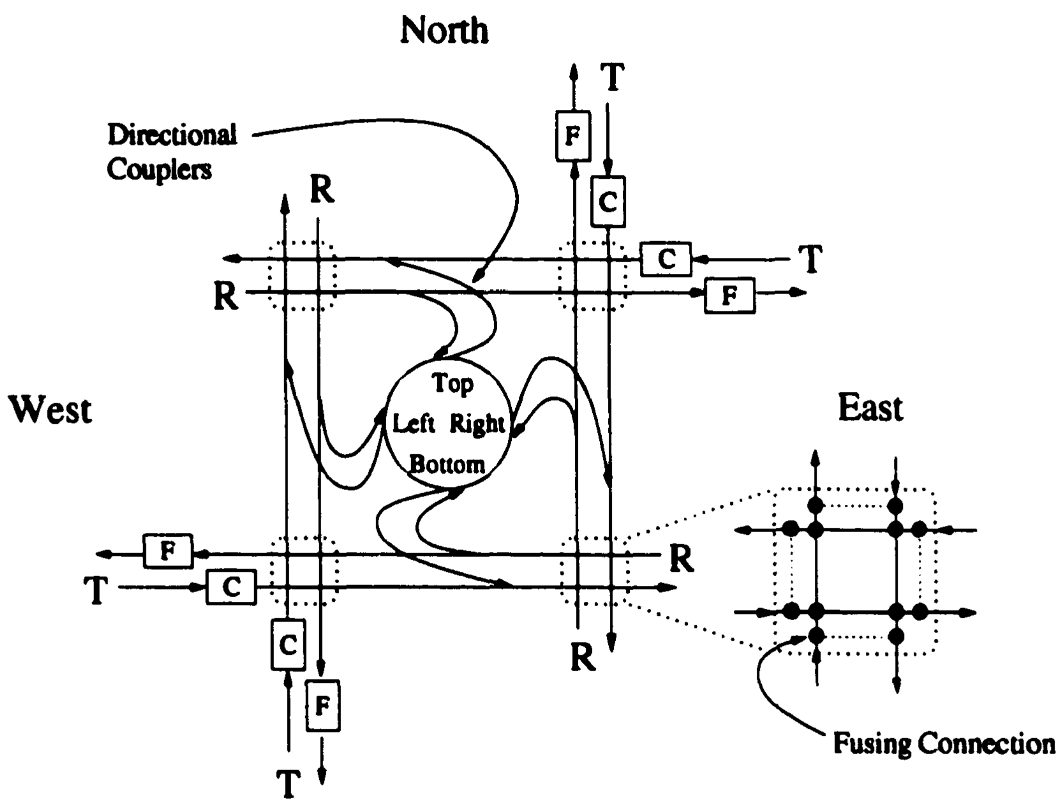
Lemma 2.2 [56] *Compression of x elements, where $x \leq N$, can be performed on an N processor LARPBS in $O(1)$ steps.*

<u>Proof:</u> Consider processor R_i, where $0 \leq i < N$, holding a marked element v_i. Processor R_i sets its conditional delay switch to cross and sends a message with its index i addressed to processor R_{N-1}. All processors holding unmarked elements set their conditional delay switches to straight. If R_i holds the marked element with the k^{th} largest index, then the actual destination for the message is R_{N-k}. Because of the conditional delays, each message written at this step arrives at a different destination processor.

Processor R_{N-k} that received an index i then replies to R_i with its index. Processor R_i stores k (that is, N minus this index $N-k$) as $count_i$; this will contribute towards determining the final position for the marked element v_i. Next, each processor holding a marked element multicasts its index to all processors above it. The lowest indexed processor R_g holding a marked element will not receive a message, and will thus determine that it has the lowest index. Processor R_g then broadcasts $count_g$ to all processors so that each processor R_i with a marked element can then locally determine the final position for its element as $compress_i = count_g - count_i$.

Repeat the process for the unmarked elements, however, the position received in the second step is its final position. Once all processors have determined the final positions, route all elements to their proper destinations. ■

2.4 PR-Mesh Description



T -- Transmitting segment C -- Conditional delay loop

R -- Receiving segment F -- Fixed delay loop

Figure 2.7: PR-Mesh processor connections

We define a new model that is a k-dimensional extension of the LARPBS called the *Pipelined Reconfigurable Mesh* (PR-Mesh). It is a mesh of processors in which each processor has $2k$ ports. Each processor can locally configure its ports by internally fusing pairs of ports or leaving ports as singletons, so all buses are linear. A two-dimensional PR-Mesh is an $R \times C$ mesh of processors in which each processor has four ports. The ports connect to eight segments of buses using directional couplers as shown in Figure 2.7. There are receiving and transmitting waveguides for the two dimensions and within each dimension there are waveguides for both directions. Each processor locally controls a set of switches at each of the bus intersections that allow it to fuse bus segments together. The dashed boxes around each bus intersection contain these sets of switches. (The intersection for the lower right corner of the processor is shown larger to distinguish the connections.) Each fusing connection can be in one of ten possible settings. The dashed segments within the box are auxiliary segments that enable the processor to create U-turns. Figure 2.8 depicts the ten possible port partitions for each processor of a two-dimensional PR-Mesh. To implement these partitions, the switches can configure from within the same set of configurations at the switch level. Local fusing creates buses that run through fused switches to adjacent processors, then through their fused switches, and so on. Each such linear bus corresponds to an LARPBS. The switches may not be set, however, so that a cycle is formed. By allowing cycles, there would be no clear head or tail of a bus, therefore, it would be impossible to determine priority among the processors for concurrent write operations.

Each processor locally controls conditional delay loops on each of the transmitting segments. There are also fixed delay loops on each of the receiving segments. The switches at each bus intersection act as the segment switches. Refer to Figure 2.7 for the placement of these switches. A pair of receiving and transmitting buses that are traversed in opposite directions corresponds to an LARPBS bus.

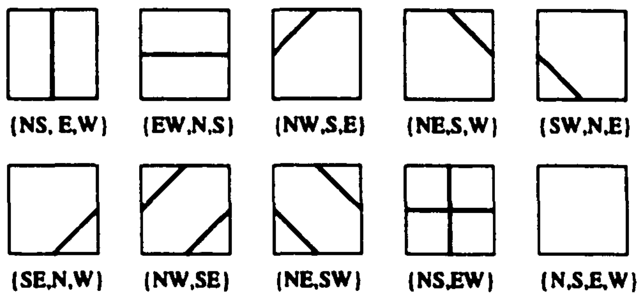


Figure 2.8: PR-Mesh switch connections

The following examples help to illustrate the processor and switch connections for different bus configurations. Consider a processor, R_i, that is connected to a segment of a horizontal bus, that is, it sets its configuration so that the East and West ports are fused. Also, assume that the North and South ports are tails of separate buses, or open rather than fused. Figure 2.9(a) pictorially shows a possible set of bus formations at processor R_i. Processor R_i configures its switch settings so that the East and West ports are fused and the North and South ports are left open. Refer to Figure 2.9(b) to see the connections of each bus intersection. With this example, the left reading and writing connections do not necessarily correspond to the West port because of bus routing internal to the processors. For example, a read from the West port would be performed by either the Top or Bottom read connections. Read and write operations for the North port are performed by the Left connections and read and write operations for the South port are performed by Right connections. The corresponding ports and connections are fixed for each bus configuration. Since there are only ten configurations, each processor can keep a table holding this information. Throughout this dissertation we will describe a read from the West port without reference to internal connections.

Once the bus is created, the orientation of the bus must be determined. To do this, the head of the bus broadcasts a message on the bus that corresponds to the correct direction and each processor connected waits for a message. For this example, if a message is sent on the upper horizontal segment, then R_i sends and receives messages using its Top port. If a message is sent on the lower horizontal segment, then R_i sends and receives messages using its Bottom port.

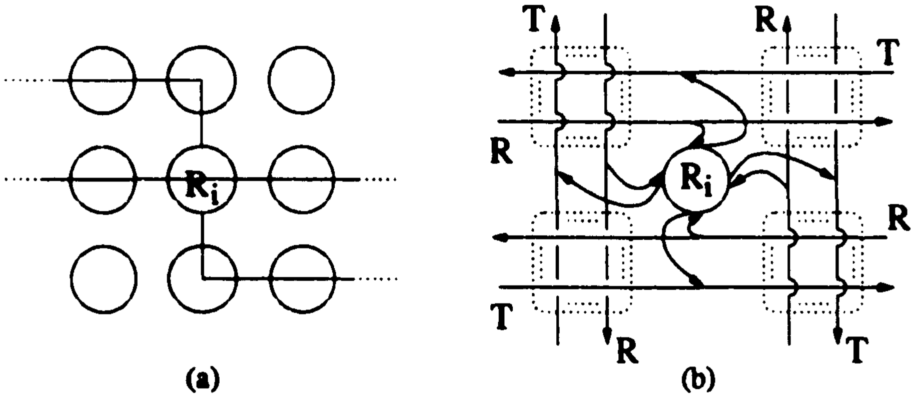


Figure 2.9: Example of PR-Mesh switch settings for {EW, N, S}

The next example illustrates the switch and port connections for creating U-turns. Consider a processor, R_j, that has each of its four ports at a U-turn of a bus, so that the processor is the head of four separate buses. Figure 2.10(a) pictorially shows a possible set of bus formations at processor R_j. Processor R_j configures its switch settings to create U-turns, utilizing the auxiliary segments, as shown in Figure 2.10(b). For this example, the Right connections handle communications for the North port. Left connections handle communications for the South port, Top connections for the East port, and Bottom connections for the West port.

The PR-Mesh is similar to the *Linear Reconfigurable Mesh* (LR-Mesh) [5] in that both allow processors to dynamically change switch settings to construct different buses. The LR-Mesh, however, uses electrical buses rather than optical buses. The

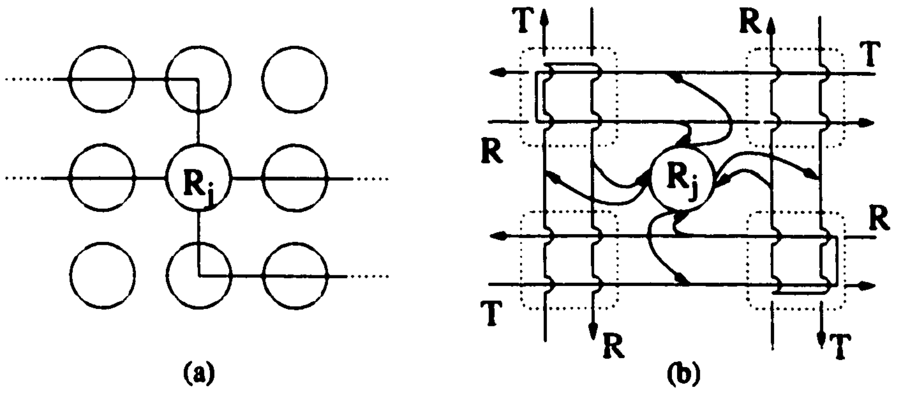


Figure 2.10: Example of PR-Mesh switch settings for {N, S, E, W}

available internal port configurations are the same as those available to the PR-Mesh (Figure 2.8), thus forming only linear buses. The buses, however, can form cycles, unlike the PR-Mesh buses.

A more general version of the LR-Mesh is the *Reconfigurable Mesh* (R-Mesh) [5, 7, 45]. This model is able to form non-linear buses, unlike the PR-Mesh, by allowing its processors to fuse its ports as shown in Figure 2.11 in addition to the ten partitions available to the PR-Mesh.

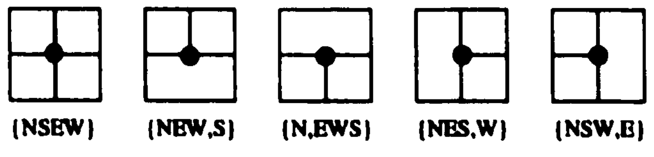


Figure 2.11: Non-linear R-Mesh port connections

Chapter 3

Literature Review

Most models based on optical buses similar to the LARPBS and PR-Mesh differ only by slight variations. For instance, they are all able to pipeline their messages. The differences among these models involve the switches used, the placement of the switches, and some other hardware features and capabilities.

The previous chapter described the structure and addressing techniques of the LARPBS and PR-Mesh in detail. This chapter considers other optical models and samples from the range of optical algorithms. In particular, Section 3.1 briefly describes other optically pipelined models that are similar to the LARPBS and PR-Mesh. Section 3.2 presents an overview of the types of algorithms that have been designed for these models.

3.1 Other Optical Models

The model most similar to the LARPBS is the *Linear Pipelined Bus* (LPB) [53]. This model is identical to the LARPBS with the exception that it does not have any segment switches. The *Pipelined Optical Bus* (POB) [42, 79] is similar to the LARPBS and LPB as it also contains three waveguides. Conditional delay switches are on the receiving segment of the reference and data waveguides rather than the transmitting segment of the select waveguide, and like the LPB, the POB does not have segment switches. We discuss these two models in more detail in Section 4.1 and show that in spite of these differences, the LARPBS, LPB, and POB are equivalent.

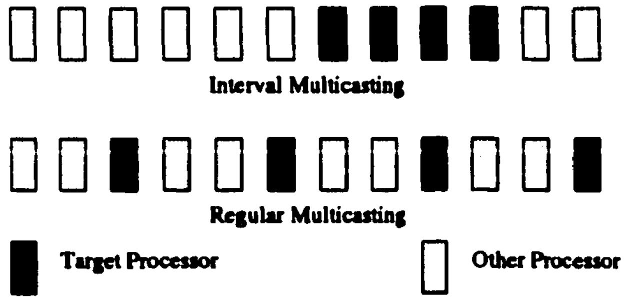


Figure 3.1: [18] Multicasting patterns

The *Linear Array with a Pipelined Optical Bus* (LAPOB) [18] is another model that uses directional couplers to connect to an optical bus. The model, however, does not possess either conditional delay or segment switches. Another restriction of the model is the methods available to multicast. The LAPOB is able to address messages using either a *contiguous interval* or *regularly spaced* addressing pattern. (Refer to Figure 3.1.) Although a processor of the LARPBS is able to arbitrarily set its select pulses, each of the algorithms presented in this work uses only the interval multicasting pattern.

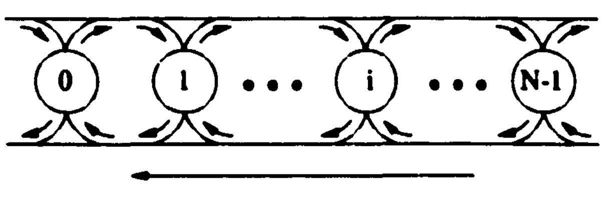


Figure 3.2: Linear Array of Processors with Pipelined Buses (APPB)

A simpler optical model is the linear *Array of Processors with Pipelined Buses* (APPB) [24]. Each processor connects to two buses by two couplers, one for transmitting and the other for receiving (Figure 3.2). Unlike the LARPBS, processors transmit messages to and receive messages from the same bus segment. Extending this model to two-dimensions, each processor connects to four buses. The *Array of Processors with Pipelined Buses using Switches* (APPBS) is a further extension. The APPBS uses switches to connect row and column buses and allow messages to pass directly between buses. The switches also provide the model with the ability to reconfigure itself, similar to the PR-Mesh. Section 5.2 discusses the APPBS in more detail and presents simulations that relate it more closely to the PR-Mesh.

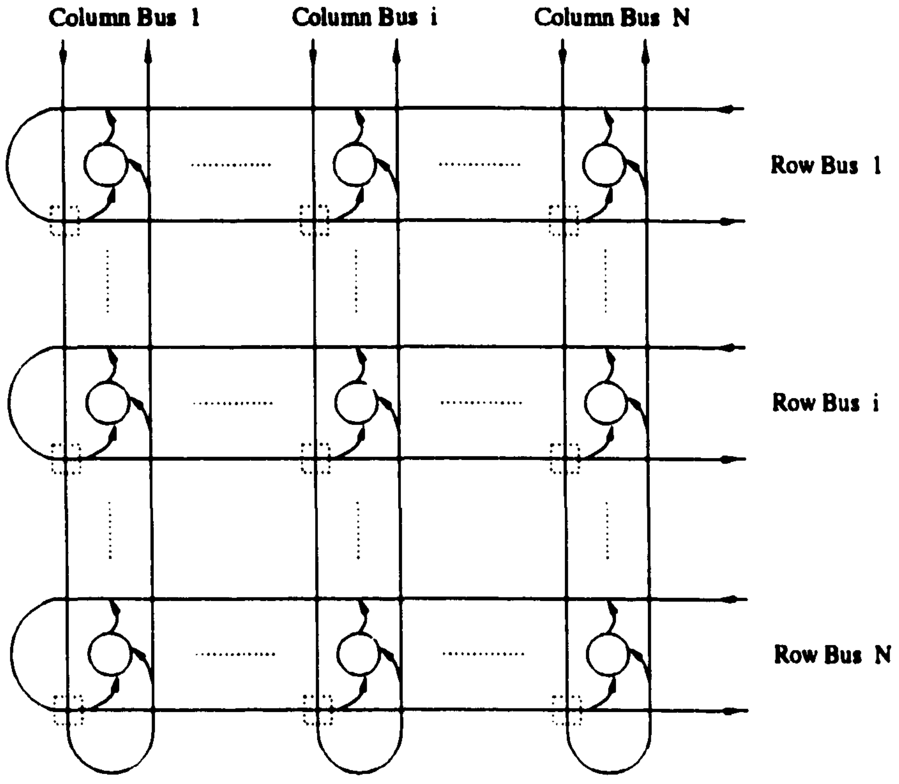


Figure 3.3: [66] Array structure with Synchronous Optical Switches (ASOS)

The *Array structure with Synchronous Optical Switches* (ASOS) [66] is another two-dimensional model that uses switches to connect row and column buses. Each processor is able to transmit on the upper segment of a row bus and receive from the lower segment of a row bus and the right segment of a column bus (Figure 3.3). The switches control the route a message takes. A switch set in the cross state causes messages to transfer from a row bus to a column bus.

The *Linear Array with Reconfigurable Optical Buses* (LAROB) [61, 62, 63] is similar to the LARPBS with extra hardware features. Each processor has switches that allow it to introduce up to N unit delays, unlike the one conditional delay of the LARPBS. Each processor also has a relative delay counter and an internal timing circuit to output a message during any petit cycle. An optical rotate-shift register and a counter are also present at each processor to assist in performing a bit polling operation. Pavel and Akl presented an extended version of the LAROB that is able to change switch settings within a bus cycle. They also presented a two-dimensional version of the LAROB called the *Array with Reconfigurable Optical Buses* (AROB).

These extra features not possessed by the other optical models seem to suggest that the LAROB (AROB) has more "power." Section 5.3 proves that the AROB has the same complexity as the PR-Mesh, that is, both are able to solve the same problems in the same number of steps with a polynomial increase in the number of processors.

3.2 Algorithm Overview

Often, algorithms designed for pipelined optical models follow the approach of R-Mesh algorithms, but additionally exploit the ability to pipeline messages, multicast, and broadcast during a single step. This results in more efficient algorithms since multiple buses are not needed to transfer multiple messages concurrently. To demonstrate this, we present existing algorithms in this section for optical models in the areas of sorting and selection, image analysis, and PRAM simulations.

3.2.1 Sorting and Selection Algorithms

Sorting and selection are basic operations finding use in many applications and have therefore been studied extensively. In this section, we sketch a variety of algorithms for sorting and selection.

ElGindy presented an $O(\log N \log\log N)$ step algorithm to sort N values on an N-processor LAPOB [18]. The algorithm uses a two-way merge sort in which there are $O(\log N)$ iterations of merges. Each successive merge is between larger pairs of sorted subsequences achieved by a multi-way divide-and-conquer strategy. The merge procedure executes in $\log\log N$ recursive steps of partitioning the input sequences into subsequences that will then be merged in parallel on disjoint sets of processors. This algorithm can also be implemented on the LARPBS as well as some of the other one-dimensional optical arrays discussed.

The algorithm takes advantage of the pipelining ability of the LAPOB. This enables multiple merge operations to be executed in parallel on a single bus.

Theorem 3.1 *An N-processor LARPBS can sort N values in $O(\log N \log\log N)$ steps.*

Rajasekaran and Sahni designed an optimal algorithm to sort N elements in $O(1)$ steps using an $N^{\epsilon} \times N$ AROB, where ϵ is any constant greater than zero [68]. This algorithm is optimal due to the lower bound of $\Omega(N^{1+\epsilon})$ processors for a comparison sort [3]. Rajasekaran and Sahni followed the column sorting algorithm of Leighton [37], which assumes the elements are stored as a matrix of size $N^{2/3} \times N^{1/3}$. The algorithm consists of a constant number of column sorts and matrix transpositions. The transposition operations are basically permutation operations that the AROB can route in a single step by pipelining messages. The AROB performs column sort as follows.

First assume that an $N^{2/3} \times N$ AROB is available, then we will extend it for any $\epsilon > 0$. This provides an $N^{2/3} \times N^{2/3}$ subarray to sort each column of $N^{2/3}$ elements. Sort the elements of each subarray in $O(1)$ steps using the R-Mesh algorithm to sort N elements on an $N \times N$ R-Mesh in $O(1)$ steps [49]. This is possible due to the ability to broadcast along a bus in a single step. In order to reduce the size of the AROB for any $\epsilon > 0$, recursively apply the sorting method for sorting columns for a total of $O(1)$ steps. This algorithm also runs on an $N^{\epsilon} \times N$ PR-Mesh.

Theorem 3.2 *An $N^{\epsilon} \times N$ AROB can sort N values in $O(1)$ steps, for constant $\epsilon > 0$.*

Integer sorting is a special case of sorting, and is usually performed by a series of radix sorts and compressions. This approach for sorting N k-bit integers takes $O(k)$ steps on an N-processor LARPBS [56]. Pavel and Akl presented an algorithm that runs in $O(\frac{k}{\log\log N})$ steps on an N-processor LAROB [62]. It takes advantage of the LAROB's bit polling operation and its ability to inject multiple delays onto the select waveguide. We will first describe the method for $k = O(\log\log N)$ bits and then extend it for $k = O(\log N)$ bits.

Each processor holds a value v_i, where $0 \leq i < \log N$. First, each processor p_i determines the number of processors p_j with $v_i = v_j$ and $i < j$ by using the bit polling operation. It then determines the total number of processors with the same value. The LAROB then uses the integer prefix sums algorithm to rank the elements and determine the final destinations [62]. The prefix sums algorithm is similar to the binary prefix sums algorithm of the LARPBS, however, a processor is able to introduce multiple delays to correspond to value v_i. Lastly, route each element to its sorted position.

This algorithm stably sorts N integers with value $0 \leq v_i < \log N$ in $O(1)$ steps on an N-processor LAROB. To extend the range of values, divide the k bits in $\frac{k}{\log\log N}$ groups, each of $\log\log N$ bits. The LAROB performs the sorting algorithm in $\frac{k}{\log\log N}$ stages. During stage i, stably sort the values with respect to the i^{th} least significant group of bits in $O(1)$ steps as above.

Theorem 3.3 *An N-processor LAROB can sort N k-bit values in $O(\frac{k}{\log\log N})$ steps.*

The problem of *selection* is to select the k^{th} smallest element out of N given elements. Li and Zheng designed a selection algorithm that runs in $O(\log N)$ time on an N-processor POB [43]. The algorithm exploits the multicasting ability of the POB. It is recursive and proceeds as follows.

Let P denote the set of active processors; initially $|P| = N$. (The base case is when $|P| \leq 5$.) Partition P into groups of five contiguous processors each. In $O(1)$ steps, the tail of each group determines the median of its group. The POB compresses the $\lceil\frac{|P|}{5}\rceil$ determined medians to the $\lceil\frac{|P|}{5}\rceil$ leftmost processors. Recursively find the median of these $\lceil\frac{|P|}{5}\rceil$ values. Denote this value as m. The leftmost processor broadcasts m and the POB computes prefix sums to count the number s of elements that are less than or equal to m. If $s = k$, then return m. If $s > k$ ($s < k$), then compress the elements less than or equal to (greater than) m and recursively call the select procedure on the s ($|P| - s$) elements. This algorithm also runs on an LARPBS.

Theorem 3.4 *The k^{th} smallest element can be selected from N elements by an N-processor LARPBS in $O(\log N)$ steps.*

Rajasekaran and Sahni designed a randomized algorithm to perform selection on a $\sqrt{N} \times \sqrt{N}$ AROB in $O(1)$ steps with high probability (w.h.p.) [68]. (High probability is a probability $\geq (1 - n^{-\alpha})$ for any constant $\alpha \geq 1$.) The algorithm takes advantage of the constant time compression operation and sorting on an AROB. The algorithm first picks a random sample S of size $q = o(N)$. The AROB compresses the sample elements in the first row of the AROB and then sorts the sample. Next, choose two elements l_1 and l_2 from the sample whose ranks in S are $k\frac{q}{N} - \delta$ and $k\frac{q}{N} + \delta$ for some δ, where $\delta = f(N)$. These two elements bound the element to be selected w.h.p. Eliminate all elements outside of the range $[l_1, l_2]$. Repeat the process again for the remaining elements. The number of iterations required is less than four w.h.p.

Theorem 3.5 *The k^{th} smallest element can be selected from N elements by a $\sqrt{N} \times \sqrt{N}$ AROB in $O(1)$ steps w.h.p.*

3.2.2 Image Analysis Algorithms

A few different image analysis algorithms have been designed for the optical models discussed. In particular we will consider algorithms to compute the Hough transform of an image and the nearest neighbor. Section 6.1.3 focuses on improving the efficiency of other image processing algorithms that have been developed for the R-Mesh.

The *Hough transform* is a method to detect the shape of object boundaries in a binary image by obtaining a set of projections of the image from different angles. The image is integrated along line contours defined by the set of points (x, y) satisfying the equation

$$x\cos(\theta) + y\sin(\theta) = \rho,$$

where θ is the angle of the line with respect to the positive y-axis and ρ is the distance of the line from the origin.

Pan and Li [56] developed an algorithm to perform the Hough transform on a $\sqrt{N} \times \sqrt{N}$ binary image in $O(N \log N)$ steps on an N-processor LARPBS. The algorithm takes advantage of the segmenting ability of the LARPBS to perform multiple prefix sums in parallel. Each processor holds the indices of a pixel of an image and the pixel value. There are N projections that are calculated, or N angle values θ_i, $0 \leq i < N$. The Hough transform maps collinear edge pixels into the same point in the parameter space. The parameter space is grouped into N θ values and N ρ values, where a (θ, ρ) pair corresponds to a linear band of edge pixels, approximating a line. As a result, it suffices to detect a point in the parameter space to which a large number of edge pixels are mapped.

Processors that hold an edge pixel perform the following steps for each angle value. First, each processor calculates the value of ρ using the above equation, ρ_i, $0 \leq i < N$. The LARPBS then sorts the N ρ values in $O(\log N)$ steps [56]. Segment the LARPBS so that each subarray holds the same ρ values and perform a binary sum operation over each subarray in $O(1)$ steps to determine the number of pixels that are mapped to the same point. The LARPBS then applies a threshold function to the summed values. Since there are N iterations (one for each angle value), the algorithm runs in $O(N \log N)$ steps.

Theorem 3.6 *The Hough transform of a $\sqrt{N} \times \sqrt{N}$ binary image can be computed in $O(N \log N)$ steps on an N-processor LARPBS.*

Pavel and Akl [64] also developed an algorithm to compute the Hough transform of an $N \times N$ image in $O(1)$ steps on an $N \times N \times N$ AROB. Their algorithm exploits the AROB's ability to reconfigure its buses at each step.

The *nearest neighbor* problem considers an $N \times N$ binary image $A = (a_{i,j})$, $0 \leq i, j < N$, where each element is either a black ($a_{i,j} = 1$) or white ($a_{i,j} = 0$) pixel. Let $B \subseteq A$ be the subset of black pixels. The Euclidean distance $dist(a_{i,j}, a_{i',j'})$ between two pixels $a_{i,j}$ and $a_{i',j'}$ is given by

$$dist(a_{i,j}, a_{i',j'}) = ((i - i')^2 + (j - j')^2)^{1/2}.$$

A black pixel $a_{i',j'}$ is a nearest neighbor of $a_{i,j}$ if the distance between the two pixels is minimum with respect to $a_{i,j}$ and $B - \{a_{i,j}\}$.

Pan *et al.* [57] presented an algorithm to compute the nearest neighbor in $O(\log\log N)$ steps using an $N^{2+\epsilon}$-processor LARPBS in which the image is stored in row major order. They proceeded by partitioning the image A into two regions for each black pixel $a_{i,j}$. The left region of $a_{i,j}$ contains the pixels in all columns j' such that $j' \leq j$. They defined the right region similarly. The algorithm then finds the nearest neighbor in each region and selects the closer of the two.

Find the nearest left neighbor as follows. First find the nearest black pixel in the same column and row in a constant number of steps by performing segmented broadcasts and row transformations. Then each processor performs a series of local computations using the information found. Next, by pipelining messages, all processors holding a black pixel send their distance from $a_{i,j}$ to the right within its row. One can view each row as a series of segments separated by black pixels, each of which acts as the head of its segment. Find the minimum distance value within each segment in $O(\log\log N)$ steps [56]. Determine the minimum of the minimums and this is the nearest neighbor.

Theorem 3.7 *The nearest neighbor problem of an $N \times N$ image can be performed in $O(\log\log N)$ steps on an N^2-processor LARPBS.*

3.2.3 PRAM Simulations

An (N, M)-PRAM is a shared memory model that consists of N processors and M memory locations. The processors are able to read from and write to any of the shared memory locations. The read and write operations to a single memory location can either be concurrent or restricted to be exclusive to one processor at a time. Simulations of both *Exclusive Read Exclusive Write* (EREW) and *Concurrent Read Concurrent Write* (CRCW) PRAMs have been developed for some optical models. In this section we present two of these simulations of the more powerful CRCW PRAM.

The first result is a simulation of an N-processor CRCW PRAM with $O(N)$ memory locations by an N-processor LARPBS in $O(\log N)$ steps [41]. The simulation takes advantage of an N-processor EREW PRAM with $O(N+M)$ memory locations being able to simulate an N-processor PRIORITY CRCW PRAM computation with M memory locations in $O(\log N)$ steps [28]. Using this result, the LARPBS proceeds in simulating an N-processor EREW PRAM in $O(1)$ steps as follows.

First assume that the EREW has $M = N$ shared memory locations. Let processor R_i of the LARPBS simulate PRAM processor P_i and hold memory location M_i. The LARPBS simulates a read step of the PRAM, where P_j reads from M_k, in two steps. In the first step, R_j sends its index to R_k, then in the second step, R_k sends the value of M_k to R_j. The LARPBS simulates a write step of the PRAM, where P_j writes value v_j into M_k, in a single step. Processor R_j sends v_j to R_k and R_k stores this value. Since each step is an exclusive read or write step, the indices sent are all distinct and there are no conflicts. For the case when $M = O(N)$, there exists a constant c such that $M = cN$. In order to accommodate this, each processor of the LARPBS holds c memory locations and then simulates the read and write steps in c iterations. Combining the results provides the following theorem.

Theorem 3.8 *Each step of an N-processor* PRIORITY *CRCW PRAM with $O(N)$ shared memory locations can be simulated by an N-processor LARPBS in $O(\log N)$ steps.*

The simulation presented by Pavel and Akl [63] is a randomized algorithm for the two-dimensional APPB model. They proceeded by first showing that a two-dimensional APPB with N processors can simulate any N-processor network, G, with constant degree in $O(1)$ steps. Map the processors of G to the APPB, however, the neighbors of a processor of G may not be neighbors in the APPB. To perform neighboring communications, construct a bipartite graph of G with k edges representing neighbor edges. From this, using k permutation routings, the APPB can simulate any communication step. This result implies that an N-processor APPB is able to simulate an N-processor butterfly network in $O(1)$ steps. Using Ranade's result [69] that an N-processor butterfly network with $O(M)$ memory can simulate a step of a CRCW (N, M)-PRAM in $O(\log N)$ steps w.h.p. provides the following result.

Theorem 3.9 *Each step of an N-processor CRCW PRAM can be simulated by a $\sqrt{N} \times \sqrt{N}$ APPB in $O(\log N)$ steps w.h.p.*

The algorithms presented in this chapter are a small sample of the algorithms that have been developed for optically pipelined models. They demonstrate the key techniques used by most of these models. It is not always clear, however, which algorithms can run on which models, besides the one for which the algorithm was developed. For this reason, we unify three of the one-dimensional models in the next chapter. The differences between the two-dimensional models make it unclear how they relate to each other. We relate three of these models to each other and to the

LR-Mesh and establish that they possess the same computational complexity. This provides a better understanding of the power of these models.

The range of algorithms that have been developed is limited, in the sense that only healthy systems are considered. The information provided is useful, however, the algorithms are of no use if one or more processors are faulty. We, therefore, consider faulty systems and algorithms that are able to accommodate faults in Chapter 7.

Chapter 4

Relating One-Dimensional Optical Models

The introduction listed several similar models with "optically pipelined buses." Many of these models have different features, making it difficult to relate results from one model to another. It is a useful endeavor, therefore, to unify these models in order to increase understanding of which features are essential and to be able to translate algorithms from one model to another. In this chapter we establish the equivalence of three one-dimensional optical models, namely the LARPBS, *Linear Pipelined Bus* (LPB) [54], and *Pipelined Optical Bus* (POB) [42, 79]. This implies an automatic translation of algorithms (without loss of speed or efficiency) among these models. In other words, any algorithm proposed for one of these models can be implemented on any of the others with the same number of processors and to within a constant factor of the same time (Theorem 4.5 in Section 4.2.2).

The only difference between the LARPBS and LPB is the segmenting ability of the former. The segmenting ability of the LARPBS simplifies algorithm design, yet, due to the equivalence of these models, it is not necessary to include the segment switches. Moreover, this equivalence establishes dynamically selectable delay loops (that are a part of each of the models considered in this chapter) as the key to the power of these models. This separation of the powers of segmentation and delays is similar to that established in the context of the RMBM [74].

Section 4.1 describes the structure of the LPB and POB models. Section 4.2 establishes the equivalence of the three optical models by constructing a cycle of simulations among these models.

4.1 Model Descriptions

The *Linear Pipelined Bus* (LPB) [54] is identical to the LARPBS with the exception that it does not have any segment switches. Therefore, the LPB is not able to segment its bus.

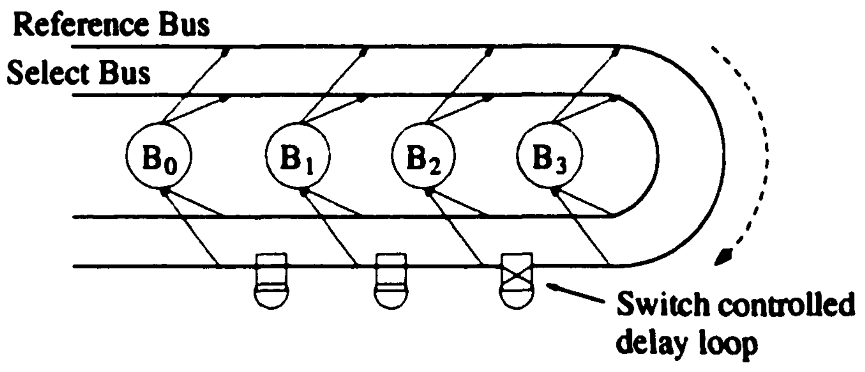


Figure 4.1: Structure of a POB

The *Pipelined Optical Bus* (POB), proposed by Li and Zheng [42, 79], is a similar model. Like the LARPBS and LPB, the POB has three waveguides. Conditional delay switches, however, are positioned on the receiving side of the reference and data waveguides, rather than on the transmitting side of the select line (see Figure 4.1). The POB contains no fixed delay loops, so the length of the bus cycle is actually shorter than that of the LARPBS and the LPB. As the POB contains no segment switches, segmenting is not possible.

The POB also uses the coincident pulse technique to route messages. The effect of conditional delay switches on the POB is to delay the reference pulse relative to the select frame, so the POB is also able to perform one-to-one addressing, multicasting, and broadcasting. The location of the conditional delay switches on the receiving end enables the POB to multicast and broadcast without having to set multiple select pulses in a select frame, although multiple select pulses could be set as in the LARPBS and LPB. Consider the case when processor B_i is the selected destination, the delay switch between B_i and B_{i-1} is straight, and all remaining delay switches are set to cross. The select and reference pulses will coincide at B_i and again at B_{i-1}, therefore both processors receive the message although only one select pulse was injected.

We now demonstrate the addressing of the POB by referring to the switch settings as shown in Figure 4.1. Suppose processor B_1 injects a select pulse so that B_3 is its selected destination, and B_0 injects a pulse so that B_2 is its selected destination. The settings of the straight switches will result in a multicast operation by B_0 to actual destinations B_2, B_1, and B_0. The actual destination of the message sent by B_1 is B_3. The normal state of operation for the POB is when all conditional delay switches are set to cross.

Throughout this chapter R_i, L_i, and B_i refer to the i^{th} processor of an LARPBS, LPB, and POB, respectively.

4.2 Equivalence of the LARPBS, LPB, and POB

In this section, we prove that the LARPBS, LPB, and POB are equivalent. That is, each model can simulate a step of either of the two other models in constant time, using the same number of processors. In our simulation of a model with segmenting by a model without segmenting, computing the prefix sums of N bits will play a key role. To this end, we now present a new algorithm to compute the prefix sums of N bits in a constant number of steps that uses the multicasting ability of the models, rather than the segmenting ability of the LARPBS. We will use the example provided in Figure 4.2 to assist with the explanation.

4.2.1 Computing Prefix Sums without Segmenting

Lemma 4.1 *The prefix sums of N bits can be computed by an N processor LPB in $O(1)$ steps.*

Proof: Consider an LPB with N processors, such that each one holds a binary value v_i, for $0 \leq i < N$. The i^{th} prefix sum, $psum_i$, is $v_0 + v_1 + \ldots + v_i$. Let the i^{th} "reverse prefix sum" be $rpsum_i = v_{i+1} + v_{i+2} + \ldots + v_{N-1}$, for $0 \leq i < N-1$, and $rpsum_{N-1} = 0$.

First, each processor L_i sets its conditional delay switch to straight if $v_i = 0$ and to cross if $v_i = 1$. Referring to Figure 4.2(a), L_1, L_5, and L_7 each hold a value of '1' and set their conditional delays to cross. Next, each processor injects a reference and a select pulse at the same time, selecting destination L_{N-1}, and sends its own ID as data. The switch settings introduce delays on the select line corresponding to the 1 bits. Consider processor L_i. If the resulting $rpsum_i$ is m, then m switches to the right of L_i are set to cross, and the two pulses from L_i will coincide at $L_k = L_{N-1-m}$. Some processor receives the message originating from L_i iff either $v_{i+1} = 1$ or $i = N-1$. Note that if a processor's message is disregarded, then all processors between it and the closest processor to its right, L_j, whose message is accepted pass through the same number of conditional delays and arrive at the same destination because they contain a value of 0. Also, $rpsum_i = rpsum_j$ because adding the zeros from the processors

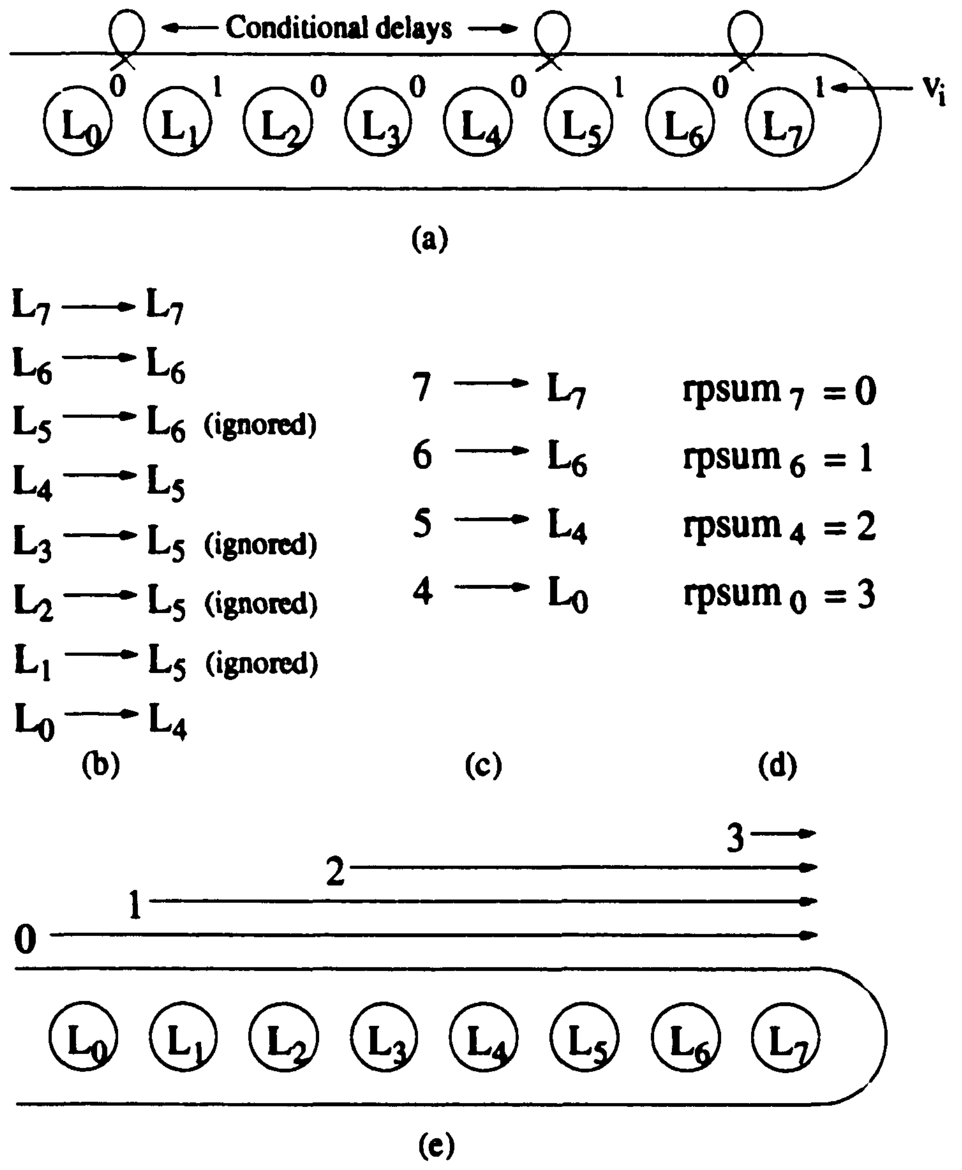


Figure 4.2: Binary prefix sums example without segmenting: (a) input values and switch settings; (b) actual destinations of first set of messages; (c) response to first set of messages; (d) *rpsum*$_i$ values; (e) multicasting step.

between L_i and L_j to the summation does not alter the result. (Figure 4.2(b) shows which messages are accepted and disregarded for the example.) Next, each processor that received a message sends its own address to the original sender, L_i, which stores $rpsum_i$. If this address is $N-1-k$, then the message was delayed by k slots, so $rpsum_i = k$. (Figure 4.2(c) shows the response messages and Figure 4.2(d) shows the $rpsum_i$ values for these processors.)

Set all conditional delay switches to straight (the normal state of operation) for the remainder of the algorithm. Since not all messages in the first step may have been accepted, some processors may not have received an *rpsum* message in the previous step. The LPB next sends *rpsum* values to these processors. Let S_r denote the set of processors that received an *rpsum* message. For each $L_i \in S_r$, we want to send $rpsum_i$ to $L_h, L_{h+1}, \ldots, L_{i-1}$ such that $v_{h+1} = v_{h+2} = \ldots = v_i = 0$, as $rpsum_i$ is equal to their *rpsum* values. To accomplish this, we exploit the feature that a processor receives the first of multiple messages sent to it. Processor L_{N-1-i} substitutes for L_i, reversing the order of the processors. For each $L_i \in S_r$, L_{N-1-i} now multicasts $rpsum_i$ to processors $L_{N-1-i}, L_{N-i}, \ldots, L_{N-1}$. Each L_k, where $0 \leq k < N$, will accept exactly one message and store it as $rpsum_{N-1-k}$. If $L_k \in S_r$, then the message accepted by L_{N-1-k} will be from itself, otherwise the message originated from the closest processor L_{N-1-g} to its left such that $L_g \in S_r$. (Refer to Figure 4.2(e) to see which processors multicast the $rpsum_i$ values and the values sent.) Now processor L_{N-1-i} sends $rpsum_i$ to L_i which stores the data as $rpsum_i$ to reverse the order of the values back to the original order. Each processor L_i now has $rpsum_i = v_{i+1} + v_{i+2} + \ldots + v_{N-1}$. The total sum is $v_0 + rpsum_0$, which L_0 broadcasts to all processors, enabling each processor L_i to calculate the correct prefix sum $psum_i = (totalsum) - (rpsum_i) = v_0 + v_1 + \ldots + v_i$.

Each phase of the algorithm runs in a constant number of steps. Based on this algorithm, computing the binary prefix sums of N bits on an LPB can be performed in $O(1)$ steps. ■

4.2.2 Equivalence of Optical Models

We will make use of the binary prefix sums algorithm presented above to show the equivalence of the LARPBS, LPB, and POB. For a more detailed discussion on the equivalence of models, see Trahan *et al.* [72]. We prove the equivalence of the three optical models by a cycle of simulations. Each simulation consists of the following three phases: (i) determine parameters for the actual destinations of all messages, (ii) create the select frames, and (iii) send the messages.

Lemma 4.2 *Each step of an N processor LARPBS can be simulated by an N processor LPB in $O(1)$ steps.*

Proof: **Find parameters for actual destinations:** First, each processor L_j of the LPB identifies the nearest segment switch that is set in the LARPBS to the left of its position. If L_i simulates a processor with a set segment switch, then L_i multicasts $i+1$ to $L_{i+1}, L_{i+2}, \ldots, L_{N-1}$, and L_j stores this as $left_j$. More than one message may coincide at a single processor, however, the first one received identifies the lowest indexed processor that is in the same subarray as L_j. If a processor did not receive a message, it will assume the lowest indexed processor within its subarray to be L_0. To identify the nearest set segment switch to the right, reverse the order of the processors, letting processor L_{N-1-j} substitute for L_j, and then proceed the same as before. If a processor did not receive a message, it assumes the highest indexed processor in its subarray to be L_{N-1}. Each L_j stores the index of the rightmost processor in its subarray as $right_j$.

Next, each processor L_j determines the number of set conditional delay switches to the right of processor R_j of the LARPBS in its subarray (that is, between processors indexed j to $right_j$). To do so, the LPB computes the binary prefix sums of the number of set switches (Lemma 4.1). Each processor L_j then refines its prefix sum based upon the prefix sum of processor $right_j$ and stores it as $psum_j$.

Create select frames: Given the location of the select pulses within the select frame (selected destinations), the information on set segment switches, and the number of set conditional delay switches, L_j locally determines the actual destination processors for its message as follows. Processor L_j shifts its select pulse(s) by $(right_j - N + 1 - psum_j)$ to match the actual destinations. If some of the resulting select pulses correspond to processors that are not within its subarray, then L_j uses $left_j$ to mask off the bits for those select pulses.

Send messages: At this point, processors set all delay switches to straight and transmit their messages. If a message was to be received by R_i in the LARPBS, then L_i successfully receives it in the LPB. A message sent by a processor of the LARPBS to multiple destinations would be sent to the corresponding processors of the LPB. Also, if multiple messages arrive at one processor in the LARPBS, then the simulating LPB maintains their order of arrival so that the processor receives the proper message. Therefore, the simulation also properly handles any concurrent-read or concurrent-write step of the LARPBS. ∎

Though neither the LPB nor the POB can segment its buses, the simulation of an LPB on a POB is not automatic due to differences in the location of conditional delay switches, normal state of operation, and methods of multicasting. For instance, if processor L_j of the LPB sets its conditional delay switch to cross to introduce a delay, then messages originating from L_i, $0 \leq i \leq j$, will be shifted. If processor B_j

of the POB sets its delay switch to straight, however, then messages destined for B_i, $0 \le i \le j$, will be shifted. The proof of the lemma below addresses these issues.

Lemma 4.3 *Each step of an N processor LPB can be simulated by an N processor POB in $O(1)$ steps.*

Proof: **Find parameters for actual destinations:** The POB first determines the number of conditional delay switches set to cross to the right of each processor on the LPB (using binary prefix sums [42]). Each processor B_i stores the prefix sum it calculated as $psum_i$. If L_j is a selected destination for the message sent by L_i, then the message will arrive at actual destination with index $(j - psum_i)$ on the LPB.

Create select frames: Based on the prefix sum values, each processor can shift and mask its select frame, as in the proof of Lemma 4.2, placing select pulses according to the actual destinations.

Send messages: After adjusting the select pulses, set all delay switches to cross on the POB and send the messages. This is the normal state of operation for the POB, so no messages will be shifted in this step. If a message was to be received by L_i in the LPB, then B_i successfully receives it in the POB. As in the proof of Lemma 4.2, this simulation properly handles any concurrent-read or concurrent-write step. ■

For an LARPBS to simulate a POB, the differences mentioned before the previous lemma pose a problem, even though the LARPBS can segment its buses and the POB cannot. In particular, one select pulse in the LARPBS can address only one processor, while the POB can address multiple processors with one select pulse by setting successive conditional delay switches to straight. To overcome these differences, the LARPBS sends messages to intermediate destinations as described in the following proof.

Lemma 4.4 *Each step of an N processor POB can be simulated by an N processor LARPBS in $O(1)$ steps.*

Proof: **Find parameters for actual destinations:** To simulate the POB on the LARPBS, we first determine the number of conditional delay switches on the receiving side that are set to straight before each of the processors. Recall that a delay switch set to straight shifts messages on the POB (and may cause multiple processors to receive the same message), so this will provide information for the actual destinations of the messages. Each processor R_i of the LARPBS calculates the binary prefix sum, $psum_i$, based on the number of straight switches.

The number of straight switches preceding the processor simulated by R_i on the receiving side is $d_i = psum_{N-1} - psum_i$. If a message was to be sent to selected destination B_i on the POB, then it would actually arrive at B_k, such that $k + d_k = i$. Also, if the computed value $k + d_k$ is the same for multiple processors, then these processors would receive the same message, corresponding to a concurrent-read step of the POB. Note that a select and a reference pulse in a frame may not coincide at any processor in the POB if enough conditional delay switches are set to straight. In this case, there will be no nonnegative k to satisfy the previous equation.

Create (partial) select frames and send messages: Send messages in the normal state (all conditional delay switches set straight) on the LARPBS without altering the select frames. Next, R_j sends a message containing its ID to the processor indexed $(j + d_j)$ requesting the data that processor $(j + d_j)$ received. This is because the message R_j would have received after being shifted by d_j in the POB was actually received by processor $(j + d_j)$ in the LARPBS without being shifted. Processor $(j + d_j)$ might be the destination of multiple such requests, corresponding to multiple contiguous processors that should receive copies of the message processor $(j + d_j)$

holds. This occurs when the multiple processors should receive the same message on the POB due to straight conditional delay switches. Processor $(j + d_j)$ then sends the data it originally received to the processor whose request it received in the previous step. Each processor R_i of the LARPBS then sets its segment switch if processor B_{i+1} has its delay switch set to cross in the simulated model. A crossed delay switch represents the boundary for which contiguous processors would receive the same message due to straight delay switches. The head of each subarray now broadcasts the data it received in the last step. Each processor R_i in the LARPBS now has the same message as B_i would in the POB. Also, the LARPBS properly handles any concurrent-read or concurrent-write step of the POB. ■

The cycle of simulations described by the preceding lemmas establishes the equivalence of these models.

Theorem 4.5 *The LARPBS, LPB, and POB are equivalent models. Each one can simulate any step of one of the other models in $O(1)$ steps with the same number of processors.*

Chapter 5

Relating Two-Dimensional Optical Models

We have listed a number of models that utilize "optically pipelined buses" in Chapter 1. In this chapter we will concentrate on the two-dimensional version of the LARPBS, the *Pipelined Reconfigurable Mesh* (PR-Mesh) [72]. Other proposed, similar two-dimensional optical models are the *Array with Reconfigurable Optical Buses* (AROB) [62, 63], the *Array Processors with Pipelined Buses* (APPB) [47], the *Array Processors with Pipelined Buses using Switches* (APPBS) [24], the *Array with Synchronous Optical Switches* (ASOS) [66], and the *Reconfigurable Array with Spanning Optical Buses* (RASOB) [65].

Many of the optically pipelined models have different features, making it difficult to relate results across models. It is a useful endeavor, therefore, to unify these models in order to increase understanding of which features are essential and to be able to translate algorithms from one model to another. In Chapter 4, we determined the equivalence of three one-dimensional reconfigurable optical models: the LARPBS, LPB, and POB. This result implies an automatic translation of algorithms (without loss of speed or efficiency) among these models. In this chapter we consider two-dimensional models. This presents obstacles not present when analyzing linear arrays, such as the larger number of configurations possible due to the multiple dimensions. To account for this, we establish their equivalence in a slightly different context; here we consider their complexity by relating their time to within a constant factor and the number of processors to within a polynomial factor. Two models have the same complexity if either model can simulate any step of the other model in a constant number of steps, with up to a polynomial increase in the number of processors.

Given the number of algorithms developed on reconfigurable models and the growing body of research on them, it is important to relate these models to each other and to other, more widely known models. In this chapter we prove that the PR-Mesh has the same complexity as the cycle-free *Linear Reconfigurable Network* (LR-Mesh), that is, in constant time using a polynomial number of processors, the PR-Mesh and the cycle-free LR-Mesh can solve the same class of problems. We also show that these models have the same complexity as the LR-Mesh that allows cycles (Section 5.1). We extend this complexity class to include two other optical models, namely the AROB and APPBS. Section 5.2 relates the APPBS and the PR-Mesh, then Section 5.3 relates the AROB and the PR-Mesh. Our results obtained in this chapter are some of the first to unify reconfigurable optical models to each other and relate them to other more widely known models.

We will first define some terminology prior to presenting the results. We draw on the complexity class definitions in this section from Johnson [30] and Karp and Ramachandran [31]. Let N denote the input size.

For model Z, let $Z(T, \text{poly}(N))$ denote the class of languages accepted by model Z in $O(T)$ steps with polynomial in N processors. The class L is the class of languages accepted by deterministic Turing machines with work space bounded by $\log N$. This class is contained inside P and the corresponding algorithms use less workspace than the size of their input [30]. For example, a problem in L is one that can be solved in a reasonable amount of time by a polynomial number of computers.

5.1 Complexity of the PR-Mesh

The *Linear Reconfigurable Network* (LR-Mesh) [5] has the same structure as the PR-Mesh and each processor can locally configure its port connections as in a PR-Mesh (Figure 2.8). The difference is that it uses electronic buses instead of optical buses. Thus, it is not able to pipeline messages. A value written on a port reaches all ports connected to the same bus in one time step, however.

Due to the U-turn structure of the PR-Mesh buses, cycles are not allowed; it is necessary to separate the transmitting segment from the receiving segment. Therefore, the LR-Mesh model that we will first relate to the PR-Mesh is one that is *cycle-free*, that is, all buses are linear and without cycles. Refer to this model as the *cycle-free LR-Mesh* (CF-LR-Mesh). We will first establish that $L = \textit{CF-LR-Mesh}(1, \text{poly}(N))$, thereby indirectly relating the complexity of the CF-LR-Mesh to that of the LR-Mesh. We will then establish in Section 5.1.2 that the PR-Mesh has the same complexity as the CF-LR-Mesh and can thus solve any problem in L in constant time using a polynomial number of processors.

5.1.1 Relating the LR-Mesh and CF-LR-Mesh

Ben-Asher *et al.* [5] established $L = \textit{LR-Mesh}(1, \text{poly}(N))$ using an LR-Mesh that allows cycles. They used the decision problem CYCLE, which is complete for L with respect to NC^1 reductions. The class NC^1 consists of all languages recognizable by log-space uniform classes of Boolean circuits having polynomial size and depth $O(\log N)$. A reduction of a problem is a mapping of problem X to an instance of another problem Y, such that the solution to Y provides a solution to the instance of X [14]. An NC^1 reduction from problem X to problem Y is a log-space uniform family of Boolean circuits that

- solves X given Y,
- contains at most a polynomial number of gates, and
- has $O(\log N)$ depth.

This implies that any problem that the LR-Mesh can solve in constant time using a polynomial number of processors can be mapped to the problem CYCLE.

Definition 1 [5] CYCLE is the following decision problem. The input is a permutation on N vertices, that is, a directed graph of out-degree 1 and in-degree 1 (given by its adjacency matrix), with two special vertices u and v. The answer is '1' if u and v are on the same cycle.

To solve the CYCLE problem, Ben-Asher *et al.* devised the following algorithm. Let each processor of an $N \times N$ LR-Mesh hold one bit of the input adjacency matrix. Assume that vertex i maps to j and j maps to vertex k. After a series of communication steps, all processors in column j hold the IDs of predecessor i and successor k. Processors then create a linear bus between adjacent vertices. For instance, processors in column j and row k fuse their ports to create a bus from processor $p(j, j)$ (representing vertex j) along column j to $p(k, j)$, then along row k to processor $p(k, k)$. In this manner, each cycle in the input permutation induces a cycle in the LR-Mesh. Processor u writes a message on its cycle, and v receives the message if the two are on the same cycle.

This gives the following LR-Mesh solution to any problem Π in L: simulate the NC^1 circuit transforming the instance of Π to an instance of CYCLE, then solve the resulting instance of CYCLE. Ben-Asher *et al.* also developed a simulation of the NC^1 circuit (without the use of cycles), establishing $L \subseteq$ *LR-Mesh*$(1, \text{poly}(N))$. They further proved that *LR-Mesh*$(1, \text{poly}(N)) \subseteq L$, thereby obtaining $L =$ *LR-Mesh*$(1, \text{poly}(N))$.

We aim to prove that *CF-LR-Mesh*$(1, \text{poly}(N)) = L$. We use an $O(N) \times O(N) \times O(N)$ CF-LR-Mesh to solve CYCLE, and thus establish the same complexity. The approach we take is similar to that of Ben-Asher *et al.*, mapping the given adjacency matrix to the bottom layer $O(N) \times O(N)$ LR-Mesh and after a series of communication steps, all processors in the j^{th} column hold the IDs of the vertices immediately before and after vertex j in the permutation. Ben-Asher *et al.* actually embed the graph in an $O(N) \times O(N)$ LR-Mesh with the cycles. The CF-LR-Mesh, however, does not allow cycles. For this reason, we embed the permutation graph edges using the third dimension of the CF-LR-Mesh, as described below.

The LR-Mesh has N layers of $O(N) \times O(N)$ processors, where each layer can be broken down into 4×4 blocks of processors, as shown in Figure 5.1. Label eight of the processors within each block as "in" or "out" to represent the direction of the permutation mapping, although the CF-LR-Mesh is undirected. We will refer to these as ports or port processors for the block. (This labeling represents the direction of buses for the simulations involving optical buses in the sections to follow.) Let *block*(i, j) denote the block in the i^{th} row and j^{th} column of blocks, where $0 \leq i, j < N$. The blocks on the diagonal represent the vertices, for instance, *block*(i, i) represents vertex i.

We create linear buses, one bus corresponding to each vertex, such that the buses extend up the layers of the mesh. Bus connections are identical in each layer and

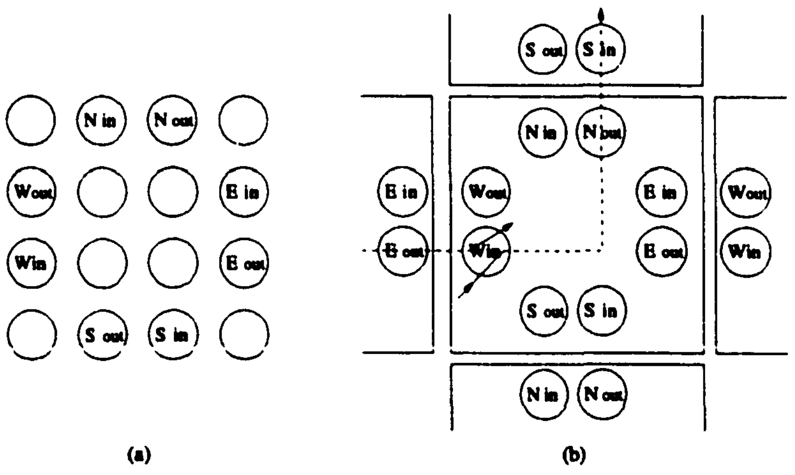


Figure 5.1: Block of 4 × 4 processors for simulations: (a) labeling of processors within blocks; (b) arrangement of blocks and connections.

depend on the permutation. For each vertex j with successor vertex k, in each layer, a bus connects $block(j,j)$ via $block(k,j)$ to $block(k,k)$ within the layer, then steps up to $block(k,k)$ in the next layer. This bus exits $block(j,j)$ from N_{out} or S_{out} and enters $block(k,k)$ from E_{in} or W_{in}, depending on the relative values of j and k. The "in" port also routes this connection up to $block(k,k)$ in the layer above. The bus coming from the layer below also enters at the same "in" port processor, and is configured to connect to the vertical bus leaving $block(k,k)$. Figure 5.1(b) shows the connections for a block whose predecessor reaches it via a block from its left, and successor corresponds to some row above. (Connections shown as dashed lines are all within the same layer. Connections shown as solid lines run either to the layer above or from the layer below.)

Consider a vertex u. The linear bus that starts at $block(u,u)$ in the bottom layer passes a block for each vertex reachable from u. Since a bus only moves up in layers of the CF-LR-Mesh, the bus from $block(u,u)$ may reach another copy of $block(u,u)$

in a later layer (because of a cycle in the permutation graph), but no cycle exists in the mesh.

To determine if vertices u and v are on the same cycle, let $block(u, u)$ in layer 0 write on its bus. If v is on a cycle with u, then $block(v, v)$ in some layer will receive the message from $block(u, u)$. Multiple blocks simulating v on different layers may receive the message. Each block simulating v sets its configuration to connect in a bus crossing all layers, but if it received the message from $block(u, u)$, then it disconnects from the layer above it and sends a message down the bus connecting it to the bottom layer. (Disconnecting the bus prevents concurrent writes.) If $block(v, v)$ receives this message on the bottom layer, then u and v are both on the same cycle, indicating a '1' answer to the CYCLE decision problem.

Therefore, we have the following result.

Theorem 5.1 *CF-LR-Mesh*$(1, \text{poly}(N)) = L$.

5.1.2 Relating the CF-LR-Mesh and PR-Mesh

We will use the result of the following lemma to show that the CF-LR-Mesh can simulate each step of a PR-Mesh in a constant number of steps with a polynomial increase in the number of processors.

Lemma 5.2 *Each step of an N processor LARPBS can be simulated in $O(1)$ steps by an $N \times N$ CF-LR-Mesh.*

Proof: Let $\pi_{i,j}$ (p_j), where $0 \leq i, j < N$, denote a processor of the CF-LR-Mesh (LARPBS). The CF-LR-Mesh computes prefix sums in constant time [52] on the set conditional delay switches to determine actual destinations. A similar computation determines the segment switch locations and the CF-LR-Mesh adjusts the actual destinations accordingly. Each processor $\pi_{0,j}$ sends its prefix sum, $psum_j$, and the select frame for p_j down column j. Each processor $\pi_{i,j}$ performs a bitwise AND between the select frame for p_j and 2^{i+pnum_j}. A nonzero result corresponds to coincident reference and select pulses at processor p_i of the LARPBS. To determine priority, a processor that detected coinciding pulses disconnects its ports and writes the message on its east port, while the remaining processors configure their ports as {N, S, EW}. Thus, the processors in the rightmost column receive the message that originated from the highest priority processor in $O(1)$ steps. ∎

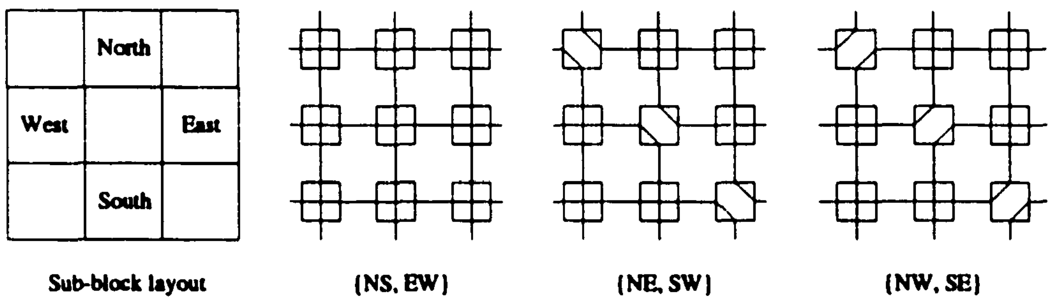


Figure 5.2: CF-LR-Mesh block configurations

Theorem 5.3 *PR-Mesh*$(\log^j N, \text{poly}(N))$ = *CF-LR-Mesh*$(\log^j N, \text{poly}(N))$.

Proof: A PR-Mesh can simulate each step of a CF-LR-Mesh in a constant number of steps, as it can configure its buses in the same manner and simply broadcast all messages [21, 63]. Therefore, *CF-LR-Mesh*$(\log^j N, \text{poly}(N)) \subseteq$ *PR-Mesh*$(\log^j N, \text{poly}(N))$.

Let $\mathcal{P}$ be an $N \times N$ PR-Mesh and let p_{ij} denote a processor of $\mathcal{P}$. We construct an $O(N^3) \times O(N^3)$ CF-LR-Mesh $\mathcal{L}$ that simulates each step of $\mathcal{P}$ in a constant number of steps. Partition $\mathcal{L}$ into $O(N^2) \times O(N^2)$ size blocks, each with nine sub-blocks of size $O(N^2) \times O(N^2)$ as shown in Figure 5.2. Number each block, B_{ij}, so that B_{ij} simulates p_{ij} of $\mathcal{P}$. Four of the sub-blocks correspond to the ports of p_i and the center sub-block is reserved for routing. All sub-blocks labeled "North" and "South" configure their ports as {NS,E,W} and those labeled "East" and "West" as {N,S,EW}. The center sub-block of B_{ij} sets processor connections according to the partition set by p_{ij} as shown in Figure 5.2. This forms the same linear buses as in the PR-Mesh.

The head of each bus sends its processor and port number as a bus id to label all ports on the bus. Rank the list of blocks along each bus starting at the head in constant time [52]. Next, transfer simulated processor p_{ij} to the rightmost column of the sub-block that matches its bus id and in the row that corresponds to its list ranking within the bus in $O(1)$ steps. Now each linear bus is in the rightmost column of its own $O(N^2) \times O(N^2)$ sub-block. Simulate one step of each such bus in $O(1)$ steps (Lemma 5.2) and then route simulated processors back to the proper blocks. Therefore, a CF-LR-Mesh of $O(N^3) \times O(N^3)$ size can simulate each step of an $N \times N$ PR-Mesh in $O(1)$ steps, so $\mathit{PR\text{-}Mesh}(\log^j N, \mathrm{poly}(N)) \subseteq \mathit{CF\text{-}LR\text{-}Mesh}(\log^j N, \mathrm{poly}(N))$.

Thus, $\mathit{PR\text{-}Mesh}(\log^j N, \mathrm{poly}(N)) = \mathit{CF\text{-}LR\text{-}Mesh}(\log^j N, \mathrm{poly}(N))$. ■

It is possible to reduce the number of processors required for this simulation to a $4N \times 4N \times N^2$ CF-LR-Mesh. The approach is similar, however, we use a 4×4 block of processors to simulate each processor of the PR-Mesh. Replace each undirected CF-LR-Mesh bus by two "directed" buses, although the buses are not actually directed. This is similar to the block shown in Figure 5.1, such that the inner four processors are used for routing and the center two processors along the perimeter of the block contain the buses. Rank the processors along each bus in $O(1)$ time using prefix sums. During this step, the active processors are on the bottom layer of the CF-LR-Mesh. The CF-LR-Mesh configures each of its layers the same as the PR-Mesh configurations and then processors with rank j write on layer j. In this way, processors with higher priority on a bus write on lower layers than other processors within the same bus. Then, buses are formed between layers, and processors that received a message disconnect from upper layers and write its message on the bus. This allows it to properly handle any concurrent writes.

Vaidyanathan and Trahan [75] established that it is possible to translate a three-dimensional R-Mesh to a two-dimensional R-Mesh by increasing the number of processors by a factor of the smallest dimension. If we were to translate this three-dimensional CF-LR-Mesh to two-dimensions, this would result in an $O(N^3) \times O(N^2)$ CF-LR-Mesh, which is smaller by a factor of N than the model used in the previous simulation.

Combining the previous results, we obtain the following result.

Corollary 5.4 *PR-Mesh*$\left(\log^j N, \text{poly}(N)\right) =$ *CF-LR-Mesh*$\left(\log^j N, \text{poly}(N)\right)$ $=$ *LR-Mesh*$\left(\log^j N, \text{poly}(N)\right)$, *for each* $j \geq 0$.

Corollary 5.5 *PR-Mesh*$(1, \text{poly}(N)) = L$.

5.2 Complexity of the APPBS

The *Array of Processors with Pipelined Buses using Switches* (APPBS) [24] is another reconfigurable model that uses pipelined optical buses. We will first describe the structure of the APPBS and then relate the complexity of the APPBS to the PR-Mesh in Section 5.2.2.

5.2.1 Structure of the APPBS

Unlike the structure of the PR-Mesh, the APPBS uses four switches at each processor to connect to each of the adjacent buses (Figure 5.3(a)). Four configurations are available to each switch. Figure 5.3(b) shows the configurations available to the top right switch at a processor. Each processor locally controls its switches, and can change its configuration once or twice at any petit cycle(s) within a bus cycle. (Recall from Chapter 2 that a petit cycle is the node-to-node propagation delay.) The available switch configurations form non-linear buses that are not allowed in the PR-Mesh, though the model is restricted so that only one of two possible converging paths can carry a message in any given petit cycle, so messages do not collide. This does allow messages to be interleaved from different buses. To overcome the obstacle of non-linear buses or the "merged" switch configurations, we create copies of the buses for each message sent. We describe this in more detail later in this section.

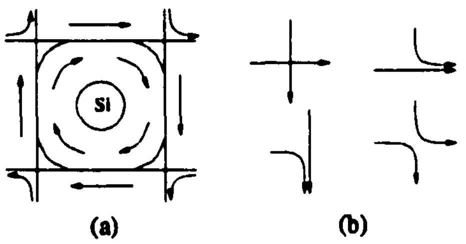


Figure 5.3: APPBS processor with switches: a) switch connections at each APPBS processor; b) switch configurations of top right switch at each APPBS processor.

Another difference between the PR-Mesh and the APPBS is that the APPBS cannot end a bus in the middle of the mesh, so each bus must extend to the outer processors in the mesh. The APPBS can use either the coincident pulse technique or the control functions *send*(m) and *wait*(n) to send a message from processor m to processor n. These functions define the number of petit cycles processor m has to wait before sending a message and processor n must wait before reading a message.

The ability of different switches to change their settings during different petit cycles could result in many different model configurations within a single bus cycle. Note that (i) the path any given message traverses is linear, despite all the switch changes, and (ii) a message may follow a different path than the one that initially precedes (or succeeds) it in the pipeline. If we do not allow an increase of processors on an N^2-processor PR-Mesh, then simulating an APPBS appears to require one step to simulate each petit cycle, leading to $O(N^2)$ steps to simulate each step of an N^2-processor APPBS. To overcome the obstacle of changing switch settings, we use a block of processors to simulate each APPBS processor, as in the simulation of an LR-Mesh by a CF-LR-Mesh. By allowing the number of processors to increase by a polynomial factor, the PR-Mesh can simulate each step of an APPBS in a constant number of steps.

In the other direction, the obstacles to simulating a PR-Mesh by an APPBS are that the APPBS does not have delay loops and is not able to segment its buses. To simplify the description of how we overcome these problems, we simulate an CF-LR-Mesh by an APPBS, rather than a PR-Mesh by an APPBS. This, along with the result of Corollary 5.4, implies that the APPBS can simulate any step of a PR-Mesh in constant steps using polynomial processors.

5.2.2 Relating the APPBS and PR-Mesh

Theorem 5.6 *PR-Mesh* $\left(\log^j N, \text{poly}(N)\right) =$ *APPBS* $\left(\log^j N, \text{poly}(N)\right)$.

Proof: **Simulation of APPBS by PR-Mesh:** Let $\mathcal{S}$ denote an $N \times N$ APPBS and let s_{ij} denote a processor of $\mathcal{S}$. We construct an $O(N) \times O(N) \times O(N^2)$ PR-Mesh $\mathcal{P}$ that simulates each step of $\mathcal{S}$ in $O(1)$ steps. Let layer ψ of $\mathcal{P}$ represent the APPBS configuration at petit cycle ψ, where $0 \leq \psi < N^2$. $\mathcal{P}$ creates a vertical bus representing the path each message would follow over the APPBS, such that the message passes switches in layer ψ corresponding to the APPBS switches it would pass in petit cycle ψ. This way, time travels up by layers within a single step of the PR-Mesh.

We first present an APPBS simulation by a PR-Mesh that does not allow non-linear connections and then extend the simulation to include non-linear connections.

Within each layer of the PR-Mesh, we use a 4×4 block of processors to simulate each processor of the APPBS, as in Section 5.1.1. Let $block_{ij}$ simulate s_{ij}. Refer to Figure 5.1 to see the arrangement of processors and blocks in each layer. The eight port processors of $block_{ij}$ represent the four ports of s_{ij} as well as the direction of the port connections. Each block in layer ψ sets its configuration to simulate the corresponding APPBS processor during petit cycle ψ. Blocks connect within the same layer to the preceding block on the bus and then route the bus up to the next layer. Referring to Figure 5.1(b), the block shown represents a processor in which a bus enters from the west port and leaves by the north port.

Consider $block_{ij}$, such that s_{ij} has the function $send(i, j)$. The block should send its message during petit cycle $send(i, j)$, however, all writing processors send their message in petit cycle 0 from layer 0. $Block_{ij}$ first broadcasts the value it holds for $send(i, j)$ along its bus. The block on the bus with value $wait(g, h)$ in layer k, such that $k = wait(g, h) - send(i, j)$, determines that it should receive the message. Next, $block_{ij}$ broadcasts its message, and each block on the bus in every layer either accepts or ignores the message it receives depending on the above considerations.

The simulation described above properly handles messages sent by an APPBS, however, certain switch configurations are not addressed in this simulation. To accommodate the non-linear, "merged" switch configurations of an APPBS switch we duplicate the simulation described for each message sent. Since non-linear connections are not allowed by the PR-Mesh and the path that each individual message follows is linear, we identify the path for a particular message within its own copy.

In this way, we use N^2 copies of an $O(N) \times O(N) \times O(N^2)$ PR-Mesh to handle all switch settings of an APPBS and all possible messages.

First, each processor in layer ψ of the PR-Mesh sets its configuration as the corresponding APPBS processor during petit cycle ψ as in the previous simulation, if it is a linear connection. Processors simulating merging switches will act as nodes in a tree and communicate with its neighbors to determine if it has a parent in order to identify the root of the tree and the leaves. Processors with linear connections act as edges in the tree. We create a linear acyclic bus that traverses the path of an Euler tour of the tree. The root of the tree segments this bus ensuring that the bus is acyclic. With the merging processors acting as nodes in the tree, we perform a prefix sums operation on the Euler tour, such that each node holds a value of '1'. This ranks the nodes of the tree and provides a preorder numbering of the nodes in the tree. An example of a tree with preorder numbering is shown in Figure 5.4.

We will consider one such copy for one particular message that passes through the leaf with preorder number j. The leaf broadcasts the value j within this message copy. All processors within the copy for this message can determine which merged setting to assume based upon its own preorder number and the number for this copy. For instance, a node with preorder number $i < j$, determines that if it is to route the message further up the tree, then the message will be received from the right. A node with preorder number $k > j$, determines that if it is to route the message further up the tree, then the message will be received from the left. Once all switches are set, the messages are sent as in the earlier simulation in a constant number of steps. Since the APPBS guarantees no message conflicts, only one block of processors simulating a particular APPBS processor will receive a message in a given layer of the PR-Mesh. Recall that each layer of the PR-Mesh represents a given petit cycle. Therefore, if two processors within the same layer that are simulating the same APPBS processor receive a message, then there was a conflict, however, this will not occur. As a result, we first merge the messages to one block by forming horizontal buses and then broadcast the received messages in each layer to the leftmost block. Next, we form buses across layers and send received messages down to the lowest layer as before.

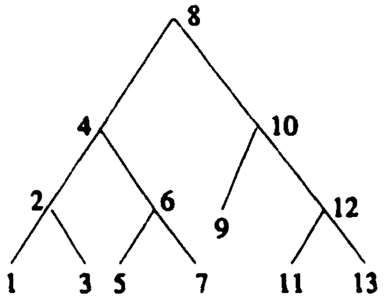


Figure 5.4: Preorder numbering of nodes in a tree

Therefore, a polynomial size PR-Mesh can simulate each step of an $N \times N$ APPBS in $O(1)$ steps, and $APPBS\left(\log^j N, \text{poly}\,(N)\right) \subseteq PR\text{-}Mesh\left(\log^j N, \text{poly}\,(N)\right)$.

Simulation of PR-Mesh (via CF-LR-Mesh) by APPBS: We now present a simulation of a CF-LR-Mesh that can in turn simulate a PR-Mesh. Now let $\mathcal{L}$ denote an $N \times N$ CF-LR-Mesh and let l_i denote a processor of $\mathcal{L}$ numbered in row major order. We construct an $O(N) \times O(N)$ APPBS S that simulates each step of $\mathcal{L}$ in a constant number of steps.

We use a 3×3 block of processors in S to simulate each processor l_i of $\mathcal{L}$, as shown in Figure 5.5(a). The center processor of the block, sc_i, sets its switches corresponding to the port configuration of l_i, and the remaining processors simulate the instances of buses that are segmented in $\mathcal{L}$. All of these processors set their switches to straight. If a bus ends at one of the ports of l_i, then the corresponding "port processor" (that

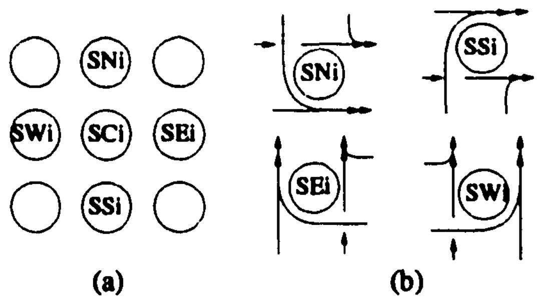


Figure 5.5: Configuration of APPBS processors to simulate a CF-LR-Mesh: a) 3×3 block of APPBS processors for each CF-LR-Mesh processor; b) configuration of port processors for a bus ending at a port of l_i.

is, either sn_i, ss_i, se_i, or sw_i) sets its switches as shown in Figure 5.5(b). This will form alleyways to shunt messages if a bus is supposed to end. All processors on the alleyway disregard messages sent along alleyways, except for the port processor at which the bus was to end. To simulate a communication step, first set all switches as described above and send the messages along the buses. Next, all processors set their switches to straight, and any port processor that handled a bus termination sends the message to sc_i, so that sc_i can get the last message sent on its bus.

Thus, an APPBS of $O(N) \times O(N)$ size can simulate each step of an $N \times N$ CF-LR-Mesh in $O(1)$ steps. Combining this with the fact that a CF-LR-Mesh of $O(N^3) \times O(N^3)$ size can simulate each step of an $N \times N$ PR-Mesh in $O(1)$ steps (Theorem 5.3), we have $\textit{PR-Mesh}\left(\log^j N, \text{poly}(N)\right) \subseteq \textit{APPBS}\left(\log^j N, \text{poly}(N)\right)$.

Therefore, $\textit{APPBS}\left(\log^j N, \text{poly}(N)\right) = \textit{PR-Mesh}\left(\log^j N, \text{poly}(N)\right)$. ■

It is possible to simulate a CF-LR-Mesh by an APPBS without using merging switches. This can be accomplished by increasing the number of rows and columns by a factor of N to give individual alleyways for each port.

5.3 Complexity of the AROB

The *Linear Array with Reconfigurable Optical Buses* (LAROB) and AROB [63, 68], are similar to the LARPBS and PR-Mesh, respectively, with some extra hardware features. They are able to segment buses into separate subarrays as are the LARPBS and PR-Mesh. We will first describe the structure of the AROB and then relate the complexity of the AROB to the PR-Mesh in Section 5.3.2.

5.3.1 Structure of the AROB

Each processor of the AROB can add an arbitrary number of unit delays to shift the select pulse with respect to the reference pulse. There is also a relative delay counter and an optical rotate-shift register at each processor enabling it to perform a bit polling operation within one step. This is the ability to select the k^{th} bit of each of N messages and determine the number of these bits that are set to 1. Pavel and Akl [61] also presented an extended version of the LAROB. The extended model allows on-line switch settings during a bus cycle and the transmission of up to N messages with arbitrary word size. The AROB is also able to address processors using the control functions *send*(m) and *wait*(n) as the APPBS. The PR-Mesh is able to simulate these functions as in the simulation of an APPBS.

These features suggest that the AROB does not have the same complexity as the PR-Mesh. By allowing the number of processors to increase polynomially, however, we establish the same complexity despite these obstacles.

5.3.2 Relating the AROB and PR-Mesh

Theorem 5.7 $\textit{PR-Mesh}\left(\log^j N, \text{poly}(N)\right) = \textit{AROB}\left(\log^j N, \text{poly}(N)\right)$.

Proof: An $N \times N$ AROB can simulate each step of an $N \times N$ PR-Mesh in a constant number of steps, as it can configure its buses in the same manner and has the same capabilities. Therefore, *PR-Mesh* $\left(\log^j N, \text{poly}(N)\right) \subseteq$ *AROB* $\left(\log^j N, \text{poly}(N)\right)$.

Let $\mathcal{B}$ denote an $N \times N$ AROB and let b_i denote a processor of $\mathcal{B}$ in row major order. We construct an $O(N) \times O(N) \times O(N^2)$ PR-Mesh $\mathcal{P}$ that simulates each step of $\mathcal{B}$ in $O(1)$ steps. The approach we take to describe the simulation is to individually present simulations of each of the extra features not possessed by the PR-Mesh.

The first feature we simulate is the bit polling operation. We use a similar approach as in the APPBS simulation without "merging" switches (Section 5.2) and consider $2N^2$ layers of a PR-Mesh to simulate an AROB. Again, we use a 4×4 block of processors, as shown in Figure 5.1, to simulate each processor of the AROB on each layer. Each block sets its configuration to form buses up through the layers of the PR-Mesh. As in the proof for Theorem 5.6, all incoming connections are routed up to the next layer, and all connections coming in from the layer below are routed on the same layer to the next block on the bus. The block that corresponds to an end of a bus in the AROB sets its connections so that it ends the bus in the PR-Mesh as well. This, once again, forms a bus for each message. In contrast to previous simulations, the base layer here is layer N^2, the center layer.

Consider one of the original buses of the AROB, where the head of the bus is processor b_{ij}. All processors on the bus now determine their distances from the head of the bus by computing prefix sums [63] on the upper N^2 layers of the bus. Call this distance d_{gh} for processor b_{gh}. Do this for all buses of the AROB.

Each block on the center layer has its own personal copy of its bus. The bus corresponding to b_{gh} begins in layer $N^2 - d_{gh}$. This way the bit that is to be polled will be polled in the corresponding layer. Each block on the center layer broadcasts its corresponding message. If processor b_{gh} was to perform a bit-polling operation on the i^{th} bit, then each block p_{gh} on each layer that received a message extracts the i^{th} bit from the message it read and uses this value in the next step. Next, all blocks connect in vertical buses and sum these bits to get the bit polling result within a constant number of steps. The sum obtained by p_{gh} represents the number of i^{th} pulses that are '1'.

The second feature we consider is the ability to set an arbitrary number of delays. We will use the result of the following lemmas to show that the PR-Mesh can simulate setting an arbitrary number of delays in $O(1)$ steps with a polynomial increase in the number of processors.

Lemma 5.8 *An N^2-processor LARPBS can simulate in $O(1)$ steps any step of an N-processor LAROB that allows an arbitrary number of delays.*

Proof: Let processor p_{Ni} of the LARPBS simulate processor b_i of the LAROB, so that each p_{Ni} has a segment of N processors corresponding to it. Processor p_{Ni} sends a message to each of the N processors in its segment with the value of its delay. For a delay of x_i corresponding to p_{Ni}, each of the first x_i processors of p_{Ni}'s segment sets its value to '1'. Perform a prefix sums operation over all N^2 processors. Processor p_{Ni} then adjusts its prefix sum by x_i. Based on the adjusted prefix sum value, p_{Ni} adjusts its select frame. Processor p_{Ni} sends this information to p_i. Now p_i simulates b_i and sends the messages in a normal state of operation, such that all conditional delay loops are set to straight. Only the first N processors are active in this last step. ■

Lemma 5.9 *An $O(N) \times O(N) \times O(N^2)$ PR-Mesh can simulate in $O(1)$ steps any step of an $N \times N$ AROB that allows an arbitrary number of delays.*

Proof: We first present this for an $O(N) \times O(N) \times O(N^4)$ PR-Mesh, and then reduce it down to the desired size. Configure all processors to form the buses of the AROB in the bottom layer of the PR-Mesh. Perform prefix sums on each bus so each processor can get its ranking within its bus. The head of each bus sends its ID along the bus to provide a bus ID to all processors on that bus. Due to the third dimension, each of the processors on the bottom layer has an N^4-processor LARPBS associated with it. For the bus with ID (j, k), map the i^{th} processor on bus (j, k) to processor $p_{N^2 i}$ of the N^4-processor LARPBS beginning at processor (j, k) on the bottom layer. From Lemma 5.8, each processor can determine the number of delays that will affect it, and can adjust its select frame accordingly. (The longest bus length possible for the AROB is N^2 processors and each processor is able to insert up to N^2 delays, hence the PR-Mesh uses a bus of length N^4 to simulate each bus of the AROB.) Repeat this four times, once for each port, in case a processor was the head of more than one bus. Once all select frames have been adjusted, all processors along the bottom layer can send their messages through the bottom layer.

To reduce the PR-Mesh to N^2 layers, first rank processors along each bus as before. Next the tail of each bus sends the count to the head of its bus, so the head holds the total number of processors on its bus. To get the bus IDs, perform a prefix sum of the bus lengths using the heads of buses. (In the case above, the bus ID was simply the index of the processor at the head of the bus. In this case, the bus ID is determined from an ordering of the buses.) By connecting the three-dimensional mesh in a snake-like pattern, the entire mesh is just a one-dimensional LARPBS. Now, place each bus in contiguous segments of the mesh, with the starting location depending on the bus

ID. This problem then reduces to the one presented in Lemma 5.8. Therefore, we can simulate any step of the AROB using arbitrary delays on a PR-Mesh in $O(1)$ steps.

■

The third feature considered is the on-line switching ability of the AROB. This simulation follows the simulation of this feature of the APPBS without "merging" switches by the PR-Mesh in Section 5.2.

The fourth feature considered is the relative delay counter. This counter of each AROB processor is able to detect the relative time delay between select and reference pulses that pass each processor. We proceed as before configuring layer i of the PR-Mesh as the AROB configuration at petit cycle i. The message is sent with the corresponding select pulses injected and a single reference pulse in the highest slot. Next, configure buses that connect each layer together and any processor that received a message broadcasts its layer value on the bus. This provides information regarding the time slot of a select pulse. Each processor can then use this information along with its layer value to determine the relative delay between the select and reference pulses. Combining these results, we can simulate any step of an AROB on a PR-Mesh by performing the following steps:

- Perform bit polling if required.
- Calculate the number of delays for each message.
- Adjust select frames.
- Send messages.

This proves that $AROB\left(\log^j N, \text{poly}(N)\right) \subseteq \textit{PR-Mesh}\left(\log^j N, \text{poly}(N)\right)$, thus establishing that the two models have the same complexity. ■

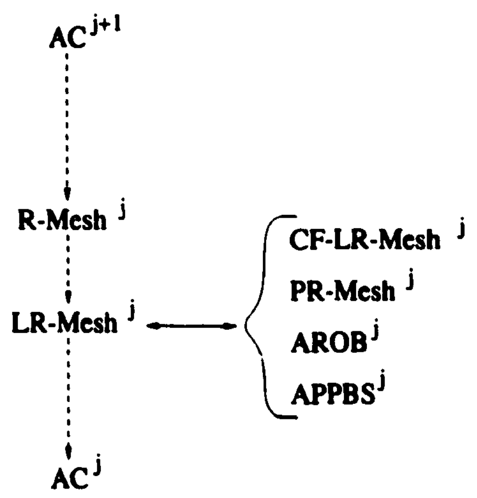


Figure 5.6: Complexity class relations

We have established that the LR-Mesh, CF-LR-Mesh, PR-Mesh, APPBS, and the AROB have equivalent complexity and can solve any problem of size N within class L in constant time using polynomial in N processors. Figure 5.6 places these models in relation to other models and their established complexity classes. For model Z, let Z^j denote the class of languages accepted by model Z in $O(\log^j N)$ time with number of processors polynomial in N. Class AC^j is the class of languages accepted by logspace-uniform, unbounded fan-in circuits of size polynomial in N and depth $O(\log^j N)$. The dashed lines represent previously known results [72]. The solid line represents results obtained in this work and places the models within their corresponding complexity class.

The results obtained prove that pipelining messages using optical buses provide us with better efficiency than electrical buses. The PR-Mesh requires fewer buses than the CF-LR-Mesh, however, the PR-Mesh possesses the same limitations as the CF-LR-Mesh in solving graph problems since non-linear connections are not allowed.

Chapter 6

Algorithm Development

It is always desirable to improve the efficiency of existing algorithms, either by reducing the time required to execute a specific algorithm or by reducing the number of processors required. In this chapter we improve existing algorithms in the areas of computational geometry, image analysis, and arithmetic algorithms by adapting them to the PR-Mesh (Section 6.1). We also briefly discuss a few algorithms that are likely candidates to be improved.

When developing algorithms, many assumptions are made that are not always realistic during implementation. Thus far, all of our work has assumed that N processors are connected to an optical bus, with no restriction on the size of N. There are many practical constraints that could have impact on the length of the bus considered, which would, in turn, limit the number of processors that could be connected to the bus. Section 6.2 discusses some of these restrictions and an approach to work within these limitations.

6.1 Algorithm Improvement

Certain features of the LARPBS and PR-Mesh may be exploited to develop faster and more efficient algorithms. These models are able to compact data, perform

Table 6.1: Improved Algorithms for the LARPBS and PR-Mesh

Algorithm	Size Reduction Factor
dominance counting	$\sqrt{N}$
prefix modular k	N
number conversion	N
conversion to quadtree	N

binary prefix sums, and route any permutation in a constant number of steps with N processors for a problem size of N. Binary prefix sums takes $O(\log N)$ steps on an N-processor LR-Mesh. Compaction and permutation routing take $O(N)$ steps on an N-processor LR-Mesh. Alternatively, at greater size cost, each of these operations takes $O(1)$ steps on an $N \times N$ LR-Mesh. The ability to pipeline messages enables the use of smaller sized models, as extra buses are not required to send multiple messages simultaneously.

A second advantage of being able to pipeline messages is a savings in steps, because many messages can be in transit during one step, and space, because extra buses are not required to transmit messages simultaneously. Another feature that is not possible is the ability to send a message on an electrical bus past a processor connected to the bus without the processor receiving it. We will identify problems and algorithms in which we take advantage of these features. The specific problems we consider and the size improvements for each problem are given in Table 6.1. The size improvements are relative to the best known R-Mesh algorithms for the problems.

6.1.1 Computational Geometry Algorithms

Computational geometry has a wide range of applications. Computer graphics utilizes computational geometry because the scenes displayed consist of geometric objects.

Geographic information systems are concerned with points and regions on the surface of the earth, generating many geometric problems. Robotics is another area utilizing computational geometry because robots are basically geometric objects that operate in 3-dimensional space.

Many computational geometry algorithms exist on reconfigurable models with electrical buses, such as convex hull [16, 27], triangulation [51], Voronoi diagram [19], and point visibility [32]. Few such algorithms, however, exist on reconfigurable optical models. We are interested in identifying algorithms that are adaptable to the LARPBS or PR-Mesh such that the time and/or size can be improved.

We have improved an existing algorithm to perform dominance counting. *Dominance counting* is to determine for each point, p, in a set S of N distinct planar points, $|\{q : q \in S, p_x > q_x \text{ and } p_y > q_y\}|$. Nigam and Sahni [50] presented an algorithm to solve this problem on an $N \times N$ R-Mesh in a constant number of steps. We follow their procedure, however, we are able to reduce the number of processors, obtaining the following result.

Lemma 6.1 *Dominance counting for each point $p \in S$, where $|S| = N$, can be computed on an $N^{1/2} \times N$ PR-Mesh in a constant number of steps.*

Proof: **Step 1:** Sort S by the y-coordinate in $O(1)$ time (Theorem 3.2). Store the results, one element per processor, in the top row of the PR-Mesh. Partition S into $N^{1/2}$ sets Y_i, $1 \leq i \leq N^{1/2}$, such that $|Y_i| = N^{1/2}$ and no point in Y_i has a larger y-coordinate than any of the points in Y_{i+1}. Within each partition, Y_i, sort by the x-coordinate. Let the processor with the highest index in Y_i be the border processor for Y_i.

Step 2: Sort S by the x-coordinate in $O(1)$ time (Theorem 3.2). Partition these elements into $N^{1/2}$ sets X_i, $1 \leq i \leq N^{1/2}$, such that $|X_i| = N^{1/2}$. Store the results, one element per processor, in the top row of the PR-Mesh.

Step 3: Each processor sets its configuration to fuse its North and South ports to form vertical buses. Each processor on the top row broadcasts the two values it holds (one element of X_i and one element of Y_i) on its column bus.

Step 4: Each processor sets its configuration to fuse its East and West ports to form horizontal buses. Broadcast the border element for Y_i on row i, for $0 \leq i < N^{1/2}$.

Step 5: On row i, compress elements that have a larger y-coordinate than the border element for Y_i. Let $S_{ij} = X_i \cap Y_j$. For each $p \in S_{ij}$, $DY(p) =$ (number of points dominated by p in $(Y_j - S_{ij})$), $DX(p) =$ (number of points dominated by p in X_i), and $D(p, S) = DY(p) + DX(p) + \sum_{u<i} \sum_{v<j} |S_{uv}|$.

Perform the summations on each row in constant time obtaining the final result.

■

The ability to identify the maximum/minimum of N elements on an N processor LARPBS in $O(\log \log N)$ steps [56] provides a savings in steps in parts of two existing algorithms. The first is an algorithm to determine the ***point visibility*** of a simple polygon using an R-Mesh. This problem is to find for a given point z in the interior of an N vertex polygon P, all the points of P that are visible from z. The existing R-Mesh algorithm [32] runs in $O(\log^2 N)$ steps; we conjecture, however, that it is possible to run in $O(\log N \log \log N)$ steps on an LARPBS using the same number of processors. The second algorithm is one to compute the Voronoi diagram for N points. The ***Voronoi diagram*** takes a set S of N points and decomposes the space in regions around each point, such that all points in the region around p_i are closer to p_i than to any other point in S. The existing R-Mesh algorithm [19] runs in

$O(\log N \log\log N)$ steps, however, we have been able to reduce the steps required for some phases of the Voronoi diagram algorithm.

6.1.2 Arithmetic Algorithms

Arithmetic algorithms include a wide range of problems that may have room for improvement. Examples of such algorithms include matrix multiplication [17, 62], Discrete Fourier Transform (DFT) [60], multiple addition [47, 60], and singular value decomposition [55]. These algorithms depend heavily on multiple additions as well as compaction of data, both of which are more efficient on the LARPBS and PR-Mesh than on the LR-Mesh and R-Mesh.

We now present extensions of some arithmetic algorithms concerning matrix multiplication. Pavel and Akl [62] presented results leading to the multiplication of dense $N \times N$ matrices on the AROB, in which the word size is assumed to be $O(\log N)$ bits. (Refer to Sections 3.1 and 5.3.1 for a description of the AROB.) We are interested in generalizing their results to account for an arbitrary word size. This can be done by either increasing the time required or the number of processors required as a factor of the word size. By allowing the time to increase, we achieve the following results for arbitrary word size of w-bits.

Lemma 6.2 *Addition of N w-bit numbers can be performed in $O(\lceil \frac{w}{\log N} \rceil)$ steps on an $O(\log N \times N)$ AROB.*

Proof: Assume the top row holds the N values v_j, $0 \leq j < N$. Broadcast v_j in column j. Processor p_{ij} stores the $k * i^{th}$ bit of v_j, for $0 \leq i < \log N$, $0 \leq j < N$, and $0 \leq k < \lceil \frac{w}{\log N} \rceil$. In l iterations, where $l = \lceil \frac{w}{\log N} \rceil$, each row determines the sum of the bits within its row using binary prefix sums. This results in $(l \log N) \log N$-bit binary values. Locally adjust the weights of the values depending upon the row in which the values are stored. Sum these values in $\log(\lceil \frac{w}{\log N} \rceil)$ steps. ■

Lemma 6.3 *For a w-bit word size, the multiplication of an $N \times N$ matrix A with an $N \times 1$ vector b can be performed in $O(\lceil \frac{w}{\log N} \rceil)$ steps on an $N \times N \times \log N$ AROB.*

Proof: Assume the elements of matrix A are stored in the base array of the AROB. Assume the elements of vector b are stored in the top row of the base array of the AROB. Broadcast b_j down column j of the base array. Processor $P_{i,j}$ locally computes $a_{i,j} * b_j = c_{i,j}$. The next step is to compute the elements of the vector c by $c_i = \sum_j c_{i,j}$. This results in the addition of N w−bit numbers on each of the N rows. Using the third dimension and Lemma 6.2, the multiplication can be completed. ■

Lemma 6.4 *Given two $N \times N$ matrices A and B, w-bit word size, the matrix multiplication $AB = C$ can be performed in $O(\lceil \frac{w}{\log N} \rceil)$ steps on an $N \times N \times N \times \log N$ AROB.*

Proof: Route the elements of A and B such that $p_{i,j,k}$ holds $a_{i,k}$ and $b_{k,j}$. Locally compute the factor $c_{i,j}(k) = a_{i,k} * b_{k,j}$. The next step is to compute the elements of the matrix C by $c_{i,j} = \sum_k c_{i,j}(k)$. This results in the addition of N w−bit numbers on each column $p_{i,j,k}$. This summation can be computed in $O(\lceil \frac{w}{\log N} \rceil)$ steps. ■

It would also be beneficial to extend these results to use floating point inputs instead of restricting them to integers.

6.1.3 Image Analysis Algorithms

Many algorithms exist in the area of image processing, such as quadtree building [33], histogram finding [29], Hough transform [53], and nearest neighbor [57], to name

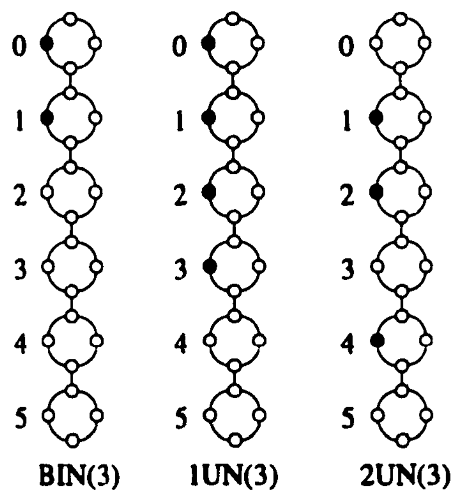


Figure 6.1: [29] Three different representations of number 3

a few. By taking advantage of the key features of the LARPBS and PR-Mesh, we improve some of these algorithms.

A basic operation in image processing is to compute a *histogram* of an $N \times N$ image. The problem is to determine the number of occurrences of each of h grey level values within the image. The R-Mesh histogram algorithm proposed by Jang *et al.* [29] consists of a few subroutines. The two subroutines we consider here both run in a constant number of steps on an LARPBS.

Lemma 6.5 *Prefix modular k computation of a 0/1 sequence of length N can be performed in $O(1)$ steps on an N-processor LARPBS.*

Proof: First compute the prefix sums of the N numbers in a constant number of steps. Next, perform a local modulo k operation. ∎

A group of N processors can represent a number with value less than N in different formats. In the 2UN representation of integer i, a subset of i processors hold a '1' and the remaining processors hold '0'. In the 1UN representation of integer i, each processor k, $0 \leq k \leq i$, holds a '1' and the remaining processors hold '0'. In the BIN representation of integer i, each processor k holds a '1' if the k^{th} bit position of i is a '1' and the remaining processors hold '0'. Refer to Figure 6.1 for an example of different representations.

Lemma 6.6 *Conversion of a number from 2UN or 1UN representation to either 1UN or BIN representation can be performed in $O(1)$ steps on an N-processor LARPBS.*

Proof:

- 2UN $\longrightarrow$ 1UN: Sum the bits of the 2UN representation in one step. Processor p_0 stores the sum, j, and then broadcasts the value to all processors. Each processor with index i such that $i \leq j$ sets its bit to high, thus obtaining the 1UN representation.

- 2UN $\longrightarrow$ BIN: Sum the bits of the 2UN representation. Processor p_0 stores the sum, j, and then broadcasts the value to all processors. Processor p_i sets its bit to high if bit i of the binary representation of j is a '1', thus obtaining the BIN representation.

- 1UN $\longrightarrow$ BIN: Each processor with a '1' broadcasts its index to the head of the array. The head of the array receives the integer value due to the priority write property and then broadcasts the value. Processor p_i sets its bit to high if bit i of the binary representation of j is a '1', thus obtaining the BIN representation.

■

Both of these subroutines provide a savings in the number of processors used. The first subroutine as presented by Jang *et al.* uses a $(k+1) \times 2N$ R-Mesh as opposed to a $1 \times N$ PR-Mesh. The second uses a $\log^2 N \times N$ R-Mesh. This may carry over to a savings of size to find the histogram of an image, as these subroutines are utilized in the algorithm. The obstacle arises from trying to reduce the number of processors to sort the pixels of the image.

A *quadtree* is a data structure often used to represent binary images and finds use in many operations on binary images and spatial information systems. It breaks an $N \times N$ image into quadrants, such that the root represents the entire image, and each node can have up to four children. It then continues to break the image down until each pixel represented by a node is of the same color. For example, if the image consists of all pixels being the same color, then the quadtree would contain only the root node, else, the root would have four children representing the NW, NE, SW, and SE quadrants of the image.

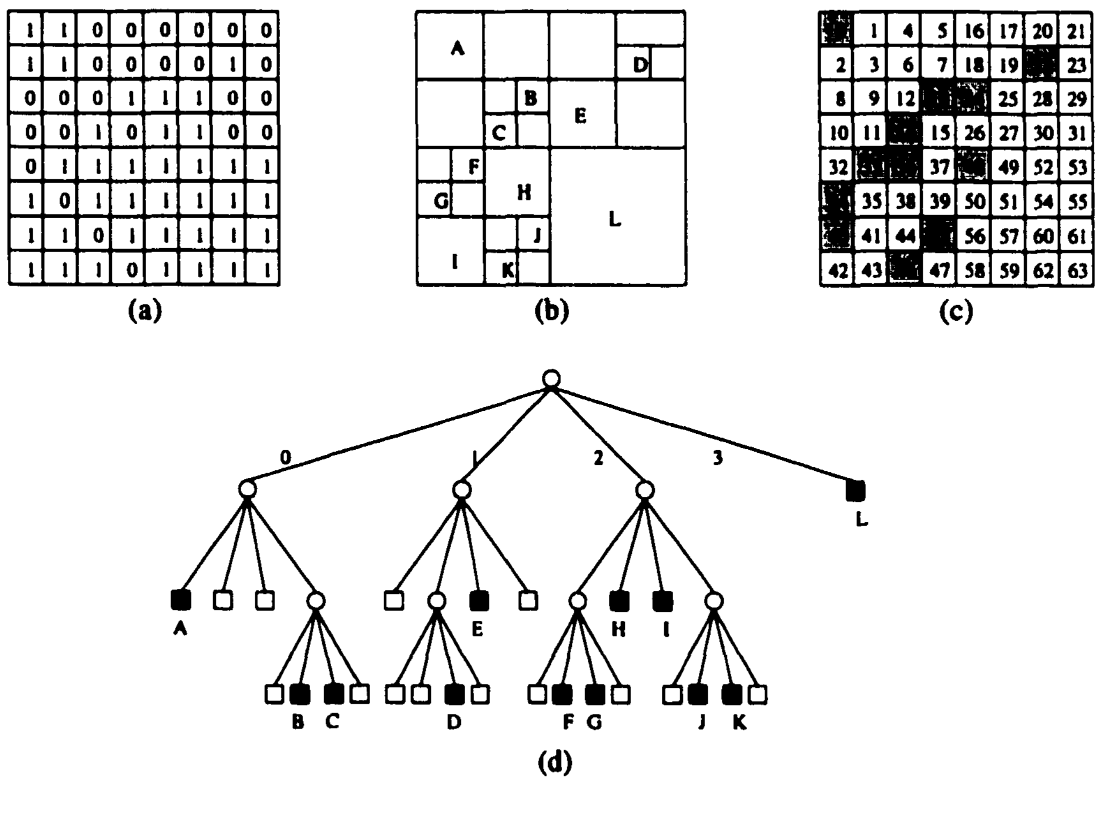


Figure 6.2: [33] Image representations (a) 8×8 binary image, (b) block decomposition of the binary image, (c) shuffled row-major order, (d) quadtree representation.

The specific algorithm in which we are interested is converting between a quadtree and a binary image, which takes a constant time number of steps on an $N \times N \times N$ R-Mesh [33]. Refer to Figure 6.2 for the different representations.

There are different methods to store the quadtree representation. An obvious method would be to use a tree structure. This, however, requires excessive space due to the pointers needed. An alternative method is a linear quadtree, in which only the black leaf nodes are stored. The data necessary for each black leaf node is the shuffled-row major number (see Figure 6.2(c)) of the top leftmost pixel of its block i (shown as a shaded block in the figure), and the level on which the node is located in the tree l (see Figure 6.2(d)). Represent each black node leaf by (i, l). Referring to the binary image in Figure 6.2(a), the linear quadtree representation is: (0,2), (13,3), (14,3), (22,3), (24,2), (33,3), (34,3), (36,2), (40,2), (45,3), (46,3), (48,1).

The algorithm presented by Kim and Jang [33] uses a three-dimensional R-Mesh. The algorithm uses the third dimension to perform permutation routing, compression, and basic data movement of N^2 elements. An $N \times N$ PR-Mesh can perform these operations in $O(1)$ steps, providing us with the following result.

Theorem 6.7 *Conversion from an $N \times N$ binary image to a quadtree can be performed in $O(1)$ steps using an $N \times N$ PR-Mesh.*

Quadtree representations find use in computing certain distance transforms, spatial information systems, and geometric applications, including data clustering and shape representation [36]. Therefore, improving the efficiency of the quadtree conversion could carry through to other image analysis algorithms.

6.2 Algorithms with Physical Constraints

In this section we consider some physical constraints that can impact algorithm performance. For instance, when considering optical models in practice, a pulse traveling from one processor to the next may not take exactly the same time. Errors of this type may accumulate when the number of processors is large, resulting in synchronization error [13]. Degradation of light intensity is another problem that grows with an increase in distance, or processors, and may prevent detectors on the receiving end from properly interpreting data. Repeaters or optical amplifiers could be placed at regular intervals to overcome these problems. This, however, would introduce additional delays along the bus, and the pulse timing for receiving messages would have to be adjusted.

One approach we can take to accommodate the problem is to place a restriction on the communication length between two processors. For instance, on an LARPBS with N processors, permit a processor to send a message to another processor only with distance at most L.

In the following sections we provide algorithms to compute prefix sums and perform compression for an N-processor LARPBS that has the restricted communication length described above. The base of the algorithms on an unrestricted PR-Mesh follows the approach of Pan and Li [56]. The results obtained for these two algorithms are time optimal for this communication length.

6.2.1 Prefix Sums with Restricted Communication Length

Assume each processor holds one data element. The LARPBS sets its segment switches so that there are $2N/L$ subarrays of length $L/2$. Number each segment from 0 to $2N/L - 1$. Each processor knows the value of N and L and can thus determine if it is in an even or odd segment. Denote h_i as the head of subarray i, for $0 \leq i < 2N/L$. Perform prefix sums within each subarray. Let h_i hold the prefix sum for subarray i, ps_i.

Consider segment head h_i. If i is even, then segment head h_{i+1} segments the bus and h_i sends ps_i to h_{i+1}. Next, h_{i+1} sets it segment switch to straight and h_i segments the bus. Segment head h_i now receives ps_{i-1} from h_{i-1}. If i is odd, the steps are in reverse order. Each segment head now segments the bus to form the $2N/L$ subarrays as before. Processor h_i now broadcasts ps_{i-1} within its subarray and forwards ps_{i-1} to h_{i+1} after setting its segment switch as in the previous step. This is repeated for $2N/L$ phases, providing us with the following result.

Lemma 6.8 *Prefix sums of N elements can be computed in $O(N/L)$ steps on an N-processor LARPBS with communication length restricted to L.*

6.2.2 Compression with Restricted Communication Length

Assume that each processor of an N-processor LARPBS holds an element that is either marked or unmarked. Recall from Section 2.3.2 that the *compression* algorithm shifts all marked elements to the lower end of the array, namely processors p_0 through p_{x-1}, and unmarked elements to the upper end of the array. The algorithm also maintains the order within the marked elements and within the unmarked elements. Let x denote the number of marked elements.

First the LARPBS computes the prefix sums of the marked processors in $O(N/L)$ steps as in the previous lemma. The prefix sum computed provides the index of the processor to which the marked element should be routed. The processor with index $N - 1$ broadcasts the total number of marked processors by passing the value from one segment head to the next in $O(N/L)$ steps. Next, compute the prefix sums of the unmarked processors. By adding this value to the sum of the ranked processors, the index of the processor to which the unmarked element should be routed is determined.

Route the messages to the proper processors in $2N/L$ phases, comprised of the following steps.

1. Even indexed segment heads segment the bus.

2. Processor p_i with rank k in an even numbered segment sends the element it holds and its destination to the k^{th} ranked processor of the segment ahead of it if the destination has index greater than i.

3. Processor p_j with rank k in an odd numbered segment sends the element it holds and its destination to the k^{th} ranked processor of the segment below it if the destination has index less than j.

4. Odd indexed segment heads segment the bus.

5. Repeat the previous steps.

6. If a processor received an element that has a final destination within its segment, then it sends the element to its final destination.

After $2N/L$ phases, the messages reach the desired locations.

Lemma 6.9 *Compression of x elements, where $x \leq N$, can be performed on an N-processor LARPBS with communication length restricted to L in $O(N/L)$ steps.*

Chapter 7

Fault Tolerant Algorithms

As mentioned in the introduction, architectures using optically pipelined buses suit many communication-intensive applications. As the sizes of the applications and problems grow, so does the number of processors. The number of processors involved in the systems considered raises the probability of a fault occurring. The occurrence of even a single fault can have dramatic impact upon the performance of various parallel platforms. It is not practical to allow an entire system to fail due to the failure of a few components. For this reason, researchers have proposed fault tolerant algorithms for many parallel architectures, such as the hypercube, mesh, and torus [11, 12, 58, 59]. They have not, however, addressed the issue of fault tolerance for reconfigurable models, and more specifically, for any of the optically pipelined models.

In this chapter we present several basic fault tolerant algorithms for the LARPBS. Specifically, we have developed algorithms to calculate binary prefix sums, perform compression, sort, and perform a general permutation routing step on an N-processor array that can have up to $N/2$ static faults. We then extend these results to other fault tolerant algorithms in the areas of image processing and matrix operations.

Section 7.1 describes the fault model used. Section 7.2 explains the preprocessing phase for fault tolerant algorithms. We present the basic fault tolerant algorithms in detail and extend the results to other more complex algorithms in Section 7.3. Section 7.4 explains the faster methods used to design fault tolerant algorithms for an LARPBS that has a constant number of faults.

7.1 Fault Model

Let a *processing element* consist of a single processor, its conditional delay switches, and its directional couplers. We consider a processing element to be faulty if any one of its components is faulty, and refer to it as a faulty processor for short. Faults on any of the three optical waveguides are not considered.

Assume that all faults are static and occur prior to the execution of any algorithm. Therefore, faults occurring during execution of an algorithm are not considered. The algorithms presented in Section 7.3 can tolerate up to $N/2$ faults on an N-processor LARPBS. These assumptions are consistent with those described by Parhami and Yeh [59] and Kim and Park [34].

If a conditional delay switch is faulty, that is, if it is stuck in either the cross or straight position, then it remains that way for the remainder of the algorithm. Faulty segment switches are not considered, since this would result in a shorter available working array, and thus, would be a scaling problem rather than a fault tolerance problem. (For work on scalable algorithms for the LARPBS, refer to Trahan *et al.* [70, 73].)

Many fault models previously described for other architectures allow a healthy processor to detect if its neighbors are faulty [1, 59]. In the LARPBS, a fault-free processor is able to determine if either of its neighbors is faulty in two phases, with each phase consisting of a constant number of steps. During the first phase, each even numbered processor segments the bus. Next, each odd numbered processor broadcasts its index. If an even numbered processor did not receive the index of the preceding processor, then it determines that its left neighbor is faulty. Each even numbered processor now broadcasts its index. If an odd numbered processor did not receive the index of the succeeding processor, then it determines that its right neighbor is faulty. The second phase is similar, except the odd numbered processors segment the bus instead of the even numbered processors. Due to the priority write rule of the LARPBS, a healthy processor will not receive incorrect information from another healthy processor if a faulty processor is unable to segment the bus.

Many fault tolerance schemes require extra hardware. The schemes of Banerjee *et al.* [4], for instance, depend upon the existence of spare processors and links. In contrast, the method presented by Varvarigou *et al.* [76] reconfigures a faulty mesh to a smaller sized system. This results in many healthy processors being unused. There are also others that ignore data held by faulty processors and handle only one datum per healthy processor [2, 78], while some methods determine alternative paths for sending messages in order to avoid faulty processors. The method presented in this paper, however, does not require any extra hardware, utilizes all healthy processors, and does not attempt to find alternative paths. Actually, since the LARPBS is a one-dimensional array of processors, it is not possible to use a path bypassing the faults.

7.2 Preprocessing Phase

Prior to running any algorithm on a faulty LARPBS, we perform some initial processing to ensure proper execution. Each working processor, p_i, determines the number of faulty processors to its right (p_j, where $i < j < N$) that have their conditional delay switches stuck at cross. Call the value of this suffix sum f_i. This value is important because any stuck delays through which a select pulse travels will alter the destination of the message sent by a working processor. By determining the total number of stuck delays ahead of it on the bus, each working processor can adjust its reference pulse to avoid miscommunication. Processor p_i shifts its reference pulse to the left by f_i slots. With this adjustment, provided each working processor has its conditional delay switch set to straight, the message sent by p_i reaches the intended destination.

Once the information concerning the number of stuck delay switches has been determined, the LARPBS must determine a mapping scheme. The fault model that we consider does not ignore data held by faulty processors, therefore, all processors need to be mapped to the remaining working processors. Section 7.2.2 discusses this mapping.

7.2.1 Determine Number of Stuck Delay Switches

To determine the number of delay switches to the right stuck at cross, first, each working processor segments the bus if it detects a faulty processor to its left. This working processor will be at the head of its segment. Each such segment will contain exactly one interval of faulty processors ending just to the left of the head processor. Two cases arise for the remainder of the segment: 1) one or more good processors are present to the left of an interval of faulty processors, or 2) no other good processors are present.

The LARPBS proceeds in two phases to determine the number of stuck delay switches ahead of each processor. The first phase calculates the number of stuck delays within each of the segments that are formed as described above. Determining the number of stuck delays within each segment, that is, within an interval of faulty processors, is not a trivial task. Each healthy processor needs to first determine the number of faulty processors within its segment. Since the number of stuck delay switches is undetermined, a processor cannot readily address a specific processor. Therefore, healthy processors must observe the effects of stuck delay switches that shift sent messages.

The second phase uses information collected within segments to determine the number of stuck delay switches to the right of each working processor over the entire array of the LARPBS. A prefix sums operation is utilized, however, due to the faulty processors, adjustments must be made to overcome the stuck delay switches.

Delays Within Each Segment

We will first determine the number of stuck delay switches within each segment. Recall that each processor that detects a faulty left neighbor sets its segment switch to cross, thus segmenting the bus. The two possibilities are that the tail of a segment is healthy or it is faulty. The head of the segment, p_h, broadcasts its index to the segment. Any other fault-free processor, p_l, with a fault to its right, broadcasts its index to the head. (There is only one such processor in a segment that fits the first case.) If the head does not receive a message, then it determines that its segment fits the second case. We repeatedly use the head of each segment during the process of this section since the messages it sends are not affected by stuck delay switches. We now describe the method to determine the delays within a single segment, although all segments execute the appropriate case simultaneously.

Consider a segment that fits the first case, in which the tail of the segment is a healthy processor. The number of faults in the segment is $(h - l - 1)$, call this value k. The head of the segment now determines the number of stuck delays by a binary search technique. Processor p_l injects select pulses into its highest $k/2$ slots and sends its index. Processor p_h then broadcasts a message indicating whether or not it received the message. (The segment head would receive the message if there were at most $k/2$ stuck delay switches.) If it did, then p_l repeats this by injecting select pulses into its highest $k/4$ slots. If not, then p_l injects select pulses into slots $(N-1)-3k/4$ through $(N-1)-k/2$. Repeat this binary search process a total of $\log k$ times to determine the number of conditional delay switches that have failed in the cross position. Worst case time complexity is when $k = N/2$, resulting in $O(\log N)$ steps.

Now consider a segment that fits the second case, in which the tail of the segment is a faulty processor. The head of the segment needs the index of the head of the previous segment to determine the number of faults within its own segment. There could, however, be a string of such segments, each needing the index of the head of the preceding segment. We proceed in $\log N$ phases to relay information between these heads of segments, with each phase corresponding to one bit position of the processor indices. During phase i, where $1 \leq i \leq \log N$, each segment head with a '0' in bit position $i-1$ of its index segments the bus and listens while each segment head with a '1' broadcasts its index within the newly formed segment. This step is then repeated with the writers now reading, and the readers now writing. Once the preceding index is known, each segment head determines the number of stuck delays within its segment, as in the first case, in $O(\log N)$ steps. Eventually, in some phase, each segment head will receive the proper index since the two must differ in at least one bit position. In addition, the first index the segment head receives is the proper index, since the previous segment head would be segmenting the bus for each of the phases until the two communicate. With $\log N$ phases, and each phase taking $O(\log N)$ steps, the total time to determine the number of stuck delay switches in each segment takes $O(\log^2 N)$ steps. This is done in $\log N$ phases rather than a simple odd/even phase, because the two processors communicating could possibly both have odd or even indices.

Consider the example shown in Figure 7.1. The LARPBS in this example has three faulty processors, namely R_2, R_3, and R_5, each of which has its conditional delay switch stuck in the crossed position. (The delay switches of the healthy processors are shown as dashed lines, as they are able to change their settings, unlike the faulty processors.) Processor R_4 is a segment head that fits case one, and determines that two switches have failed in the cross position within its segment. Processor R_6 is a segment head that fits case two, that first determines that R_4 is the head of the previous segment, then it determines that one switch has failed in the cross position within its segment.

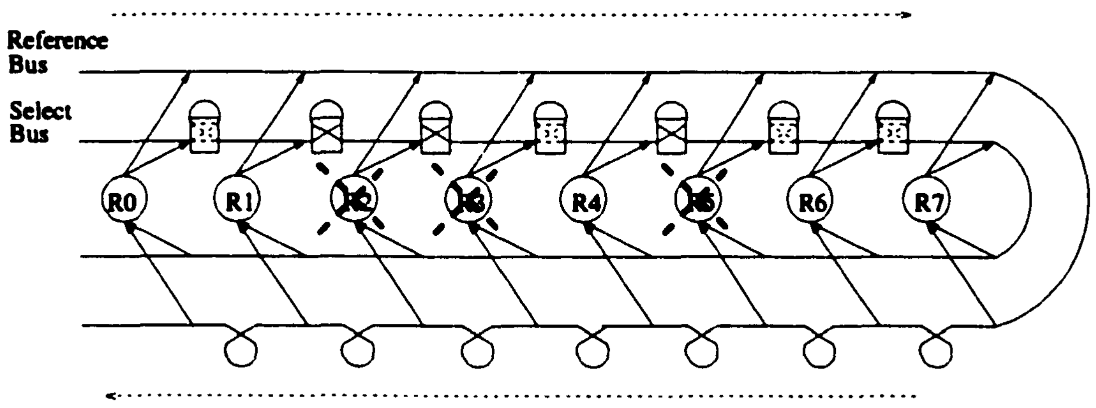


Figure 7.1: Example of a faulty LARPBS

Delays Over the LARPBS

At this point, the LARPBS has calculated the number of stuck delay switches for each segment. With this information, it is possible to determine the number of stuck delays ahead of each working processor in the array by the prefix sums of the stuck delays in each segment, as follows.

Perform a prefix sums operation as on a tree-like structure. We will refer to this procedure as the ALL PROCESSORS PREFIX SUMS. The head of each segment holds the data for its segment, and each other working processor holds a value of '0'. In phase j of the prefix sums, processor pairs with indices differing in bit position j communicate with each other. Each processor of a communicating pair must determine whether its partner is faulty, so that the working processor can take the place of the faulty processor in the following phases. For each communicating pair, the higher indexed processor segments the bus, in order for the two to exchange information by broadcasting within their segment (since the exact identity of the partner is unknown because another working processor may be substituting for a faulty expected partner). When communicating, the writing processor first sends its index and then its data so that a reading processor can determine if it is paired with a faulty processor. If the lower indexed processor is faulty, then the higher indexed processor will not receive a message. If the higher indexed processor is faulty, then it will not have segmented the bus, so the lower indexed processor may receive a message from a processor in another segment. Using the index of the writer, the lower indexed processor can determine that the writer was not in the expected range, so its partner is faulty. Once a working processor determines that it is paired with a faulty processor, the working processor continues on to the next phase. After $\log N$ phases, the head of the array broadcasts the total, so that each processor can then locally determine the number of stuck delay switches ahead of it on the bus. The prefix sums can be computed in $O(\log N)$ steps.

Example: Figure 7.2 shows which processors communicate during the execution of ALL PROCESSORS PREFIX SUMS for the example given in Figure 7.1. For instance, since R_5 is faulty, R_4 takes its place in the following phases as shown in the figure. Also, in the first phase, when R_5 is supposed to segment the bus and write, R_4 will actually receive the message from R_7. Then, when R_4 writes, its message will reach R_7, but will be ignored.

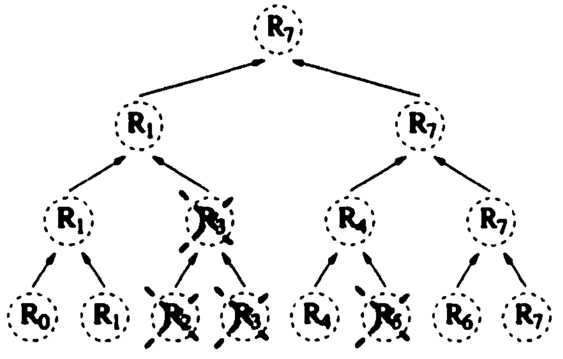


Figure 7.2: Communication steps to perform prefix sums

7.2.2 Determine Mapping

The next item to consider is the mapping of all processors to working processors, since each good processor will need to simulate up to two processors. Two different methods exist. The first is a ranked mapping and the second is a compaction mapping. The algorithms presented in this paper all use compaction mapping. The algorithms for a constant number of faults (Section 7.4), however, can use either mapping.

A *ranked mapping* is one in which the i^{th} working processor simulates the i^{th} faulty processor. In this method, each working processor always simulates itself as well as possibly one faulty processor.

Compaction mapping differs such that the i^{th} working processor simulates processors with indices $2i$ and $2i + 1$, for $i < f$, where f is the total number of faults. The remaining working processors simulate the processor with index $i+f$. In this method, each working processor simulates up to two processors; it is possible, however, that neither of the two simulated is itself.

To perform the compaction mapping, the LARPBS ranks all fault-free processors. Set the data value to '1' for each good processor and perform All Processors Prefix Sums in $O(\log N)$ steps. With this ranking, each working processor can determine which processor(s) it simulates.

Referring to the example in Figure 7.1, the resulting mapping would be as follows:

- R_0 simulates R_0 and R_1
- R_1 simulates R_2 and R_3
- R_4 simulates R_4 and R_5
- R_6 simulates R_6
- R_7 simulates R_7

Combining the time to determine information on the number of stuck delay switches and to determine the mapping results provides us with the following result.

Theorem 7.1 *An N-processor LARPBS with up to $N/2$ faults is able to compute the number of stuck delay switches succeeding each working processor and determine the mapping of all processors to working processors in a total of $O(\log^2 N)$ preprocessing steps.*

It is important to note that the preprocessing stage is not necessary before execution of each algorithm. If the LARPBS is to execute a sequence of algorithms, it need only perform preprocessing once. Once the mapping and information on the number of stuck delays has been established, it will apply to all algorithms run thereafter on the LARPBS.

7.3 Fault Tolerant Algorithms

In this section we describe some basic algorithms for an N-processor LARPBS that can tolerate up to $N/2$ faults. The basic algorithms considered are prefix sums, compression, sorting, and permutation routing. Using these fundamental algorithms, we can then extend the results to develop other more complex fault tolerant algorithms for the LARPBS, such as median row, image area and perimeter, histogram, and matrix transposition and multiplication.

After the preprocessing is complete, each healthy processor has determined the number of stuck delay switches ahead of it on the array, its ranking among healthy processors, and the indicies of the processors it is simulating. In spite of having this information available, it is still necessary to develop algorithms designed specifically for instances when faults are present. Algorithms for a fault-free LARPBS depend on the ability to set conditional delay switches. If a healthy processor sets its conditional delay switch to cross, then a message sent by a healthy processor could possibly land at a faulty processor. The index of this faulty processor could not be identified in constant time, therefore, alternate algorithms are necessary.

7.3.1 Fundamental Algorithms

The first algorithm we consider is the prefix sums of N elements on an N-processor LARPBS. We are not able to use the standard LARPBS prefix sums algorithm as described in Section 2.3.1, because messages may arrive at faulty processors. In this case, the ranking of the healthy processors determines which processors communicate with each other; this results in only working processors attempting to communicate. In contrast, ALL PROCESSORS PREFIX SUMS used the indices of the processors for determining which processors participate in a specific step. This results in all processors attempting to communicate, rather than just the working processors. With the ranking of the working processors known, as well as the number of stuck delays ahead of each processor, it is possible perform the operation in $O(\log N)$ steps.

Theorem 7.2 *Prefix sums of N elements can be computed on an N-processor LARPBS with up to $N/2$ faults in $O(\log N)$ steps.*

<u>Proof:</u> First, each good processor locally determines the total sum for the one or two elements it is simulating. Next, using the rankings of the good processors, perform prefix sums as in ALL PROCESSORS PREFIX SUMS. Since only healthy processors are participating, there is no need to check for a faulty partner. Each healthy processor is able to determine from its ranking whether or not it should segment the bus. Then each communication phase is performed in two steps. In the first, the lower ranked processor broadcasts its message on the subarray, and in the second, the higer ranked processor broadcasts its message. Once the prefix sums is complete, each working processor can locally determine the prefix sum for each of the elements it is simulating. ■

Figure 7.3 shows the processors involved during each step of the prefix sums operation for the system shown in Figure 7.1. For example, processor R_1 participates in the operation by simulating faulty processors R_2 and R_3. Also, R_7 does not exchange data with any other processor until the third phase of steps, since it is the fifth and last ranked working processor out of a possible eight processors.

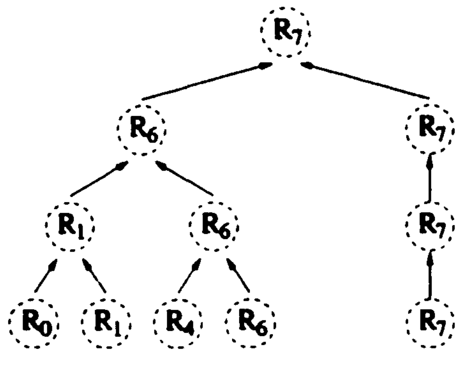


Figure 7.3: Communication phases for prefix sums on a faulty LARPBS

Recall from Section 2.3.2 that the compression algorithm shifts all marked elements to the lower end of the array and unmarked elements to the upper end of the array maintaing the original order.

Theorem 7.3 *Compression of x elements, where $x \leq N$, can be performed on an N-processor LARPBS with up to $N/2$ faults in $O(\log N)$ steps.*

Proof: First the working processors rank the marked processors, using the prefix sums algorithm of the previous theorem, in $O(\log N)$ steps. Call this the marked rank. The processor with marked rank i determines the index of the processor simulating p_i and routes its data to that processor.

Each working processor holds the indices of only the processors it is simulating. It does not hold the indices of the faulty and healthy processors, therefore, it is not able to easily determine which processor is simulating a specific processor. The method for the processor with marked rank i to determine the index of the processor simulating p_i is described below.

The processor, p_k, with marked rank $x/2$ broadcasts its index to all processors. Next, the processor simulating processor $p_{x/2}$ broadcasts its index, j, to all processors. As a result, all processors receive the index of the processor simulating the processor with the middle rank. Next, the processor with marked rank $x/4$ ($3x/4$) multicasts its index to $p_0, p_1, \ldots, p_{j-1}$ ($p_{j+1}, p_{j+2}, \ldots, p_{N-1}$). Similar to the previous phase, the processor simulating $p_{x/4}$ ($p_{3x/4}$) multicasts its index to the segment of processors below (above) p_k. Repeat this phase $\log x$ times, until all ranked processors can determine the corresponding indices.

Refer to Figure 7.4 to see the communication steps for the first two iterations of a sixteen processor array with five faulty processors and seven marked elements. In the first iteration, p_6 broadcasts its index since it simulates p_8 which holds the element with the middle rank of three. Processor p_2 then broadcasts its index since it simulates p_3. At this point, processors holding an element with rank below three determine that the final destination will be p_2 or below. Processors holding an element with rank above three determine that the final destination will be p_2 or above. During the second iteration, processors simulating p_4 (rank 1) and p_{11} (rank 5) multicast their indicies below and above p_2 respectively. Next, processors simulating p_1 and p_5 multicast their indicies in the corresponding subarrays. The procedure continues for $\log x$ iterations, for x marked elements.

Repeat these steps to compress data in unmarked processors to the right end of the LARPBS. These processors will determine the indices of processors starting after the last ranked processor in the previous phase, however. Once all indices of the simulating processors have been determined, send messages in two steps. First, send messages destined for an even numbered simulated processor, then those for odd numbered simulated processors. Recall that each working processor simulates up to two processors with consecutive indices. Therefore, routing messages this way will prevent messages from colliding at any processor, since at most one message will be destined for a particular processor at each step. ■

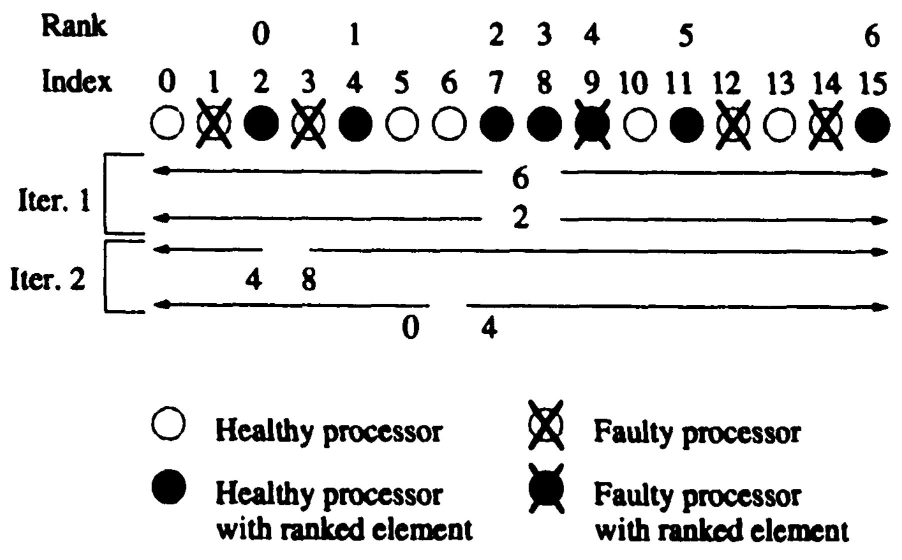


Figure 7.4: Communication phases for compression on a faulty LARPBS

Theorem 7.4 *Sorting N k-bit integers can be performed on an N-processor LARPBS with up to $N/2$ faults in $O(k \log N)$ steps.*

Proof: We use the radix sort method and the compression algorithm to sort the N integers [56]. The algorithm proceeds in k phases, one for each bit position of the integers. During execution of phase j, where $j \leq k$, perform compression based upon the j^{th} bit position (Theorem 7.3). Each phase takes $O(\log N)$ steps, for a total of $O(k \log N)$ steps. ■

A generalized permutation routing step is one in which each processor sends at most one message and is the intended destination for at most one message.

Theorem 7.5 *Any generalized permutation routing step can be performed on an N-processor LARPBS with up to $N/2$ faults in $O(\log^2 N)$ steps.*

Proof: The LARPBS first sorts the messages by their destinations in $O(\log^2 N)$ steps (Theorem 7.4). Since some processors may not be receiving messages, the

Table 7.1: Fault Tolerant LARPBS Algorithms

Algorithm	Time on Faulty	Time on Fault-Free	No. of Processors
median row	$O(\log N)$	$O(1)$	$O(N)$
image area	$O(\log^2 N)$	$O(1)$	$O(N)$
image perimeter	$O(\log^2 N)$	$O(1)$	$O(N)$
histogram	$O(\log h \log N)$	$O(\log h)$	$O(N)$
matrix transposition	$O(\log^2 N)$	$O(1)$	$O(N^2)$
matrix multiplication	$O(N \log^2 N)$	$O(N)$	$O(N^2)$

messages are in order after the sort, but not necessarily at their final destinations, so the LARPBS will next shift the messages to the intended processors. Perform the algorithm in two phases, one for messages destined to even numbered processors, and one for messages destined to odd numbered processors.

To perform the shifting, the processors holding the messages before the shifting determine the indices of the destination processors. Since all messages are in proper order, we can proceed in $O(\log N)$ phases broadcasting the indices of midpoints of segments, as in the compression algorithm (Theorem 7.3). The algorithm runs in $O(\log^2 N)$ steps. ∎

7.3.2 Extended Algorithms

We extend the results from the previous subsection to apply to other algorithms in the areas of image processing and matrix operations. Table 7.1 lists the algorithms considered, the time complexity on a faulty and a fault-free LARPBS, and the number of processors required. The algorithms listed tolerate at most $N/2$ faults for an N-processor LARPBS. Our fault tolerant algorithms combine the techniques of the previous fundamental algorithms presented and build upon existing algorithms for the LARPBS. The image processing algorithms follow the approach of Pan and Li [56]. The matrix operation algorithms follow the approach of Li *et al.* [39, 40].

Specifically, the median row, area, and perimeter algorithms make use of the binary prefix sums algorithm. The histogram algorithm utilizes the sorting and binary prefix sums algorithms. The matrix multiplication algorithm consists of multiple phases of the permutation routing algorithms along with local computations, while the matrix transposition uses the general permutation routing algorithm once.

7.4 Constant Number of Faults

Consider an LARPBS of N processors in which a constant number of processors are faulty, say f. The algorithms discussed earlier will apply here, but it is possible to do better utilizing the limit on faults to a constant number.

To begin preprocessing steps, each working processor determines if its neighbors are faulty in the same manner as in Section 7.1. Next, each processor needs to determine the number of fixed delay switches ahead of it on the bus. Each processor sends a message with its index to itself. If it did not receive its own message, then shift the select frame by one to the right and repeat. This may need to be repeated $f + 1$ times. Once a processor receives its own message, it then knows how many fixed delays are ahead of it on the bus. Call this d_i for processor R_i. To compensate for the stuck delays in future steps, each processor shifts its reference pulse by d_i to the left and does not alter its select frame.

Once the preprocessing is complete, each healthy processor keeps a table listing the faulty processors and the working processors that are simulating them. The algorithms then run as required, with a constant number of straightforward steps to accommodate the faulty processors. Each communication step is executed in the following four phases:

- Good to good
- Good to faulty
- Faulty to good
- Faulty to faulty

Separating each communication step into these four phases ensures that each processor is the actual destination for at most one message in a single bus cycle.

Lemma 7.6 *Any algorithm executed on an N-processor LARPBS with $O(1)$ faults will result in a constant factor slowdown.*

Chapter 8

Conclusions

The aim of this dissertation is to further demonstrate the claim that pipelined optical models are powerful parallel architectures and to show how these models fit into the well established hierarchy of complexity classes. We accomplished this by developing simulations relating different optical models to one another and also by developing more efficient algorithms and algorithms that considered certain physical restrictions.

In Chapter 4 we established the equivalence of three one-dimensional optical models, namely the LARPBS, LPB, and POB. We first developed an algorithm to compute binary prefix sums without using the segmenting ability of the LARPBS. This algorithm is instrumental in developing a cycle of simulations among the three models, as both the LPB and POB do not have segment switches. The equivalence establishes reconfigurable delay (rather than the segmenting ability) as the key to the power of optically pipelined buses. This separation of the powers of segmentation and delays is similar to that established in the context of the RMBM [74].

The equivalence established provides us with the ability to efficiently translate algorithms designed for any of these models to any other regardless of their structure differences. It would be beneficial to consider other one-dimensional optical models and determine their relations to the LARPBS. The LAROB and LAPOB are examples of other one-dimensional models to consider.

In Chapter 5 we introduced the PR-Mesh, a k-dimensional extension of the LARPBS, and established that the PR-Mesh has the same complexity as the LR-Mesh. This relation differs from the equivalence relations of the one-dimensional models of Chapter 4. Here we relate time and processor-bounded complexity classes for these models. Essentially, any step of the PR-Mesh can be simulated by the LR-Mesh or vice versa within a constant number of steps allowing a polynomial increase in processors. We also prove that the PR-Mesh can solve the same class of problems as the LR-Mesh within the same order of steps using polynomial processors. We extend this complexity class to include two other optical models, the AROB and APPBS.

This result allows us to translate algorithms from one model to another and also helps to unify existing research on reconfigurable optical models. The relations also distinguish capabilities and limitations of these models by placing the models into an established complexity class.

An open problem that involves establishing relations among models is the relation between the LARPBS and PR-Mesh. It does not seem likely that the LARPBS is as powerful as the PR-Mesh due to the steps required to perform list ranking along a bus. The LARPBS may be more powerful than the HV-RN, since it is not known if the HV-RN can compute prefix sums in a constant number of steps. (The HV-RN is a restricted version of the R-Mesh in which only horizontal and vertical buses are allowed.) It may be possible, however, to place the LARPBS into a class that lies between the LR-Mesh and HV-RN. There are three types of simulations we could consider: i) fix the number of processors to be the same and determine the number of steps required by the LARPBS to simulate the LR-Mesh, ii) determine the number of processors required for the LARPBS to simulate the LR-Mesh to within a constant factor of the same number of steps, and iii) allow an $O(\log N)$ factor increase in steps and determine the number of processors required. The same could be done between the LARPBS and HV-RN.

In Chapter 6 we developed algorithms in the areas of arithmetic analysis, computational geometry, and image analysis. Some of these algorithms are more efficient than other existing algorithms, in the sense that there is a reduction in either time and/or size. Some of the algorithms generalize existing algorithms to accommodate arbitrary word sizes.

We also developed algorithms to compute binary prefix sums and perform compression that limit the communication distance between two processors. This is an important consideration when evaluating practical implications. For instance, without restricting communication distances, additional hardware, such as repeaters or optical amplifiers, may become necessary, thus increasing the size and cost of the systems.

Consideration of other physical constraints could lead to further algorithm development. One example is, rather than limiting the communication distance, one could limit the bus length. If this is considered, then a natural direction is the development of scalable algorithms. Currently, few papers consider restricted bus length for reconfigurable models [7, 15, 35], despite cost and space limitation factors motivating this research.

Rather than focusing only on constraints, it is desirable to develop algorithms for a more generalized system. Thus far, all algorithms developed for reconfigurable models have assumed that a healthy system is available. For practical purposes this is not a reasonable assumption, therefore, in Chapter 7 we developed algorithms that can tolerate up to $N/2$ faults on an N-processor LARPBS. In particular, we present fault-tolerant algorithms to compute binary prefix sums, perform compression, sorting, and a general permutation routing. We then use these fundamental algorithms as building blocks to develop more extensive algorithms in the areas of image analysis and matrix operations. There are many other problems for pipelined optical models that do not yet have fault tolerant algorithms.

Bibliography

[1] B.F.A. AlMohammad and B. Bose, "Fault-Tolerant Communication Algorithms in Toroidal Networks," *IEEE Trans. Parallel Distrib. Systems*, vol. 10, (1999), pp. 976–983.

[2] Y. Aumann and M. Ben-Or, "Computing with Faulty Arrays," *Proc. 24th Ann. ACM STOC*, (1992), pp. 162–169.

[3] Y. Azar and U. Vishkin, "Tight Comparison Bounds on the Complexity of Parallel Sorting," *SIAM J. Comput.*, vol. 16, (1987), pp. 458–464.

[4] Banerjee, Rahmeh, Stunkel, Nair, Roy, Balasubramanian, and Abraham, "Algorithm-Based Fault Tolerance on a Hypercube Multiprocessor," *IEEE Trans. Comput.*, vol. 39, (1990), pp. 1132–1144.

[5] Y. Ben-Asher, K. J. Lange, D. Peleg, and A. Schuster, "The Complexity of Reconfiguring Network Models," *Information and Computation*, vol. 121, (1995), pp. 41–58.

[6] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," *J. Parallel Distrib. Comput.*, vol. 13, (1991), pp. 139–153.

[7] B. Beresford-Smith, O. Diessel, and H. ElGindy, "Optimal Algorithms for Constrained Reconfigurable Meshes," *J. Parallel Distrib. Comput.*, vol. 39, (1996), pp. 74–78.

[8] A. G. Bourgeois and J. L. Trahan, "Fault Tolerant Algorithms for a Linear Array with a Reconfigurable Pipelined Bus System," to appear in *Proc. Wkshp. on Optics and Comp. Sc.*, (2000).

[9] A. G. Bourgeois and J. L. Trahan, "Relating Two-Dimensional Reconfigurable Meshes with Optically Pipelined Buses," to appear in *Int'l. J. on Found. of Comp. Sc.*

[10] A. G. Bourgeois and J. L. Trahan, "Relating Two-Dimensional Reconfigurable Meshes with Optically Pipelined Buses," to appear in *Proc. Int'l. Par. and Distr. Process. Symp.*, (2000).

[11] H.-L. Chen and S.-H. Hu, "Distributed Submesh Determination in Faulty Tori and Meshes," *Proc. Int'l. Par. Processing Symp.*, 1997.

[12] G.-M. Chiu and S.-P. Wu, "A Fault-Tolerant Routing Strategy in Hypercube Multicomputers," *IEEE Trans. Comput.*, vol. 45, (1996), pp. 143–154.

[13] D. M. Chiarulli, S. P. Levitan, R. G. Melhem, M. Bidnurkar, R. Ditmore, G. Gravenstreter, Z. Guo, C. Qiao, M. Sakr, and J. P. Teza, "Optoelectronic Buses for

High-Performance Computing," *Proceedings of IEEE*, vol. 82, (1994), pp. 1701–1709.

[14] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Electrical Engineering and Computer Science Series MIT Press, Cambridge, MA, 1990.

[15] O. Diessel, H. ElGindy, and L. Wetherall, "Efficient Broadcasting Procedures for Constrained Reconfigurable Meshes," Technical Report 96-07, Dept. of Comp. Sci. and Software Engr., Univ. of Newcastle, Australia.

[16] H. ElGindy, "Improved Convex Hull Algorithm on a Processor Array with a Reconfigurable Bus System," Technical report No. SOCS 91.12, School of Computer Science, McGill Univ.

[17] H. ElGindy, "A Sparse Matrix Multiplication Algorithm for the Reconfigurable Mesh Architecture," Technical Report 96-08, Dept. of Comp. Sci. and Software Engr., Univ. of Newcastle, Australia.

[18] H. ElGindy, "An Improved Sorting Algorithm for Linear Arrays with Optical Buses," Manuscript, 1998.

[19] H. ElGindy and L. Wetherall, "A Simple Voronoi Diagram Algorithm for a Reconfigurable Mesh," *IEEE Trans. Parallel Distrib. Systems*, vol. 8, (1997), pp. 1133–1142.

[20] J. A. Fernández-Zepeda, J. L. Trahan, and R. Vaidyanathan, "Scaling the FR-Mesh under Different Concurrent Write Rules," *Proc. World Multiconf. on Systemics, Cybernetics, and Informatics*, (1997), pp. 437–444.

[21] J. A. Fernández-Zepeda, R. Vaidyanathan, and J. L. Trahan, "Improved Scalability Simulations of the General Reconfigurable Mesh," *Proc. 6th Reconfigurable Architecture Workshop*. LNCS vol. 1586, April 1999, pp. 616-624.

[22] J. A. Fernández-Zepeda, R. Vaidyanathan, and J. L. Trahan, "Scaling Simulation of the Fusing-Restricted Reconfigurable Mesh," *IEEE Trans. Parallel Distrib. Systems*, vol. 9, (1998), pp. 861–871.

[23] Z. Guo, "Sorting on Array Processors with Pipelined Buses," *Proc. Int'l. Conf. Par. Processing*, (1992), pp. 289–292.

[24] Z. Guo, "Optically Interconnected Processor Arrays with Switching Capability," *J. Parallel Distrib. Comput.*, vol. 23, (1994), pp. 314–329.

[25] Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, "Array Processors with Pipelined Optical Busses," *J. Parallel Distrib. Comput.*, vol. 12, (1991), pp. 269–282.

[26] M. Hamdi and Y. Pan, "Efficient Parallel Algorithms on Optically Interconnected Arrays of Processors," *IEE Proceedings - Computers and Digital Techniques*, vol. 142, (1995), pp. 87–92.

[27] T. Hayashi, K. Nakano, and S. Olariu, "An $O((\log\log n)^2)$ Time Convex Hull Algorithm on Reconfigurable Meshes," *IEEE Trans. Parallel Distrib. Systems*, vol. 9, (1998), pp. 1167–1179.

[28] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Co., 1992.

[29] J. Jang, H. Park, and V. K. Prasanna, "A Fast Algorithm for Computing a Histogram on a Reconfigurable Mesh," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 17, (1995), pp. 97–105.

[30] D. S. Johnson, "A Catalog of Complexity Classes," in *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, J. van Leeuwen, ed., MIT Press, (1990), pp. 67–162.

[31] R. M. Karp and V. Ramachandran, "Parallel Algorithms for Shared-Memory Machines," in *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, J. van Leeuwen, ed., MIT Press, (1990), pp. 869–941.

[32] H. Kim and Y. Cho, "Point Visiblility of a Simple Polygon on Reconfigurable Mesh," *Proc. Proc. 5th IEEE Symp. Par. and Distr. Proc.*, (1993), pp. 748–751.

[33] M. Kim and J. Jang, "Fast Quadtree Building on a Reconfigurable Mesh," *Proc. 3rd Workshop on Reconfig. Arch. and Algs.*, 1996.

[34] S. -R. Kim and K. Park, "Fully-Scalable Fault-Tolerant Simulations for BSP and CGM," *Proc. 13th Int'l. Par. Process. Symp. & 10th Symp. Par. Distr. Process.*, (1999), pp. 117–124.

[35] M. Kunde and K. Gürtzig, "Efficient Sorting and Routing on Reconfigurable Meshes Using Restricted Bus Length," *Proc. Int'l. Par. Processing Symp.*, 1997.

[36] S. -S. Lee, S. -J. Horng, H. -R. Tsai, and S. -S. Tsai, "Building a Quadtree and Its Applications on a Reconfigurable Mesh," *Pattern Recognition*, vol. 29, (1996), pp. 1571–1579.

[37] T. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. Comput.*, vol. 34, (1985), pp. 344–354.

[38] K. Li, Y. Pan, and S. Q. Zheng, *Parallel Computing Using Optical Interconnections*, Kluwer Academic Publishers, Boston, MA, 1998.

[39] K. Li, Y. Pan, and S. Q. Zheng, "Fast and Efficient Parallel Matrix Operations Using a Linear Array with a Reconfigurable Pipelined Bus System," in *High Performance Computing Systems and Applications*, J. Schaeffer and R. Unrau, eds., Kluwer Academic Publishers, Boston, MA, 1998.

[40] K. Li, Y. Pan, and S. Q. Zheng, "Fast and Efficient Parallel Matrix Multiplication Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *IEEE Trans. Parallel Distrib. Systems*, vol. 9, (1998), pp. 705–720.

[41] K. Li, Y. Pan, and S. Q. Zheng, "Simulation of Parallel Random Access Machines on Linear Arrays with Reconfigurable Pipelined Bus Systems," *Proc. Int'l. Conf. Par. Distr. Proc. Tech. and App.*, (1997), pp. 590–599.

[42] Y. Li, Y. Pan, and S. Q. Zheng, "Pipelined TDM Optical Bus with Conditional Delays," *Optical Engineering*, vol. 36, (1997), pp. 2417-2424.

[43] Y. Li and S. Q. Zheng, "Parallel Selection on a Pipelined TDM Optical Bus," *Proc. Int'l. Conf. Par. Distr. Comput. and Sys.*, (1996), pp. 69-73.

[44] R. Lin and S. Olariu, "Reconfigurable Buses with Shift Switching: Concepts and Applications," *IEEE Trans. Parallel Distrib. Systems*, vol. 6, (1995), pp. 93-102.

[45] Y. Matias and A. Schuster, "Fast, Efficient Mutual and Self Simulations for Shared Memory and Reconfigurable Mesh," *Par. Algs. and Appl.*, vol. 8, (1996), pp. 195-221.

[46] R. Melhem, D. Chiarulli, and S. Levitan, "Space Multiplexing of Waveguides in Optically Interconnected Multiprocessor Systems," *The Computer Journal*, vol. 32, (1989), pp. 362-369.

[47] M. Middendorf and H. ElGindy, "Matrix Multiplication on Processor Arrays with Optical Buses," to appear in *Informatica*.

[48] K. Nakano, "A Bibliography of Published Papers on Dynamically Reconfigurable Architectures," *Parallel Proc. Letters*, vol. 5, (1995), pp. 111-124.

[49] M. Nigam and S. Sahni, "Sorting n Numbers on an $n \times n$ Reconfigurable Meshes with Buses," *J. Parallel Distrib. Comput.*, vol. 23, (1994), pp. 37-48.

[50] M. Nigam and S. Sahni, "Computational Geometry on a Reconfigurable Mesh," *Proc. 8th Int'l. Par. Proc. Symp.*, (1994), pp. 86-93.

[51] M. Nigam and S. Sahni, "Triangulation on a Reconfigurable Mesh with Buses," *Proc. Int'l. Conf. on Par. Proc.*, (1994), pp. 251-257.

[52] S. Olariu, J. Schwing, and J. Zhang, "Applications of Reconfigurable Meshes to Constant Time Computations," *Par. Comput.*, vol. 19, (1993), pp. 229-237.

[53] Y. Pan, "Hough Transform on Arrays with an Optical Bus," *Proc. 5th Int'l. Conf. Par. Distr. Comput. and Sys.*, (1992), pp. 161-166.

[54] Y. Pan, "Order Statistics on a Linear Array with a Reconfigurable Bus," *Future Generation Computer Systems*, vol. 11, (1995), pp. 321-328.

[55] Y. Pan and M. Hamdi, "Singular Value Decomposition on Processor Arrays with a Pipelined Bus System," *J. Network and Computer Appl.*, vol. 19, (1996), pp. 235-248.

[56] Y. Pan and K. Li, "Linear Array with a Reconfigurable Pipelined Bus System: Concepts and Applications," *Information Sciences - An International Journal*, vol. 106, (1998), pp. 237-258.

[57] Y. Pan, K. Li, and S. Q. Zheng, "Fast Nearest Neighbor Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," to appear in *Parallel Algorithms and Applications*.

[58] B. Parhami, "Fault Tolerance Properties of Mesh-Connected Parallel Computers with Separable Row/Column Buses," *Proc. Midwest Symp. on Cir. and Syst.*, (1993), pp. 1128–1131.

[59] B. Parhami and C.-H. Yeh, "The Robust-Algorithm Approach to Fault Tolerance on Processor Arrays: Fault Models, Fault Diameter, and Basic Algorithms," *Proc. Int'l. Par. Processing Symp.*, (1998), pp. 742–746.

[60] H. Park, V. K. Prasanna, and J. Jang, "Fast Arithmetic on Reconfigurable Meshes," *Proc. Int'l. Conf. Par. Processing*, (1993), pp. 236–243.

[61] S. Pavel and S. G. Akl, "Integer Sorting and Routing in Arrays with Reconfigurable Optical Buses," *Int'l. J. Foundations of Computer Science*, vol. 9, (1998), pp. 99–120.

[62] S. Pavel and S. G. Akl, "Matrix Operations Using Arrays with Reconfigurable Optical Buses," *Par. Algs. and Appl.*, vol. 8, (1996), pp. 223–242.

[63] S. Pavel and S. G. Akl, "On the Power of Arrays with Optical Pipelined Buses," *Proc. Int'l. Conf. Par. Distr. Proc. Techniques and Appl.*, (1996), pp. 1443–1454.

[64] S. Pavel and S. G. Akl, "Efficient Algorithms for the Hough Transform on Arrays with Reconfigurable Optical Buses," *Proc. Int'l. Par. Processing Symp.*, (1996), pp. 697–701.

[65] C. Qiao, "On Designing Communication-Intensive Algorithms for a Spanning Optical Bus Based Array," *Parallel Proc. Letters*, vol. 5, (1995), pp. 499–511.

[66] C. Qiao and R. Melhem, "Time-Division Optical Communications in Multiprocessor Arrays," *IEEE Trans. Comput.*, vol. 42, (1993), pp. 577–590.

[67] C. Qiao, R. Melhem, D. Chiarulli, and S. Levitan, "Optical Multicasting in Linear Arrays," *Int'l. J. Optical Computing*, vol. 2, (1991), pp. 31–48.

[68] S. Rajasekaran and S. Sahni, "Sorting, Selection and Routing on the Arrays with Reconfigurable Optical Buses," *IEEE Trans. Parallel Distrib. Systems*, vol. 8, (1997), pp. 1123–1132.

[69] A. G. Ranade, "How to Emulate Shared Memory," *J. Comput. System Sci.*, vol. 42, (1991), pp. 301–324.

[70] J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "Optimally Scaling Permutation Routing on Reconfigurable Arrays with Optically Pipelined Buses," *Proc. 13th Int'l. Par. Process. Symp. & 10th Symp. Par. Distr. Process.*, (1999), pp. 233–237.

[71] J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "An Optimal and Scalable Permutation Routing Algorithm for Reconfigurable Linear Arrays with Optically Pipelined Buses," submitted to *J. Parallel Distrib. Comput.*

[72] J. L. Trahan, A. G. Bourgeois, and R. Vaidyanathan, "Tighter and Broader Complexity Results for Reconfigurable Models," *Parallel Proc. Letters*, vol. 8, (1998), pp. 271–282.

[73] J. L. Trahan, Y. Pan, R. Vaidyanathan, and A. G. Bourgeois, "Scalable Basic Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *Proc. Int'l. Conf. on Parallel and Distributed Computing Systems*, (1997), pp. 564–569.

[74] J. L. Trahan, R. Vaidyanathan, and R. K. Thiruchelvan, "On the Power of Segmenting and Fusing Buses," *J. Parallel Distrib. Comput.*, vol. 34, (1996), pp. 82–94.

[75] R. Vaidyanathan and J. L. Trahan, "Optimal Simulation of Multidimensional Reconfigurable Meshes by Two-Dimensional Reconfigurable Meshes," *Info. Proc. Letters*, vol. 47, (1993), pp. 267–273.

[76] T. A. Varvarigou, V. P. Roychowdhury, and T. Kailath, "Reconfiguring Processor Arrays Using Multiple-Track Models: The 3-track-1-spare-approach," *IEEE Trans. Comput.*, vol. 42, (1993), pp. 1281–1293.

[77] B. F. Wang and G. H. Chen, "Constant Time Algorithms for the Transitive Closure and Some Related Graph Problems on Processor Arrays with Reconfigurable Bus Systems," *IEEE Trans. Parallel Distrib. Systems*, vol. 1, (1990), pp. 500–507.

[78] C. -H. Yeh, B. Parhami, H. Lee, and E. A. Varvarigos, "2.5n-Step Sorting on $n \times n$ Meshes in the Presence of $o(\sqrt{n})$ Worst-Case Faults," *Proc. 13th Int'l. Par. Process. Symp. & 10th Symp. Par. Distr. Process.*, (1999), pp. 436–440.

[79] S. Q. Zheng and Y. Li, "Pipelined Asynchronous Time-Division Multiplexing Optical Bus," *Optical Engineering*, vol. 36, (1997), pp. 3392–3400.

Vita

Anu Goel Bourgeois was born in Baton Rouge, Louisiana, on December 11, 1969. She received her bachelor's degree in electrical engineering from Louisiana State University (LSU) in 1991. She worked as a consulting engineer from 1991 until 1994. She is currently a doctoral student in the Department of Electrical and Computer Engineering at LSU. Since 1996, she has been a Dean's Fellow and is expecting to complete her degree in May 2000. Her current research interests include parallel processing, algorithm design and analysis, reconfigurable models, and pipelined optical networks, in which she has a number of conference and journal publications. She will receive the degree of Doctor of Philosophy in May, 2000.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Anu Goel Bourgeois

Major Field: Electrical Engineering

Title of Dissertation: Simulations and Algorithms on Reconfigurable Meshes with Pipelined Optical Buses

Approved:

[illegible]

Major Professor and Chairman

[illegible]

Dean of the Graduate School

EXAMINING COMMITTEE:

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Date of Examination:

March 21, 2000