

Spring 5-31-1997

## Efficient parallel processing with optical interconnections

Lili Hai  
*New Jersey Institute of Technology*

Follow this and additional works at: <https://digitalcommons.njit.edu/dissertations>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Hai, Lili, "Efficient parallel processing with optical interconnections" (1997). *Dissertations*. 1044.  
<https://digitalcommons.njit.edu/dissertations/1044>

This Dissertation is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Dissertations by an authorized administrator of Digital Commons @ NJIT. For more information, please contact [digitalcommons@njit.edu](mailto:digitalcommons@njit.edu).

## **Copyright Warning & Restrictions**

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

**Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation**

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

## ABSTRACT

### EFFICIENT PARALLEL PROCESSING WITH OPTICAL INTERCONNECTIONS

by  
Lili Hai

With the advances in VLSI technology, it is now possible to build chips which can each contain thousands of processors. The efficiency of such chips in executing parallel algorithms heavily depends on the interconnection topology of the processors. It is not possible to build a fully interconnected network of processors with constant fan-in/fan-out using electrical interconnections. Free space optics is a remedy to this limitation. Qualities exclusive to the optical medium are its ability to be directed for propagation in free space and the property that optical channels can cross in space without any interference. In this thesis, we present an electro-optical interconnected architecture named Optical Reconfigurable Mesh (ORM). It is based on an existing optical model of computation. There are two layers in the architecture. The processing layer is a reconfigurable mesh and the deflecting layer contains optical devices to deflect light beams. ORM provides three types of communication mechanisms. The first is for arbitrary planar connections among sets of locally connected processors using the reconfigurable mesh. The second is for arbitrary connections among  $N$  of the processors using the electrical buses on the processing layer and  $N^2$  fixed passive deflecting units on the deflection layer. The third is for arbitrary connections among any of the  $N^2$  processors using the  $N^2$  mechanically reconfigurable deflectors in the deflection layer. The third type of communication mechanisms is significantly slower than the other two. Therefore, it is desirable to avoid reconfiguring this type of communication during the execution of the algorithms. Instead, the optical reconfiguration can be done before the execution of each algorithm begins. Determining a right configuration that would be suitable

for the entire configuration of a task execution is studied in this thesis. The basic data movements for each of the mechanisms are studied. Finally, to show the power of ORM, we use all three types of communication mechanisms in the first  $O(\log N)$  time algorithm for finding the convex hulls of all figures in an  $N \times N$  binary image presented in this thesis.

**EFFICIENT PARALLEL PROCESSING WITH OPTICAL  
INTERCONNECTIONS**

by  
**Lili Hai**

**A Dissertation  
Submitted to the Faculty of  
New Jersey Institute of Technology  
in Partial Fulfillment of the Requirements for the Degree of  
Doctor of Philosophy**

**Department of Computer and Information Science**

**May 1997**

Copyright © 1997 by Lili Hai

ALL RIGHTS RESERVED

## APPROVAL PAGE

### Efficient Parallel Processing with Optical Interconnections

iii

---

Dr. Mary M. Eshaghian, Dissertation Advisor	Date
Director of Advanced Computer Architecture and Parallel Processing Laboratory	
Assistant Professor of Computer and Information Science	
Assistant Professor of Electrical and Computer Engineering, NJIT	

---

Dr. John D. Carpinelli, Committee Member	Date
Associate Professor of Electrical and Computing Engineering	
Associate Professor of Computer and Information Science, NJIT	

---

Dr. Kurfess, Committee Member	Date
Co-Director of Software Engineering Laboratory	
Assistant Professor of Computer and Information Science, NJIT	

---

Dr. Peter A. Ng, Committee Member	Date
Chairperson of CIS Department	
Full Professor of Computer Science, NJIT	

---

Dr. Muhammad Shaaban, Committee Member	Date
Assistant Professor of Computer Engineering,	
Rochester Institute of Technology	



## BIOGRAPHICAL SKETCH

**Author:** Lili Hai  
**Degree:** Doctor of Philosophy  
**Date:** May 1997

### Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,  
New Jersey Institute of Technology, Newark, New Jersey, 1997
- Master of Application and Engineering in Computer Science,  
North China Institute of Technology, Beijing, China, 1983
- Bachelor of Science in Computer Science,  
Beijing University, Beijing, China, 1978

**Major:** Computer Science

### Publications:

- M. Eshaghian and L. Hai,  
“An Efficient Electro-Optical Parallel Architecture”,  
*To appear in Proceeding of International Symposium on Automotive Technology  
and Automation, Florence, Italy, June, 1997.*
- M. Eshaghian and L. Hai,  
“A Fast Parallel Image Convexity Algorithm”,  
*To appear in Proceeding of International Symposium on Automotive Technology  
and Automation, Florence, Italy, June, 1997.*
- M. Eshaghian and L. Hai,  
“Application Specific Design of the Optical Communication Topology in  
ORM”,  
*To appear in Proceeding of International Symposium on Automotive Technology  
and Automation, Florence, Italy, June, 1997.*
- M. Eshaghian and L. Hai,  
“Efficient Parallel Computing with an Optically Interconnected Reconfigurable  
Mesh”,  
*Submitted to Journal of Parallel Distributed Computing*

To my beloved family

## ACKNOWLEDGMENT

I would like to express my deepest appreciation to my advisor Dr. Mary Eshaghian, who not only served as my research supervisor, providing valuable and countless guidances, but also constant support, encouragement, reassurance and friendship.

Special thanks are given to Dr. Peter Ng, Dr. John Carpinelli, Dr. Franz Kurfess and Dr. Muhammad Shaaban for serving as members in my committee and offering invaluable suggestions to this dissertation.

Special recognition to many of my fellow graduate students in the CIS department, colleagues in my department of LUCENT Technologies Inc. and especially Dr. Song Chen and Dr. Jay Y. Wu for their support. In addition, I appreciate the works done by Amanda Haseltine for English editing on this thesis. I also wish to thank all the secretaries in CIS department for their assistance over the years.

# TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION . . . . .	1
1.1 VLSI Interconnection Topologies . . . . .	2
1.2 Ideal Parallel Computational Model . . . . .	4
1.3 Optical Interconnection Networks . . . . .	5
1.3.1 Why Optics in VLSI . . . . .	5
1.3.2 Previous Work . . . . .	7
1.3.3 Overview of the Thesis . . . . .	12
2 AN OPTICAL MODEL OF COMPUTATION . . . . .	14
2.1 OMC . . . . .	14
2.1.1 Definition . . . . .	14
2.1.2 Implementations . . . . .	16
2.2 Applications . . . . .	21
2.2.1 Optimal Geometric Algorithms . . . . .	21
2.2.2 Distance Problems . . . . .	23
2.2.3 Constant Time Geometric Algorithms . . . . .	24
2.2.4 A Special Purpose Design for Parallel Implementation of Iterative Methods for Higher Level Vision Processing . . . . .	25
2.2.5 Conclusion . . . . .	25
3 OPTICAL RECONFIGURABLE MESH . . . . .	26
3.1 Introduction . . . . .	26
3.2 The ORM Architecture . . . . .	28
3.2.1 The Processing Unit . . . . .	29
3.2.2 The Deflection Unit . . . . .	30
3.3 Data Movement in ORM . . . . .	31
3.3.1 Electrical Routing . . . . .	31

Chapter	Page
3.3.2 Optical Routing . . . . .	31
3.3.3 Electro-Optical Routing . . . . .	32
3.4 Conclusion . . . . .	37
4 A FAST PARALLEL IMAGE CONVEXITY ALGORITHM . . . . .	39
4.1 Introduction . . . . .	39
4.2 $O(\log N)$ Convexity Algorithm . . . . .	40
4.2.1 The Algorithm . . . . .	41
4.2.2 CONNECT: The Method to Find $i_{ceil}$ . . . . .	53
• 4.3 Conclusion . . . . .	56
5 APPLICATION SPECIFIC DESIGN OF THE OPTICAL COMMUNICATION TOPOLOGY IN ORM . . . . .	57
5.1 Introduction . . . . .	57
5.2 Preliminaries . . . . .	59
5.3 The CONST Methodology . . . . .	62
5.3.1 Phase 1: (Declustering) . . . . .	62
5.3.2 Phase 2: (Assigning) . . . . .	62
5.4 Conclusion . . . . .	68
6 CONCLUSION AND FUTURE WORKS . . . . .	69
APPENDIX A CLUSTER-M PRELIMINARIES . . . . .	72
REFERENCES . . . . .	87

## LIST OF FIGURES

Figure	Page
1.1 Optical crossbar system . . . . .	9
2.1 The OMC model . . . . .	15
2.2 An optical mesh using mirrors . . . . .	18
2.3 Reconfiguration using acousto-optic devices . . . . .	19
2.4 Reconfiguration using an electro-optical crossbar . . . . .	20
2.5 Merging of blocks and processor assignment . . . . .	23
3.1 The ORM architecture . . . . .	29
3.2 The detailed structure of a processing element . . . . .	30
3.3 The detailed structure of a deflector . . . . .	30
3.4 The optical interconnections for electro-optical routing of ORM . . . . .	33
3.5 Concurrent read on ORM . . . . .	37
4.1 Convexity Algorithm . . . . .	42
4.2 Function $REGION_2(i)$ . . . . .	43
4.3 The region divisions . . . . .	44
4.4 The region 1 in the figure with multi-most points . . . . .	45
4.5 Quadrants, the $v$ line and the $h$ line in a submesh of ORM . . . . .	47
4.6 P nodes: the cross points of tangents and line $y$ . . . . .	49
4.7 The example of the rule violation: $D(k) \geq D(k+1)$ . . . . .	50
4.8 The example of the rule violation: $D(k) \leq D(k+1)$ . . . . .	51
4.9 Drop the nonextreme points depending on $LN$ and $RN$ nodes . . . . .	52
4.10 Two pieces of figure 1 and two pieces of figure 2 in one submesh . . . . .	53
4.11 Find the route for each figure . . . . .	56
5.1 A children set and a children pool . . . . .	61
5.2 The optical connection to the header of children set . . . . .	64

<b>Figure</b>	<b>Page</b>
5.3 The example of the conflict . . . . .	65
A.1 Cluster-M mapping process. . . . .	73
A.2 Horizontal and vertical partitioning of a task graph. . . . .	74
A.3 Clustering Nonuniform Directed Graphs (CNDG) algorithm . . . . .	78
A.4 Clustering on a join-node: a general case . . . . .	81
A.5 Clustering on a fork-node: a general case . . . . .	81
A.6 A task graph and steps for obtaining the Spec graph . . . . .	82
A.7 Clustering Nonuniform Undirected Graphs (CNUG) algorithm. . . . .	85
A.8 A nonuniform system graph and its Rep graph. . . . .	85

# CHAPTER 1

## INTRODUCTION

Speedups due to technological advances in solid state electronic design are reaching theoretical limits. To get around these limits, researchers have considered concurrent processing of data as a promising alternative for achieving speedups proportional to the level of concurrency. Since the late 1970's, many multiprocessor architectures have been proposed to obtain such speedups. However, the desired speedups have not been realized because of a limited understanding of issues in designing efficient parallel algorithms and in designing interconnection networks and their interactions. During the last decade, many parallel algorithms have been designed based on a theoretical shared memory model, the Parallel Random Access Machine (PRAM) [66], in which a unit-delay interconnection network is assumed. Unfortunately, a realization of this machine does not exist.

Traditionally electronic interconnects have dominated the interconnection methods in parallel computers. In practice, electronic interconnection networks introduce a delay factor in the implementation of parallel algorithms. The main issues in the design of such interconnection networks have been routing delay, communication bandwidth, hardware cost and ease of control. However, a new technique for interconnecting processing elements in massively parallel computers is emerging: optical interconnection. Over the last ten years, various optical interconnection systems have been developed. The advantages of optical technology are realized and studied. This promising technique is playing a more and more important role in parallel computations. An introduction to the above issues is discussed in this chapter.



### 1.1 VLSI Interconnection Topologies

Research in the design of interconnection networks can be divided into two topological classes: static and dynamic [19]. In a static network, links between any two processors are passive and direct connections cannot be reconfigured between processors. In a dynamic network, links can be reconfigured by setting the switching elements in the network. Among many static topologies, those having smaller diameters are most attractive. The *diameter* of a network is the maximum distance between any pair of processors. The *distance* between a pair of processors is the smallest number of nodes that have to be traversed in order to get from one processor to the other. The diameter of an architecture represents a lower bound on worst case communication delay between any two processors. A fully connected network has unit diameter. However, when an  $N$ -processor system is implemented in electronic technology such as VLSI, its diameter becomes  $\Omega(\log N)$  due to pin-out limitations of processors if  $N$  is large. Also, the VLSI layout area becomes too large to be practically implemented. A balance has to be found among various network parameters, such as the node degree, the switching complexity, and the network latency. Therefore, an appropriate alternative is to consider area efficient architectures which have some degree of communication delay. The typical example is the hypercube topology in which there are  $\log N$  interconnections from each node to others such that the diameter is  $\log N$ . Pyramid and Mesh of Trees are other examples of such architectures [50, 34, 44].

Dynamic networks are categorized in three topological classes: single-stage, multistage, and crossbar [19]. The switches in an  $N \times N$  crossbar can be set in  $O(\log N)$  time so that every input port can be connected to a free output port. An  $N$  input crossbar requires  $\Omega(N^2)$  VLSI area, using the usual two dimensional VLSI model [59]. Several  $N$  input  $N$  output multistage networks are known which require  $O(N \log N)$  switching elements, significantly fewer than a crossbar network [56].

Multistage networks can be divided into two major classes: rearrangeable and non-rearrangeable [5]. Non-rearrangeable networks can realize only a proper subset of all permutations. A Butterfly network is a widely used non-rearrangeable network. The well known Omega network is a non-rearrangeable multistage network. It has  $O(N \log N)$  switching elements and requires  $\Omega(N^2 / \log^2 N)$  VLSI area [34]. Rearrangeable networks support any arbitrary permutation using appropriate switch settings. However, finding a switch setting to realize a permutation on a rearrangeable network can be time consuming; for example it can take as much as  $O(\log^4 N)$  time using a cube connected computer or a perfect shuffle computer with  $N$  processors [46]. Their layout area in the two-dimensional VLSI model is not significantly superior compared to the area requirement of the  $N$  input crossbar. Therefore, realizing multistage interconnection networks in  $O(N \log N)$  area does not seem possible unless one assumes that wires do not occupy any area.

From the theoretical computational point of view, for a given problem, there is a lower bound on the VLSI circuit area  $A$  on which the problem is run, and its computational time  $T$ . The lower bound in the VLSI model, called “ $AT^2$  bound”, is represented by the formula  $AT^2 = \Omega(I^2)$ , where  $I$  is the information content of the problem [59]. Through this lower bound, we can see that if the time  $T$  is fixed, the required VLSI area grows with the problem size. We will see in section 1.3 that this problem is overcome by using optical interconnection.

In this thesis, we study parallel architectures that use free space optics as a means of interprocessor communications. Replacing electrical interconnects with optical beams has a significant impact on the performance of VLSI architectures [12, 26]. This fact arises from the following two important properties of free space optics. First, free space optical beams can cross each other without any interference. Second, the connections need not be fixed and can be redirected [6].

## 1.2 Ideal Parallel Computational Model

From the point of view of the parallel algorithm people, the design issue of an algorithm depends more on the number of usable processors, the data accessing mode and the routing method in the system than on the circuit layout design. The computational model affects the algorithms design directly. The ideal model or the best algorithm design environment is that no matter what kind of computational architecture is used, the detail of the routing method does not reflect into the algorithm.

One of the widely used models of parallel computation is the Parallel Random Access Machine (PRAM). The basic assumption in this model is that in unit time each of  $N$  processors can simultaneously access any one memory location [66]. Unfortunately, it is unlikely that a PRAM model will ever faithfully represent any “real” parallel machine. A real parallel computer will most likely consist of a large number of simple processors, each connected to a small number of other processors. Each processor in this network has its own local memory, and processors communicate by sending messages over links to neighboring processors. To reconcile the convenience of a PRAM with the limitations of a real computer, it is simulated on a real network.

One of the first randomized simulations is given in [61], where it is shown that there exists an  $N$ -processor realistic computer that can simulate an idealistic  $N$  processor PRAM with only a factor of  $O(\log N)$  loss of run time efficiency. In [51], this was improved by obtaining similar bounds but requiring only bounded queue size. The deterministic simulation takes the longer time. The deterministic simulation of EREW (exclusive read exclusive write) or CRCW (concurrent read concurrent write) PRAM on an  $N$ -processor butterfly needs  $O(\log M \log N \log \log N)$  steps, where  $M$  is the number of memory cells and  $M > N^2$ . A similar performance can be obtained from an  $N$ -node hypercube [35].

Since the end of the 1980’s, there has been an emerging interest in the design of models of parallel computation which more closely simulate a realistic

machine. In [24], a more restricted PRAM model called Distributed Random Access Machine (DRAM) is introduced, which reflects an assumption of limited communication bandwidth in the underlying network. All memory in DRAM is local to the processors, and is accessed by routing messages through a communication network. A stronger model called local memory PRAM was introduced in [2]. Like the DRAM, the memory is distributed. However, there is no restriction on the underlying communication network and hence it is assumed to have a unit time delay. Such a communication medium is feasible with fixed connection architectures of unbounded degree, or those with reconfigurable optical interconnects.

### 1.3 Optical Interconnection Networks

A considerable amount of research has been done on exploring the features of optics. The theoretical and experimental results support a point that the optical interconnection could be an ideal substitution of the electronic interconnection in parallel computers or networks. The significance of this on computer science may be deeper than expected.

#### 1.3.1 Why Optics in VLSI

Several characteristics of present hardware techniques limit the density of electrical interconnects. One limitation is that the edge of the chip is reserved for I/O functions. Another is that electrical interconnects are confined to pseudo-planar structures (e.g., printed wiring boards, backplanes). The phenomenon of crosstalk is a fundamental limitation on spacing between individual interconnects. This density issue is aggravated with increases in speed in individual devices. As speeds increase, sensitivity to crosstalk through the electrical interconnect also increases and the required distance between devices decreases to ensure that the signal propagation time is less than the clock period. Obviously, as density increases, the spacing between lines

decreases and it is necessary to reduce the cross-sectional area of the conductors in order to place more interconnections in a given volume.

It has been demonstrated that optical techniques provide a much higher density for a given bandwidth than electronic techniques. Using free space or wavelength propagation, it is possible to take advantage of the high density bandwidth product available in the optical domain. In addition, using integration of opto-electronic devices, it is possible to communicate with the interior of the chip rather than confining I/O to the chip boundary.

The system implemented by electronic computers and optical interconnection among the computers is called opto-electronic system. In section 1.1, the VLSI model and computational time of the electronic computer was discussed. In a three dimensional electro-optical model called VLSIO [3], which is the generalization of the VLSI model into three dimensional opto-electronic systems, a similar lower bound can be expressed as  $VT^{3/2} = \Omega(I^{3/2})$  where  $V$  is the circuit volume. This overcomes the problem caused by the two-dimension electronic wire interconnection limitation. This advantage is not gained by the optical versions of existing electronic architectures implemented by replacing wires on a VLSI chip with optical waveguides [21, 27]. The reason is that by using waveguides, the communication bandwidth can be increased but the interconnection topology is still the same as before which is two-dimensional while the optical free space technique uses the third dimension to transfer data.

An even better lower bound for the optical free space interconnection system is given in [18]. It is based on the fact that the distance of the free space between two planes of such kind of system is a constant. In some systems, each plane contains processors and the processors on one plane can communicate with the processors on another plane [45] through free space. Another typical optical free space interconnection is that one plane contains processors and another plane consists of light deflection devices. The data is sent from the processor plane and is received by the

processors on the same plane. The lights go to the deflection layer, and are redirected back to the processor plane through free space. For an optimal lower bound of optical free space interconnection system, please refer to [18].

### 1.3.2 Previous Work

There are two approaches to define the optical interconnection medium: guided-wave interconnect and free space coupling. Although many efficient models using wavelength division multiplexing (WDM) have been developed in recent years [57, 13, 62], they are outside the focus of this thesis. The work based on using free space interconnection methods are examined more closely in this section.

Some qualities of the optical medium are its abilities to be directed for propagation in free space and to have two optical channels cross in space with out interaction. These properties allow optical interconnects to utilize all three dimensions of space. Such a capability will allow optical interconnection to improve upon many of the functions presently implemented on a limited scale with electronics, such as routing data between processing elements based on data dependent decisions, as well as multiplexing and demultiplexing information. Another widely used advantage of optical free space is that the node-to-node interconnection can be changed freely and quickly. In the other words, the interconnection topology of an optical free space system can be reconfigured during the computation even though the reconfiguring takes time.

One of the first attempts in using free space optics as a means of data-communications is [23]. In their hybrid GaAs/Si approach to data communication, a GaAs chip with optical sources was connected in a hybrid fashion (with conventional wire bond techniques) to a Si chip such that light was generated only along the edges of the Si chip. The sources were of the edge-emitting or surface emitting type. The optical signals were routed to the appropriate locations on the Si chip using

conventional and/or holographic optical elements. The Si chip contained detectors to receive the optical data streams generated by the sources. Since the detector-amplifier combinations were fabricated in Si, every computational component on the Si chip was capable of receiving data.

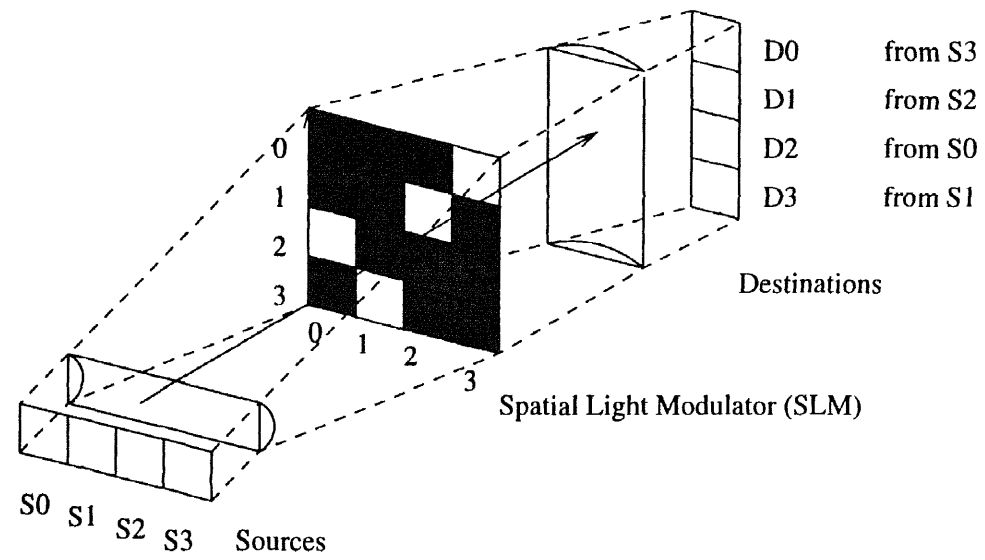
To explore this promising concept, it was extended to support efficient interconnection networks for massively parallel computing.

One idea is to make optical crossbar networks by using optical matrix- vector multiplication [22, 41, 47]. In the late 80's, Sawchuck et al. [55] described several possible bulk optical systems for implementing optical crossbar networks. Unlike electrical crossbar, these crossbars provided unit time interconnectivity and had a slow switching rate. A typical  $N$ -node optical crossbar system is shown in the figure 1.1 [53]. An  $N \times N$  switch array consists of  $N \times N$  optical switches, or some kind of optical devices. This matrix is called spatial light modulator (SLM) and modulates the cross-section of a light beam. The  $N \times N$  array corresponds to the permutation matrix that represents the interconnection among  $N$  processors. Once the switches are set, the messages can be transmitted at optical transmission rates, which can be several Gigahertz. But the switch array is not energy efficient and the switch setting is slow.

To avoid the above problems, the holographic technique is used. A hologram can be written dynamically and the recorded information can be retrieved later. So, it is used to remember the address pattern. The holograms can be dynamically recorded [68, 11], or can be fixed [15, 65, 40, 52] in the system. The time for recording a hologram is also long. Using fixed holograms to implement dynamic routing schemes is an important research topic on optical free space interconnection computers.

Many application of optical systems using the free space technique are emerging. A model for a data flow based processor is given in [8]. The concept is to set the interconnection dynamically among processors of the optoelectric network

Figure 1.1 Optical crossbar system





to match the dynamically abstracted data flow graph. The interconnection network is an optical implementation of a rearrangeably non-blocking Clos network, which is a three-stage permutation network. Several possible ways to realize the system are given. One way is to use the optical crossbar developed in [7]. Ferroelectric liquid crystal (FLC) devices are used, in which beam-blocking mask patterns are written similar to the way that an image is produced on a liquid crystal display screen. By the feature of the data flow system, the reconfiguration is needed. For a static data flow graph which does not need to change frequently, this approach is realistic.

A similar optical reconfigurable parallel architecture [25] supports advanced multiple functions. It provides a set of instructions to allow users to configure both the topology and the behavior of the architecture. The topology means the interconnection pattern (mesh, cube, etc.) and the behaviour means the style of the data processing (pipeline, dataflow, etc). This system adopts the SLM-based crossbar switch matrix which we introduced before. The SLM is situated between two banks of I/Os from the processors: outputs on one side, inputs on the other. The outputs are placed vertically and the inputs horizontally, thus each output illuminates a row of the SLM and each input reads from the column. The data transfer is through the optical free space. Since the switching time on the SLM frame in such a system dominates the system performance, the fast electronically controlled matrix-addressed ferroelectric liquid crystal (FLC) SLM is chosen.

The earlier work based on the similar idea is [62]. But the architecture is not an optical free space one; the model named Multiple Channel Architecture (MCA) can support large numbers of users with vastly different computing requirements through a fact that the large amount of independent, selectable channels (or virtual buses) can be provided by a single optical fiber. Depending on different computation requirement, different channels can be chosen by a computer to connect to some new neighbors, so that new interconnection topology can be built.

In recent years, there have been many experimental works done in optoelectronic computer systems supported by universities, governments and international organizations. These efforts are supported by theoretical research and they pace the commercialization of the opto-electric computers. One is ESPRIT II OLIVERS [49]. This is a three year collaborative project under the European Strategic Program for Research in Information Technology (ESPRIT). OLIVES means Optical Interconnections for VLSI and Electronic Systems. Four demonstrators of optical interconnections at the module, backplane, multichip module and chip levels were constructed. Various optical techniques were used in different demonstrators led by theoretical investigations. Among them, the optical free space technique is used to implement the mastercard for backplane interconnections. Its slab is provided with holograms which perform beam directing splitting and possibly focusing functions.

Another project of an optoelectronic 3-D system is [45]. The free space optically interconnected system with internal feedback loop based on the concept of [29] was presented. Two optoelectronic arrays (also know as smart-pixel arrays) each containing  $8 \times 8$  one-bit processors optically connected face-to-face through free space using a bidirectional holographic element. Optical sources are vertical-cavity surface-emitting lasers (VCSELs) which is an important research topic in the project. The goal of this experimental work is to find the parameters of the optical scheme such as the hologram and detector dimensions, their spacing and the Fourier objective diameter.

Free space optical interconnections are classified according to the degree of space variance [38], which determines the network's complexity and regularity. A totally *space-variant* network allows a completely arbitrary interconnection between components, whereas a totally *space-invariant* network has a definite, regular structure so that all the nodes in a system have the same connection patterns.

Most optical free space interconnection architectures mentioned above are space-variant. For the features of space-invariant system claimed in [39] and the interconnection simplicity, many space-invariant networks are researched [32, 39]. The typical method is to simulate an existing electric multistage network by assigning each node the regular connection to other nodes through optical components such as lenses, mirrors and holograms. Another optical space-invariant system is [37]. As in OLIVERS, it combines free space and waveguide technologies into one system even though each uses them in different system levels.

### 1.3.3 Overview of the Thesis

In this thesis, there are six chapters. In Chapter 2, we introduce an optical model of computations, OMC, and several implementations for the model. In Chapter 3, we present an electro-optical parallel architecture called the Optical Reconfigurable Mesh (ORM). It is an implementation of an existing optical model of computation (OMC) [15]. The ORM has two layers, the deflection layer and the processing layer. The processing layer is an  $N \times N$  reconfigurable mesh. The deflection layer is situated directly above the processing layer to provide unit-time free space optical interconnections for the processors. Three types of unit-time communication mechanisms supported by the architecture are introduced in the thesis. The first is for arbitrary planar connections among sets of locally connected processors using the reconfigurable mesh. The second is for arbitrary connections among  $N$  of the processors using the electrical buses on the processing layer and  $N^2$  fixed passive deflecting units on the deflection layer. The third is for arbitrary connections among any of the  $N^2$  processors using the  $N^2$  mechanically reconfigurable deflectors in the deflection layer. A set of basic data movement algorithms for those mechanisms are presented. To show the power of ORM, we use all three types of communication mechanisms in the first  $O(\log N)$  time algorithm for finding the convex hulls of all figures in an

$N \times N$  0/1 image presented in Chapter 4. The optical reconfiguration in the third type of communication mechanisms is significantly slower than the other two. It is desirable to avoid using this type of communication during the execution of the algorithms. Instead, we do the optical reconfiguration before the execution of each algorithm begins. Determining a right configuration that would be suitable for the entire configuration of a task execution is studied in Chapter 5. The conclusion and future works are discussed in Chapter 6. An appendix in this thesis includes some preliminary that used in the methodology studied in Chapter 5.

## CHAPTER 2

### AN OPTICAL MODEL OF COMPUTATION

In this chapter, the optical computational model OMC [15] is introduced and the implementations are reviewed.

#### 2.1 OMC

The OMC model (Optical Model of Computation) was first introduced by Eshaghian [15] in 1988. Its inherent EREW PRAM capability makes it very powerful to be used for various existing application parallel algorithms.

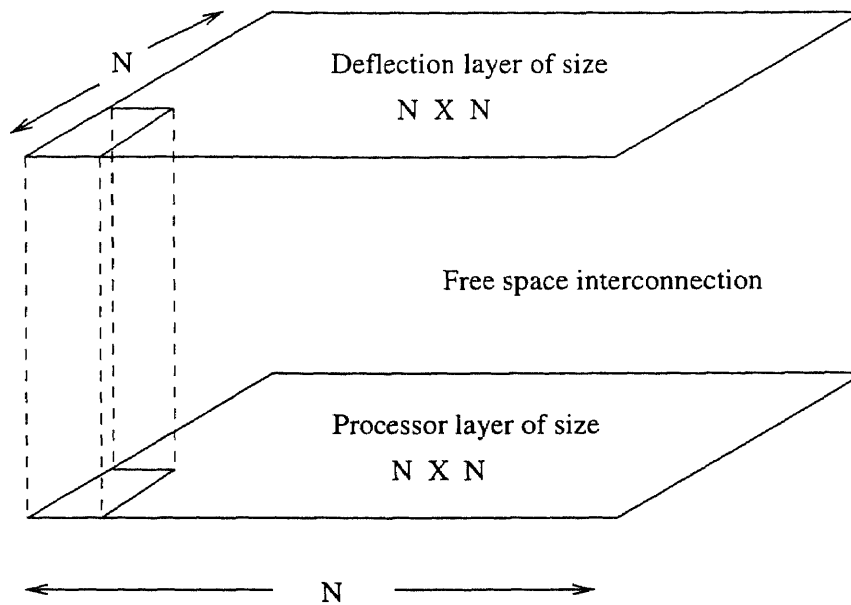
##### 2.1.1 Definition

The OMC model is shown in Figure 2.1. Formally, this model is defined as follows:

*An optical model of computation represents a network of  $N$  processors each associated with a memory module and a deflecting unit capable of establishing direct optical connection to another processor. The interprocessor communication is performed satisfying the following rules similar to [2]:*

- 1. At any time a processor can send at most one message. Its destination is another processor.*
- 2. The message will succeed in reaching the processor if it is the only message with that processor as its destination, within that step.*
- 3. All messages succeed or fail (and thus are discarded) in unit time.*

To insure that every processor knows when its message succeeds we assume that the OMC is run in two phases. In the first phase, read/write messages are sent and in the second, values are returned to successful readers and acknowledgements are



**Figure 2.1** The OMC model

returned to successful writers. It assumes that the operation mode is synchronous, and all processors are connected to a central control unit. The above definition is supplemented with the following set of assumptions for accurate analysis.

1. Processors are embedded in the Euclidean plane. This is referred to as the processing layer.
2. Each of the processing/memory elements occupies unit area.
3. Deflectors are embedded in the Euclidean plane. This is referred to as the deflecting layer.
4. Each deflecting unit occupies one unit area.
5. The deflecting layer is collinear to the processing layer.
6. I/O is performed at I/O pads. Each I/O pad occupies unit area.
7. The total volume is the sum of the volume occupied by the processing layer, the deflecting layer and the space for optical beams.
8. The intercommunication is done through free space optical beams.

9. Time is measured in terms of number of units of clock cycles.
10. An optical beam carries a constant amount of information in one unit of time, independent of the distance to be covered.
11. A deflector is capable of redirecting an incident beam in one unit of time.
12. A processor can perform a simple arithmetic/logic operation in one unit of time.
13. The time  $T$  for computation is the time between the arrival of the first input and the departure of the last output.

Compared with an electronic VLSI computation model, the following result can be stated:

**Proposition 1** *Any computation performed by a three dimensional VLSI organization having  $N$  processors with degree  $d$ , in time  $T$ , and volume  $V$  can be performed on OMC in volume  $v$ , and time  $t$ , where  $dT/N \leq t \leq T$ , and  $Nd \leq v$ .*

Its lower bound can be simply obtained by multiplying  $T$  by  $d/N$  which is the maximum speed up factor that can be obtained due to its unit time interconnection medium. The lower bound on  $v$  is obtained by the minimum area requirement for having  $d$  deflectors for each of the processing elements. In the next sections three different parallel architectures are presented as possible efficient upper bounds for  $v$ .

### 2.1.2 Implementations

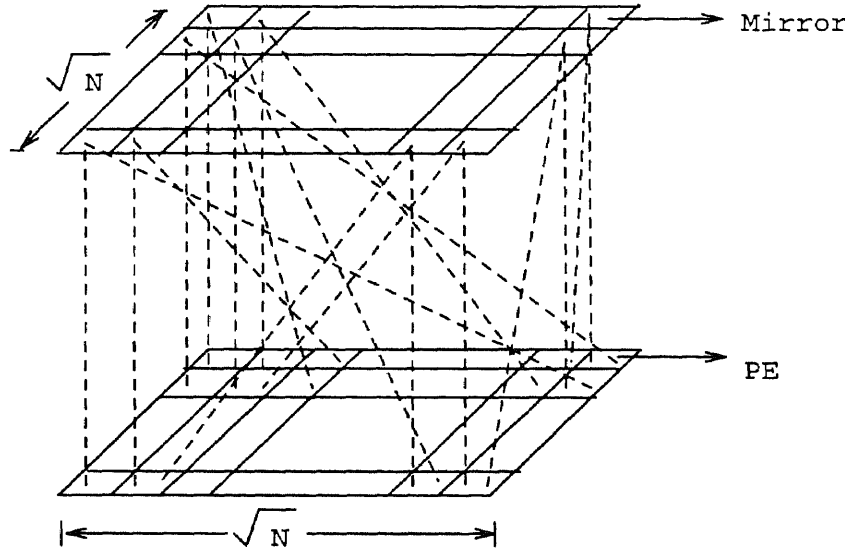
In this section, a class of optical interconnection networks as a realization of the OMC introduced in [15] are reviewed. Each of the designs uses a different optical device technology for redirection of the optical beams to establish a new topology at any clock cycle, and represents an upper bound on the volume requirement of OMC.

**2.1.2.1 Optical Mesh Using Mirrors:** In this design, there are  $N$  processors on the processing layer of area  $N$ . Similarly, the deflecting layer has area  $N$  and holds  $N$  mirrors, each with its own arithmetic unit.. These layers are aligned so that each of the mirrors is located directly above its associated processor (see Figure 2.2). Each processor has two lasers. One of these is directed up towards the arithmetic unit of the mirror and the other is directed towards the mirror's surface. A connection phase would consist of two cycles. In the first cycle, each processor sends the address of its desired destination processor to the arithmetic unit of its associated mirror using its dedicated laser. The arithmetic unit of the mirror computes a rotation degree such that both the origin and destination processors have equal angle with the line perpendicular to the surface of the mirror in the plane formed by the mirror, the source processor, and the destination processor. Once the angle is computed, the mirror is rotated to point to the desired destination. In the second cycle, a connection is established by the laser beam carrying the data from the source to the mirror and from the mirror being reflected towards the destination. Since the connection is done through a mechanical movement of the mirror, with the current technology this leads to an order of milli-second reconfiguration time. Therefore this architecture is suitable for applications where the interconnection topology does not have to be changed frequently. In [33], the design of various topologies has been studied to minimize the time complexity of several problems for a fixed period of computation.

The space requirement of this architecture is  $O(N)$  under the following assumption. Each mirror is attached to a simple electro-mechanical device which takes one unit of space and can rotate to any position in one unit of time. The assumptions are as valid as those in VLSI such that the constant propagation delay assumption regardless of the wire length. Other assumptions can also be made based on the following arguments. Many mirrors have a reconfiguration delay



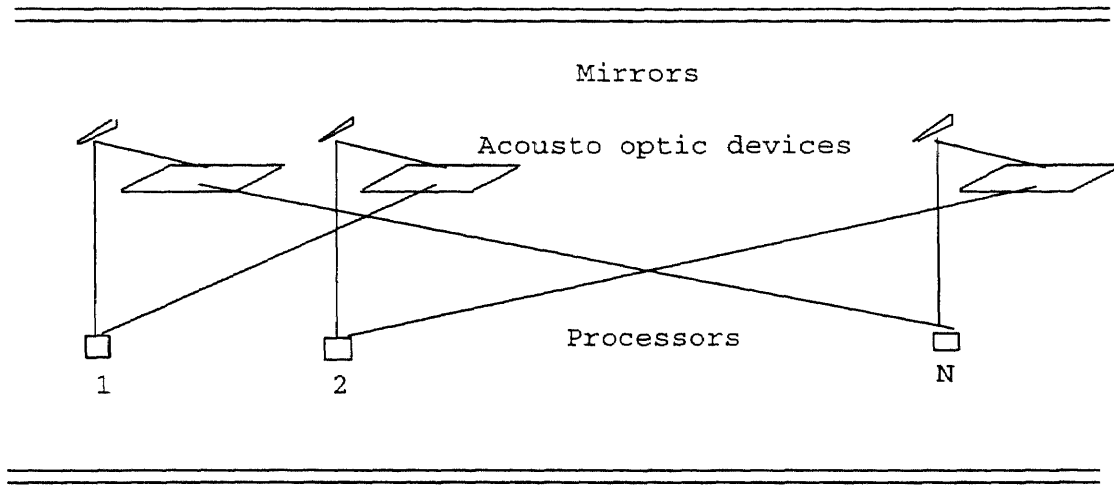
proportional to their rotation angle,  $O(N)$ . More complex mirrors on the other hand, can rotate faster for a larger angle (unit time rotation delay ) but their size can grow proportional to the number of angles they can realize (  $O(N)$  ).



**Figure 2.2** An optical mesh using mirrors

**2.1.2.2 Reconfiguration Using Acousto-Optic Devices:** In this organization,  $N$  processors are arranged to form a one-dimensional processing layer and the corresponding acousto-optic devices are similarly located on a one-dimensional deflecting layer (see Figure 2.3).

The size of each of the acousto-optic devices is proportional to the size of the processing array, leading to an  $O(N^2)$  area deflection layer. Similar to the design using the mirrors, every processor has two lasers, and each connection phase consists of two cycles. In the first cycle, each processor sends the address of its desired destination processor to the arithmetic unit of its associated acousto-optic unit using its dedicated laser beam. The acousto-optic cell's arithmetic unit

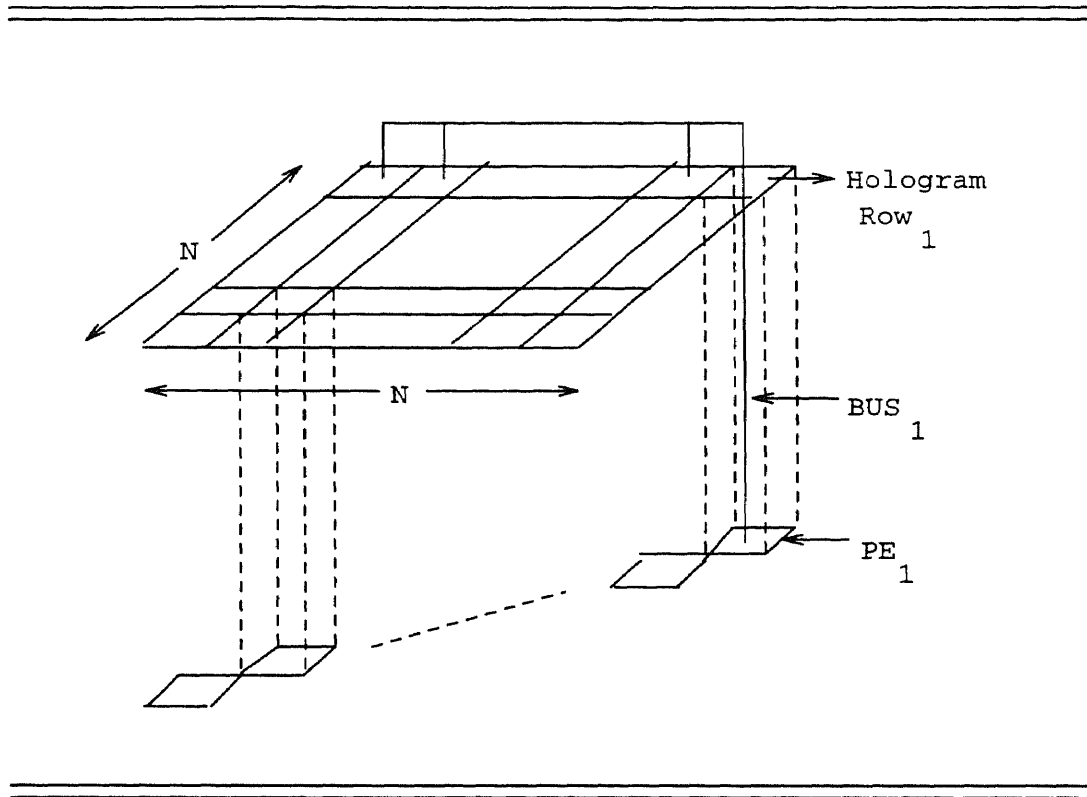


**Figure 2.3** Reconfiguration using acousto-optic devices

computes the frequency of the wave to be applied to the crystal for redirection of the incoming optical beam to the destination processor. The acousto-optic device then redirects the incident beam from the source to the destination processor. One of the advantages of this architecture over the previous design is its order of micro-seconds reconfiguration time, which is dominated by the speed of sound waves. The other advantage is its broadcasting capability, which is due to the possibility of generating multiple waves through a crystal at a given time.

**2.1.2.3 Electro-Optical Crossbar:** This design uses a hybrid reconfiguration technique for interconnecting processors. There are  $N$  processors, each located in a distinct row and column of the  $N \times N$  processing layer. For each processor, there is a hologram module having  $N$  units, such that the  $i^{th}$  unit has a grating plate with a frequency leading to a deflection angle corresponding to the processor located at the grid point  $(i, i)$ . In addition, each unit has a simple controller and a laser beam. To establish or reconfigure to a new connection pattern, each processor broadcasts the address of the desired destination processor to the controller of each of  $N$  units of its hologram module using an electrical bus (see Figure 2.4). The controller activates a laser (for conversion of the electrical input to optical signal), if its ID matches the

broadcast address of the destination processor. The connection is made when the laser beams are passed through the predefined gratings. Therefore, since the grating angles are predefined, the reconfiguration time of this design is bounded by the laser switching time which is in the order of nanoseconds using Gallium Arsenide (GaAs) technology [30].



**Figure 2.4** Reconfiguration using an electro-optical crossbar

This architecture is faster than the previous designs and, further, it compares well with the clock cycle of current supercomputers. One of the advantages of this simple design is its implementability in VLSI, using GaAs technology. Due to the above advantages, it gives the flexible usage of the  $N^2 - N$  vacant areas on the processing layer, the extension on this architecture leads to a new implementation of OMC.

## 2.2 Applications

In this section, using OMC, we present several parallel algorithms for fine grain image computation. We categorize the results in the following order. We present a set of processor efficient optimal  $O(\log N)$  algorithms and a set of constant time algorithms for finding geometric properties of digitized images. Finally, we focus on special purpose designs tailored to meet both the computation and communication needs of problems such as those involving irregular sparse matrices.

### 2.2.1 Optimal Geometric Algorithms

In this section, we present  $O(\log N)$  algorithms for problems such as labeling figures and finding the nearest neighbor figure to each figure in an  $N \times N$  image. The input to our algorithms is a *digitized* picture with  $PE(i, j)$  storing the pixel  $(i, j)$ ,  $0 \leq i, j, \leq N - 1$  in the plane, where the black pixels are 1-valued, and white pixels are 0-valued.

Connectivity among pixels can be defined in terms of their adjacency. Two black pixels  $(i_1, j_1)$  and  $(i_2, j_2)$  are *8-neighbors* if  $\max\{|i_1 - i_2|, |j_1 - j_2|\} \leq 1$ , and *4-neighbors* if  $|i_1 - i_2| + |j_1 - j_2| \leq 1$ . Two black pixels  $(i_1, j_1)$  and  $(i_k, j_k)$  are said to be connected by a *8-path*(*4-path*) if there exists a sequence of black pixels  $(i_p, j_p)$ ,  $2 \leq p \leq k$ , such that each pair of pixels  $(i_{p-1}, j_{p-1})$  and  $(i_p, j_p)$  are 8-neighbors(4-neighbors). A maximal connected region of black pixels is called a *connected component*.

**2.2.1.1 Labeling Digitized Images:** An early step in image processing is identifying figures in the image. Figures correspond to connected 1's in the image. An  $N \times N$  digitized picture may contain more than one connected region of black pixels. The problem is to identify which figure (label) each "1" belongs to.

**Lemma 1** *Given an  $N \times N$  0/1 image, all figures can be labeled in  $O(\log N)$  time using an  $(N \times N)$ -optical mesh.*

**Proof:** The basic idea of the algorithm is to identify the outer and inner boundaries of each figure and then to uniquely label all of the connected figures surrounded by each of these boundaries [1]. To assure circular boundaries, the input image is magnified by a factor of two along each dimension. Each pixel then locally determines whether or not it is a boundary pixel by checking if at least one of its four adjacent pixels along the  $x$  and  $y$  axis, hold a “0”. The pixels along each boundary are linked to form a circular list.

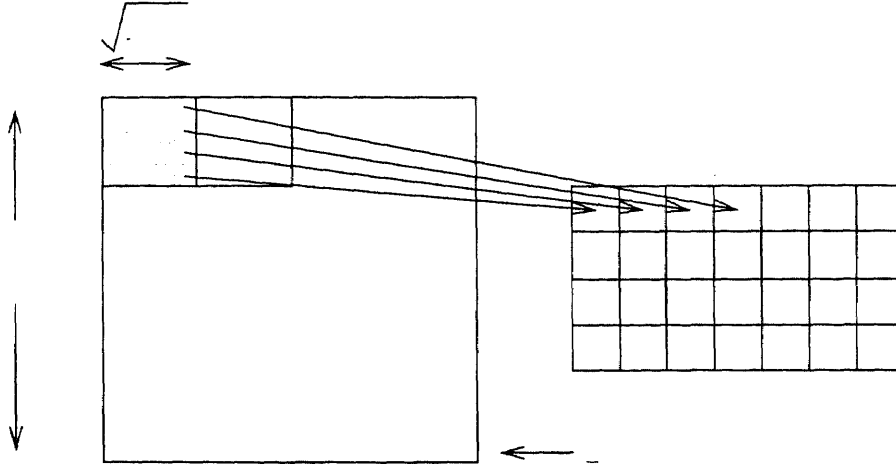
From now on, only the boundary PEs take part in the computation to identify the least numbered PE in their list. Each PE during iteration  $i + 1$ , sets its pointer to the pointer of the PE it was pointing to at the end of iteration  $i$ . This technique is called pointer jumping and is commonly used in parallel algorithms now. Since this has the effect of doubling the distance “jumped” during each iteration, in  $O(\log N)$  time all the PEs in each list know the least numbered PE in their list. The final step is the propagation of the unique IDs of each of the outer boundaries to its inner region. Broadcasting of IDs is done in parallel along each row of the image. It is easy to see that since the figures do not cross there is always a unique ID broadcasted to each of the inner PEs.  $\square$

In the following, we use fewer processors to lead to the optimal solution.

**Theorem 1** *Given a  $N \times N$  0/1 image, all figures can be labeled in  $O(\log N)$  time using an  $(N/\log^{1/2} N \times N/\log^{1/2} N)$ -optical mesh.*

**Proof:** In the first step, we assign a  $\log^{1/2} N \times \log^{1/2} N$  block of image to each processor, and we sequentially label the figures within these regions. This is accomplished using a serial graph traversal technique.

In the second step, these blocks are merged together until the block size becomes  $\log N \times \log N$  (see Figure 2.5). During each iteration, four blocks of size



**Figure 2.5** Merging of blocks and processor assignment

$k \times k$  are merged to obtain a block of size  $2k \times 2k$ . This is performed by assigning the available PEs to block boundary pixels, and then applying the algorithm of lemma 1 to merge each pair of blocks. Since there are not enough processors available to hold all the block boundary points, they are processed by groups of  $\log N$  at a time. Hence, the total time to simulate each of  $\log \log N$  iterations is  $O(\log^{1/2} N)$ . This leads to a total of  $O(\log^{1/2} N \log \log N)$  time complexity for the second reduction step. Using Lemma 1, the remaining pixels are labeled.

### 2.2.2 Distance Problems

Another interesting problem is to identify and to compute the distance to the nearest figure to each figure in a digitized image. In the following, we use the  $l_1$  metric. However, it can be modified to operate for any  $l_k$  metric.

**Theorem 2** *Given an  $N \times N$  0/1 image, the nearest figure to all figures can be computed in  $O(\log N)$  time using an  $(N/\log^{1/2} N \times N/\log^{1/2} N)$ -optical mesh.*

**Proof:** This algorithm consists of two steps. In the first step, each black pixel finds its nearest neighboring black pixel which belongs to a different figure. In the second step, each figure finds its nearest neighboring figure by finding the minimum among all those values obtained for the PEs at the boundaries. This can be done in  $O(\log N)$  time, using the techniques of [50], once the input has been reduced to match the number of processors. Reduction is possible by assigning  $O(\log^{1/2} N \times \log^{1/2} N)$  pixels to a single processor.

□

### 2.2.3 Constant Time Geometric Algorithms

One of the most attractive properties of optics is superposition [28]. This property suggests that the resultant disturbance at any point in a medium is the algebraic sum of the separate constituent waves. Hence, it enables many optical signals to pass through the same point in space at the same time without causing mutual interference or crosstalk. Using this property, in [31] it is shown how a single memory element can be read by many processors at the same time. In this section, we employ this characteristic to allow concurrent writes if all the requesting processors want to write a “1”. This leads to constant running time of the following geometric algorithms, under the assumption that broadcasting can be done in unit time:

**Lemma 2** *Given an  $(N^{1/2} \times N^{1/2})$  image, using an  $(N \times N)$  optical mesh, in  $O(1)$  time,*

1. *For a single figure, its convex hull and a smallest enclosing box can be determined.*
2. *For each figure, the nearest neighboring figure can be identified.*

#### 2.2.4 A Special Purpose Design for Parallel Implementation of Iterative Methods for Higher Level Vision Processing

Solutions to many problems in image understanding can be posed in terms of iterative improvement to an initial configuration. For example, discrete relaxation-based approaches to scene labeling can be viewed as an iterative improvement process. In such problems, the underlying graph is usually sparse. But this sparsity is not regular. Efficient parallel implementations of such relaxation methods are possible with OMC.

Any iterative matrix structure can be realized by OMC using devices such as holograms (or those described in Chapter 1). Although the reconfiguration time for the holograms can be in the order of seconds, it only has to be set once during the processing phase. The structure of the coefficient matrix is used to define the holographic connections. This interconnection pattern remains the same throughout the computation. An optimal  $O(\log m)$  time can be achieved by this design where  $m$  is the number of nonzero elements in the matrix. This method is attractive when many computations are to be performed in which the structure of the coefficient matrix is fixed, such as iterative methods.

#### 2.2.5 Conclusion

The Optical Model of Computation, OMC, was introduced in this chapter. The optical parallel architecture ORM presented in the next chapter is based on this model.



## CHAPTER 3

### OPTICAL RECONFIGURABLE MESH

In this chapter, we will present a new electro-optical parallel architecture which is an implementation of the OMC model. This architecture is called the Optical Reconfigurable Mesh (ORM). The ORM has two layers, the deflection layer and the processing layer. The processing layer is an  $N \times N$  reconfigurable mesh. The deflection layer situated directly above the processing layer, provides unit-time free space optical interconnections for the processors. The three types of unit-time communication mechanisms supported by the architecture are introduced. A set of basic data movement algorithms for those mechanisms is discussed.

#### 3.1 Introduction

The ORM has two layers, the deflection layer and the processing layer. Each processor has a corresponding deflection unit situated above it. A reconfigurable mesh is used to build the processing layer so each processor can communicate with other processors either through an optical interconnection or electrical buses. Reconfigurable bus systems have been studied extensively in the past few years. Many different models of reconfigurable systems have been designed since the end of the 1980's [43, 4, 48, 58, 60, 36, 63]. Typically, a reconfigurable-bus architecture consists of a multi-dimensional array of processing elements (PEs). Those PEs are connected to buses through a fixed number of I/O ports. Each PE can locally control the I/O port connection to the bus in each machine communication cycle. The bus reconfiguration can then be made by a different connection style. A reconfigurable mesh is a two dimensional reconfigurable bus system. The most general and powerful reconfigurable mesh model is PARBS [63].

Since we want to concentrate on the optical interconnection in the architecture, we will not describe the reconfigurable mesh and its applications in detail. What

we discuss here are its major advantages and disadvantages, how it is served in our optical computation model and how the combination of the traditional reconfigurable mesh and optical interconnections makes an even more powerful architecture. The reconfigurable mesh has a simple and uniform VLSI layout. For those applications in which most computations need to be done through communication with local neighbors by each processor (directly or indirectly connected through the bus segments), it is a very powerful computation architecture. However, for global communications, for example, when large amounts of data need to be transferred among different electrical bus segments, the reconfigurable mesh becomes very inefficient. On the other hand, a free space optical architecture is very good for the global data communications due to the free connections among processors. If fixed deflecting units (e.g. fixed holograms) are used in a free space interconnection, we usually need  $N \times N$  deflecting units to implement the random permutation for only  $O(N)$  processors. If reconfigurable deflecting units are used, we may have  $N$  deflecting units for  $N$  processors. However, this use of the VLSI area is paid off by the time used for reconfiguration of optical deflecting units. Since the computation pattern could be changed very often in an application algorithm, this can be very time consuming for a pure optical reconfigurable interconnection system. Here, "pure" means all interconnections among processors are optical.

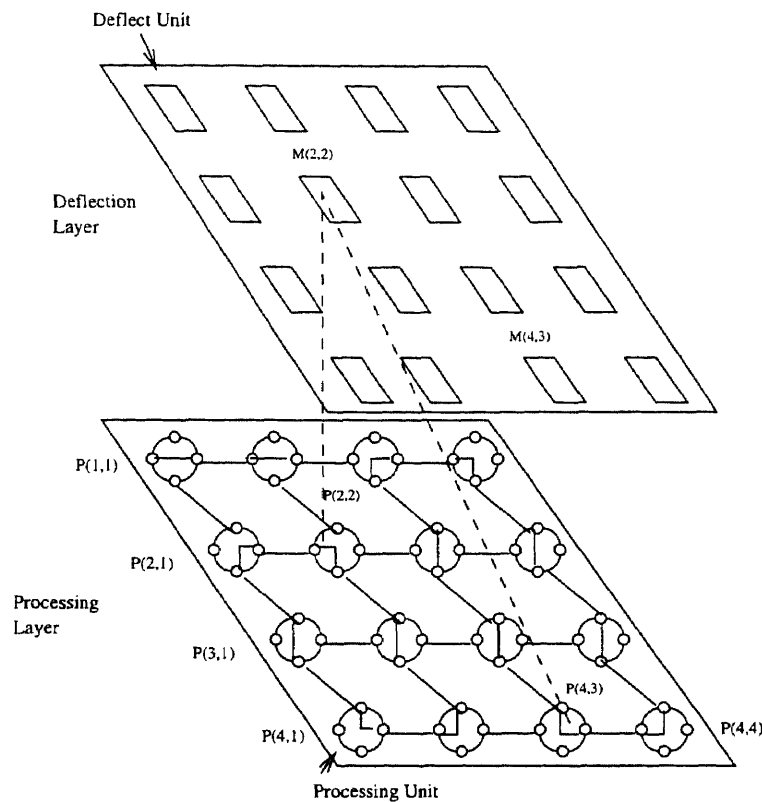
ORM provides three types of communication mechanisms. The first is for arbitrary planar connections among sets of locally connected processors using the reconfigurable mesh. The second is for arbitrary connections among  $N$  of the processors using the electrical buses on the processing layer and  $N^2$  fixed passive deflecting units on the deflection layer. The third is for arbitrary connections among any of the  $N^2$  processors using the  $N^2$  mechanically reconfigurable deflectors in the deflection layer. The reconfiguration time of the first two communication mechanisms is on the order of nanoseconds, while for the third it is on the order

of milliseconds. One method to avoid the third type of reconfiguration during the execution is to set it up to an efficient topology before the execution starts. A good topology would be one which matches the data flow requirement of the problem being solved. Butrym, Craft, Guise, Murdocca and Sauer [8] and Guan, Barros [25] studied this problem recently.

The rest of the chapter is organized as follows. In Section 3.2, we introduce the ORM architecture in detail. In Section 3.3, the read/write and data movement operations on three communication mechanisms of ORM are described. The conclusion is given in Section 3.4.

### 3.2 The ORM Architecture

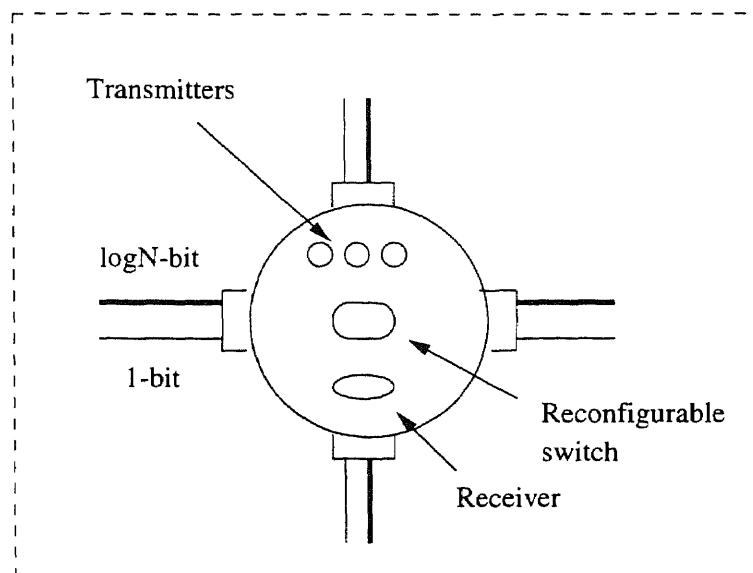
A  $4 \times 4$  optical reconfigurable mesh (ORM) is shown in Figure 3.1. There are two layers in ORM: the deflection layer and the processing layer. The deflection layer consists of  $N^2$  deflecting units and the processing layer has  $N^2$  processing units. The processors on the processing layer are interconnected as a reconfigurable mesh and can also intercommunicate optically using the deflection layer. The reconfigurable mesh model used here is similar to PARBS [63]. The reconfigurable mesh of size  $N^2$  consists of an  $N \times N$  array of processors connected to a grid-shaped reconfigurable broadcast bus, where each processor has a locally controllable bus switch. The switches allow the broadcast bus to be divided into subbuses, providing smaller reconfigurable meshes or reconfigurable bus segments. Figure 3.2 shows the detailed structure of a processing unit in the processing layer and Figure 3.3 shows the detailed structure of a deflecting unit in the deflection layer. In the following we describe each of those components.



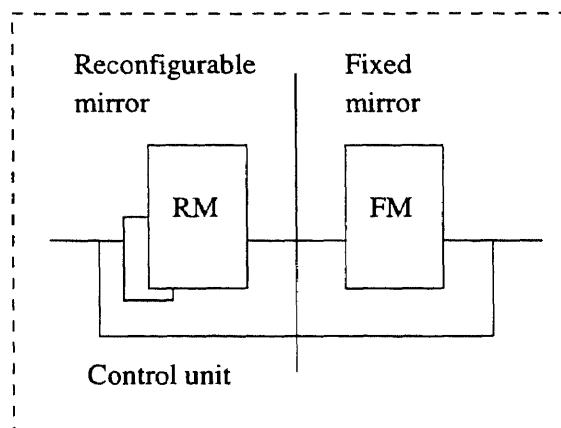
**Figure 3.1** The ORM architecture

### 3.2.1 The Processing Unit

There are  $N \times N$  processing units on the processing layer. There are three optical transmitters and one receiver residing in each processing unit. One of the transmitters,  $TR(1)$ , is directed towards the control unit of the deflection unit. The second one,  $TR(2)$ , is directed towards the reconfigurable mirror (RM) of the deflection unit and the third one,  $TR(3)$ , is directed towards the fixed mirror (FM) of the deflection unit. Each processing unit has a constant number of  $\log N$  bit memory cells and simple computation capabilities. It is connected to other processing units in the mesh by the electrical reconfigurable buses. Each processing unit controls the internal reconfigurable switches and is responsible for sending and receiving data to and from other processing units. We index the processing unit in the  $i$ th row and the  $j$ th column of the mesh on the processing layer as  $P(i, j)$  in which,  $1 \leq i, j \leq N$ .



**Figure 3.2** The detailed structure of a processing element



**Figure 3.3** The detailed structure of a deflector

### 3.2.2 The Deflection Unit

The deflecting layer contains  $N \times N$  deflecting units. Each deflecting unit consists of two mirrors and an arithmetic control unit. One of the mirrors, called FM (fixed mirror), is fixed which transfers data from the processor under it to a fixed address whenever it is used. Another mirror, called RM (reconfigurable mirror), is reconfigurable. The control unit receives an address from the processor under it, translates the address and controls the direction of the RM. Since the angle of the FM is fixed, the processor can send data directly from one of dedicated transmitters to the desti-

nation without any need to go through the control unit. We define each deflecting unit located directly above  $P(i, j)$  (a mirror and the related control unit) as  $M(i, j)$ .

### 3.3 Data Movement in ORM

The data can be routed in three different styles in this architecture. In the first method, routing is done only through electric buses. This is called *electrical routing*. The second one is called *optical routing* which uses free space optics. The third type uses electrical and optical free space connections to allow a complete connections among  $N$  processors, and is called *electro-optical routing*. Each of the movements is described below.

#### 3.3.1 Electrical Routing

The electric routing in ORM is similar to that of the PARBS system [63]. The electric routing in ORM is any routing from one node to another or a broadcast which uses electric buses in the reconfigurable mesh only. An example for reconfiguration is shown in Figure 3.1. This type of communication is suitable for providing arbitrary connections in the processing layer. For example, see the processing layer of Figure 3.1. There are three groups of processors. Each group has one common bus it can use for intercommunications.

- $P(1, 1)$  and  $P(1, 2)$  are connected to each other.
- $P(2, 1), P(3, 1), P(4, 1), P(4, 2), P(3, 2)$  and  $P(2, 2)$  are connected as a circle.
- $P(1, 3), P(2, 3), P(3, 3), P(4, 3), P(4, 4), P(3, 4), P(2, 4)$  and  $P(1, 4)$  are connected as a circle.

#### 3.3.2 Optical Routing

The optical routing in ORM is the routing through optical free space interconnections only. The data transfer would not use any electric bus in the system. All  $N^2$

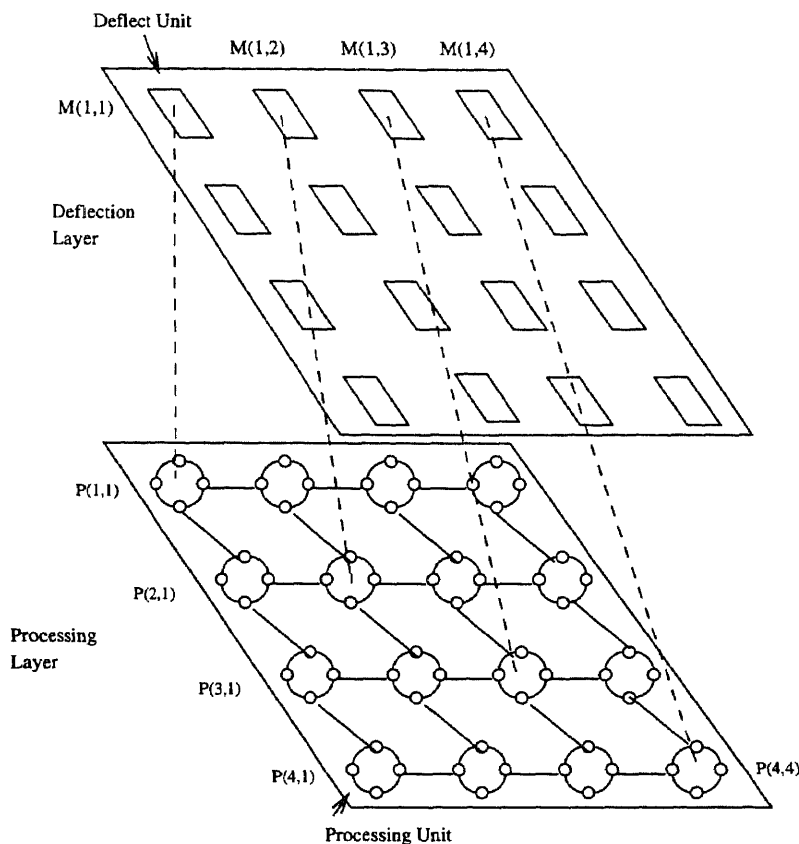
processors can intercommunicate in unit time delay, as long as there is only one read or write from or to each location. In the following, we describe how such an optical connections is established between two processors through the RM (Reconfigurable Mirror).

A connection phase consists of two cycles. In the first cycle, each processor sends the address of its desired destination processor to the arithmetic control unit of its associated mirror using its dedicated laser TR(1). The arithmetic control unit of the mirror computes a rotation degree such that both the origin and destination processors have equal angle with the line perpendicular to the surface of the mirror in the plane formed by the mirror, the source processor and the destination processor. Once the angle is computed, the mirror is rotated to point to the desired destination. In the second cycle, the connection is established by the laser beam TR(2) carrying the data from the source to the mirror and from the mirror being reflected towards the destination. An example of an optical routing from processor  $P(2,2)$  to processor  $P(4,3)$  is shown in figure 3.1.

The read operation has two phases. In the first phase, the read requirement and the reader's address are sent to the processor which stores the desired data. In the second phase, the data is sent back to the reader depending on the reader's address. Both phases use the two-cycle write routing method.

### 3.3.3 Electro-Optical Routing

This communication mechanic establishes an efficient full connectivity among only  $N$  processors of  $N^2$  processors of ORM situated diagonally in the processing layer as show in Figure 3.4. (i.e. for processors  $P(j,j)$  where  $1 \leq j \leq N$ ). The routing technique uses electric buses on the processing layer and the fixed mirrors on the deflection layer. This type of connection is implemented in the following way as shown in Figure 3.4. Each processor  $P(j,j)$  is associated with the  $j$ th row of the



**Figure 3.4** The optical interconnections for electro-optical routing of ORM

deflection unit, where the row contains  $N$  fixed mirrors. The  $i$ th fixed mirror in that row for  $1 \leq i \leq N$  is directed to the processing unit  $P(i, i)$ . The Figure 3.4 shows this communication mechanism. The  $i$ th fixed mirror in the first row which is directed to the processing unit  $P(i, i)$ ,  $1 \leq i \leq 4$ , is displayed in the figure. The other rows have the same type of optical interconnections which are not shown in the figure for the sake of clarity. There are two types of routing possible; Exclusive Read Exclusive Write (EREW) and Concurrent Read Concurrent Write (CRCW). We explain both methodologies below (The other two techniques described earlier, electrical and optical routings, are EREW).

**3.3.3.1 EREW:** Any processor  $P(i, i)$  sends data to  $P(k, k)$  in the following way.

1.  $P(i, i)$  sends the data to  $P(i, k)$  through the electrical row bus;



2.  $P(i, k)$  sends data to  $P(k, k)$  through transmitter TR(3) and its deflector  $M(i, k)$ .

### 3.3.3.2 CRCW

**Definition 1** *The CRCW access model for  $N$  diagonal processors on ORM is defined as follows:*

- *In one write step, each  $P(i, i)$  can send one write request to another PE in the diagonal. If there is more than one write request to  $P(i, i)$ ,  $P(i, i)$  will receive only one of them.*
- *In one read step, each  $P(i, i)$  can send one read request to  $P(k, k)$ ,  $k \neq i$ . The reader (multiple readers are allowed) can get the requested data back in the same step.*

Now, we prove the following theorem.

**Theorem 3** *The concurrent write and the concurrent read of  $N$  PEs can be done on ORM in  $O(1)$  time.*

**Proof:** The proof is done by giving the following constant time algorithm. We assume that the read or write operation signal (operation command) is known by all PEs. The following steps are executed in constant time.

#### Write Operation

There are three steps in this operation. In step 1, the destination address for a write request is broadcast to the row  $i$  by each  $P(i, i)$ . The processor (in row  $i$ , for each  $i$ ) with a  $j$  index matching the destination address is an active processor in the step. This processor will be responsible to send data to the destination and its optical light beam is activated. In step 2, a unique data is chosen among multiple write requests to write to a processor. In this step, the losers will become inactive. In step 3, the

unique writing data is sent to each destination. The implementation details of each step are as follows.

1. Initially, the ORM performs the row bus connection. Each  $P(i, i)$  sends a write request destination address  $j$  to the row  $i$ . The address can be received by all PEs through the row buses of the mesh. Each PE compares the address with its own column index. The  $P(i, j)$  will mark itself as an active PE if the address  $j$  is matched to its column index. The others in row  $i$  do nothing.
2. All PEs of ORM performs the column bus connection except that each active PE disconnects its north port from the south port. Each active PE sends a signal to the south and check the north port. If an active PE does not receive any signal from its north, means that it is a northmost active PE in the column, it activates the light beam. All the other active PEs will not be active any more. The data in  $P(i, i)$  for which the  $P(i, j)$  is active will be the chosen one writing data to  $P(j, j)$ .
3. Each  $P(i, i)$  sends the writing data to the row again. The data will be received by active  $P(i, j)$  and sent to  $P(j, j)$  through the activated laser beam. Since there is only one sender left in each column after step 2, each  $P(j, j)$  will receive at most one data from the free space in the step.

### Read Operation

The concurrent read operation contains two phases. In the first phase, the readers send read requests to the destination  $P(j, j)$ . During this step, the electrical and optical route for a  $P(j, j)$  to send the data back to multiple readers has been established. In the second phase, the data is sent to the readers by  $P(j, j)$ . Two variables  $R$  and  $C$  are used in each PE to implement the operation.

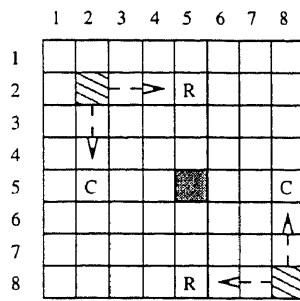
The implementation is as follows:

1. The ORM does the row buses connection. Each  $P(i, i)$  sends the destination processor address  $j$  and the read request (requested memory cell address) to the row  $i$ . When  $j$  is matched to the column index of a PE,  $P(i, j)$  saves the read request and sets the variable  $R = 1$ .
2. The ORM does the column buses connection. Each  $P(i, i)$  sends the address of the destination processor  $j$ , to the column  $i$ . Each PE compares  $j$  with its row index. If they match, the PE sets the variable  $C = 1$ .
3. The ORM keeps the column buses connection. The PE whose  $R = 1$  is an active PE in this step. Find the northmost active PE in each column. This PE activates the light beam and sends the read request to  $P(j, j)$  using transmitter TR(3). This can be done because the read request has been saved in this PE in step 1 and the active  $P(i, j)$  has its  $M(i, j)$  connect the optical path to  $P(j, j)$ .
4. The ORM does the row buses connection. Each PE with  $C = 1$  is an active PE now. It activates the light beam using transmitter TR(3). The requested data is retrieved by each  $P(j, j)$  and broadcast to row  $j$ . Then the data is sent to the requester  $P(i, i)$  through the bus and the light beam of  $P(j, i)$  using transmitter TR(3) as shown in the subfigure 3 in Figure 3.5.

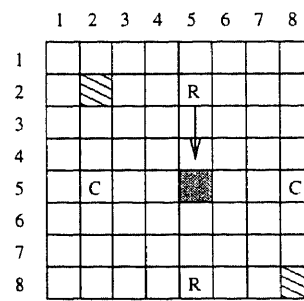
□

### Examples

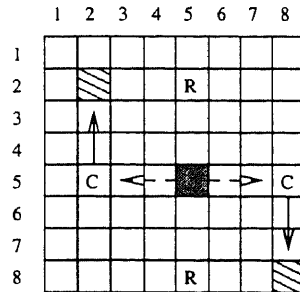
In this subsection, we give examples of concurrent write and concurrent read operations in electro-optical routing communication mechanism. The following is an example of a concurrent writing on an  $8 \times 8$  ORM. Assume that the requests for writing values to a variable stored in  $P(2,2)$  are made by  $P(1,1)$  and  $P(6,6)$  in the same step. The value received by  $P(2,2)$  is the value sent by  $P(1,1)$  since  $P(1,2)$  is the northmost active PE in column 2. In Figure 3.5, we use an  $N=8$  ORM system to explain the read operation. Assume that we view the architecture from the top.



1. Transparent top view of processing layer and deflection layer in step 2 and 3.





2. Step 4: the representative sends the read request to MPE(5) which stored the requested data.





3. Step 5: the receiver sends the data back to the requesters.

 Read requester

 Receiver

 R=1 in SPE

 C=1 in SPE

 The data transfer through the electric bus

 The data transfer through optical free space

**Figure 3.5** Concurrent read on ORM

We can see PEs in the bottom layer transparently through the deflection layer. Except for senders and receivers, other PEs are not indicated. In this example, we assume that  $P(2,2)$  and  $P(8,8)$  need to read data from  $P(5,5)$ .  $P(2,5)$  is chosen as the representative to send the required data address to  $P(5,5)$  in step 3. In step 4,  $P(5,5)$  sends data back. Since the light beams in  $P(5,2)$  and  $P(5,8)$  are activated, when the data is sent to the bus of row 5, it is redirected by  $M(5,2)$  and  $M(5,8)$  to  $M(2,2)$  and  $P(8,8)$ .

### 3.4 Conclusion

We introduced a powerful parallel architecture ORM, a network of  $N \times N$  processors on a VLSI chip interconnected through free space optical beams as well as through

electrical reconfigurable buses. The architecture supports three types of communication mechanisms. The basic data movements of those three communication types on ORM were discussed in the chapter. ORM combines the advantages of electrical reconfigurable bus interconnections and optical free space interconnections. The reconfigurable mesh has a simple and uniform VLSI layout and is very efficient for applications with regular data movement style. On the other hand, the optical mediums can be directed for propagation in free space and have two optical channels cross in space without interaction. So, data can be sent without any routing delay through optical free space interconnections when the destinations are known. It is efficient for applications with irregular data movement style. However, the third type of communication mechanism is significantly slower than the other two. In Chapter 5, we will present a methodology CONST. This methodology sets the optical reconfigurations before the execution of each algorithm begins to avoid using the third type of communication during the execution of the algorithm.

## CHAPTER 4

### A FAST PARALLEL IMAGE CONVEXITY ALGORITHM

In this chapter, we propose an application on ORM. It is the first  $O(\log N)$  time algorithm for finding the convex hulls of all figures in an  $N \times N$  0/1 image. This algorithm is faster than any existing one by a factor of  $O(\log N)$ .

#### 4.1 Introduction

We present an  $O(\log N)$  step algorithm to solve the challenging problem of finding the convex hull of multiple figures in an  $N \times N$  image in the chapter. The best known solution for this problem using any of the existing reconfigurable mesh models [42] and [16] is  $O(\log^2 N)$ . Furthermore, this problem has a time complexity of  $O(\log^2 N)$  using the optical mesh (an implementation of OMC [15] with mirrors). The convexity problem is an important computational task in image processing. The problem is to find the extreme points forming the convex hull for each figure on the image. The input for the problem is an  $N \times N$  digitized image distributed one pixel per processor on a reconfigurable mesh of size  $N \times N$  so that processor  $P_{i,j}$  stores pixel  $(i,j)$ . The pixels are either black or white where black represented by 1 is an image pixel and white represented by 0 is a background pixel. An assumption used in our algorithm is that any figure which has less than two rows or two columns of image pixels is not processed in this algorithm, since the extreme points of those figures can be easily recognized in constant time.

In this chapter, we propose the first  $O(\log N)$  time algorithm for finding the convex hulls of all figures in an  $N \times N$  0/1 image. This algorithm is faster than any existing ones by a factor  $O(\log N)$ . This illustrates that the proposed architecture is more powerful than any of the reconfigurable models or any of the earlier implementations of OMC on this application. The rest of the chapter is organized as

follows. In the next section, the convexity algorithm is described. The conclusion of the chapter is in Section 4.3.

## 4.2 $O(\log N)$ Convexity Algorithm

In this section, an efficient parallel algorithm for solving the convexity problem on ORM is presented.

An assumption is that any figure which has less than two rows or two columns of image pixels is not processed in this algorithm, since the extreme points of those figures can be easily recognized in constant time. All figures considered here have more than two rows and two columns.

The algorithm consists of two phases. In phase 1, the outer boundaries are found in  $O(\log N)$  time and then the boundary segment of each figure in  $G$  is cut into four sub boundary segments. Each subsegment represents a region of the figure. The concept of the region largely restricts the number of nodes which participate in the comparisons and makes the extreme point recognitions simple and easy. In phase 2, divide and conquer methodology is used to recognize the valleys in each of the regions. To obtain the extreme points for all four regions, it is repeated four times, once for each region. Each of the  $\log N$  iterations takes  $O(1)$  time. In each iteration, points in the valleys are dropped and the remaining ones are the input points for the next iteration. The set of points at the end of the last iteration are the extreme points. The method for finding the valleys includes the calculation of the angle each point makes with its neighbor and the highest reference point (rightmost, topmost, leftmost, or bottommost) in its region. In the following, the theorem and the proof for the algorithm is presented.

### 4.2.1 The Algorithm

**Theorem 4** *Given an  $N \times N$  image  $G$ , the convex hulls of all figures in  $G$  can be found in  $O(\log N)$  time using an  $N \times N$  ORM.*

**Proof:** We prove the theorem by giving the following  $O(\log N)$  algorithm. The algorithm consists of two phases. Phase 1 recognizes four regions in each figure. Phase 2 drops all nonextreme points in one of the four regions. Therefore, the phase 2 runs four times in the whole algorithm for dropping nonextreme points in all four regions of each figure. We define the indexing system for the image and the ORM. The first row on the top has  $i$  index  $i = 1$  and the first column on the left has  $j$  index  $j = 1$ . The node  $PE(i, j)$  means the node in the  $i$ th row and the  $j$ th column of the ORM. We assume that each node  $PE(i, j)$  represents a pixel  $p(i, j)$ . We also assume that each boundary node can recognize if it is an outer boundary node or an inter boundary node. The outer boundaries can be found and pointer directions can be set clockwise in  $O(\log N)$  time using the technique explained in [15]. It does not affect the performance of our algorithm. The rest of the algorithm continues with using outer boundary points only. For the convenience, we use “boundary node” instead of “outer boundary node” in the rest of paper. The pseudo-code of the algorithm is shown in Figure 4.1. The major function  $REGION_2(i)$  for dropping nonextreme points in region 2 in iteration  $i$  is shown in Figure 4.2.

#### PHASE 1:

First, we define that the topmost point of a figure as the one with the smallest  $i$  index on the figure’s boundary. If there is more than one boundary node with the same smallest  $i$  index, both the leftmost and the rightmost ones are called the topmost points of the figure. Similarly, we can define the downmost, the leftmost and the rightmost points.



### Convexity Algorithm

Assume an  $N \times N$  image on an  $N \times N$  optical reconfigurable mesh.

The algorithm is applied on outer boundary nodes only for each figure.

PHASE 1: Each figure does boundary nodes connection.

Find the upmost point in each figure boundary.

Find the downmost point in each figure boundary.

Find the leftmost point in each figure boundary.

Find the rightmost point in each figure boundary.

Each upmost, downmost, leftmost and rightmost point sends a signal which represents the direction ("U", "D", "L" or "R") to one region on the boundary in counter-clockwise direction.

PHASE 2 main()

for  $i = 1$  to  $\log N$

For  $j = 1$  to 4

case  $j$

1: call  $REGION_1(i)$

2: call  $REGION_2(i)$

3: call  $REGION_3(i)$

4: call  $REGION_4(i)$

end case

end for

end for

end main

(The function  $REGION_j(i)$ , where  $j=1,2,3,4$ , recognizes and drops the nonextreme points on the boundary region  $j$  for each figure in each submesh of size  $2^i \times 2^i$ .)

end {algorithm}

**Figure 4.1** Convexity Algorithm

Second, we define four regions of a figure. For convenience, we assume that there is only one topmost point on each figure's boundary. A similar assumptions can be made for the downmost, the leftmost and the rightmost points. We will discuss how to treat the case with more than one topmost /downmost/leftmost/rightmost point in one side after we define each of the regions as follows.

- *Region 1* is the boundary segment between the rightmost point and the topmost point.

```

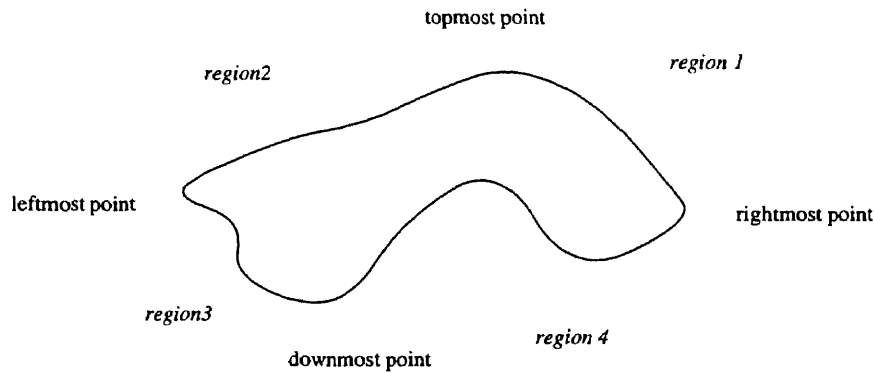
function REGION2(i)
  Super step 1:
  Find the extreme points in the upper half and lower half of the submesh.
  Each figure:
    call CONNECT() returning iceil, istore
    (iceil is the smallest i index of the figure in the half submesh;
    istore is an i index indicating a unique line for the figure)
    broadcasts iceil and istore to the boundary nodes
  Each C(k) node calculates:
    (k is the number for describing the relative position of a C
    node in the C node sequence along a boundary
    C(k + 1) is the right C node neighbor of C(k) in clockwise.)
    T(k) = tangent of C(k) and C(k + 1)
    P(k) = intersection of T(k) and y = iceil
    if P(k) is out side of the current submesh
      E = E - C(k + 1)
      (Drop C(k + 1) from E. It is a nonextreme point.)
    end if
  Each C(k):
    sends T(k) to the position P(istore, j) on line y1 = istore
    in which j is the j index of P(k).
    (P(istore, j) is called P node.)
  Each P(k) node:
    (we index the P nodes on the line y1=istore by the
    order of j index of P from left to right.
    The index of T and C are reordered following the order of P.
    sender(T(k)) is the C node which sends the tangent T to P(k))
    calculates the degree D(k) between T(k) and y = iceil.
    gets D(k + 1) and T(k + 1) from the right neighbor P(k + 1) on row
    istore.
    if D(k) ≤ D(k + 1) then
      E = E - sender(T(K + 1))
    else
      if (iofsender(T(K)) < iofsender(T(K + 1))) or
      (jofsender(T(K)) > jofsender(T(K + 1))) then
        E = E - sender(T(K + 1))
      end if
    end if
  Super step 2:
  (Merge the segments in the upper half and lower half of the submesh
  for each figure. It is similar to super step 1.)
end function

```

**Figure 4.2** Function *REGION*<sub>2</sub>(*i*)

- *Region 2* is the boundary segment between the topmost point and the leftmost point.
- *Region 3* is the boundary segment between the leftmost point and the downmost point.
- *Region 4* is the boundary segment between the downmost point and the rightmost point.

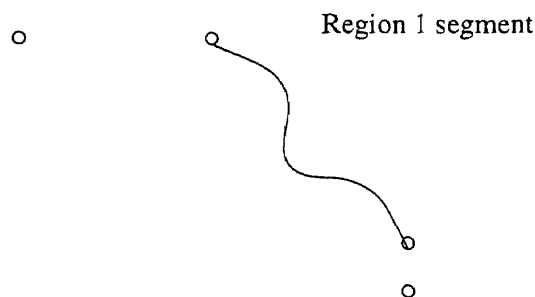
See Figure 4.3 for the region divisions.



**Figure 4.3** The region divisions

If there are two topmost points and two rightmost points in one figure, the region 1 would be the segment between the rightmost point which has the smaller  $i$  index and the topmost point which has the larger  $j$  index. Other regions are defined in a similar way. See Figure 4.4.

The algorithm in this phase is as follows. Each figure finds the topmost, downmost, leftmost and rightmost points on the figure's boundary. Finding the minimum and maximum of  $N$  elements connected with a bus can be done in  $O(\log N)$  time by checking  $\log N$  bits one at a time, similar to the bit polling technique explained in [17]. Now, each figure's boundary is cut into four boundary segments. Each node recognizes the region it belongs to and marks itself. This can be done in constant time by each leftmost, rightmost, upmost and downmost node disconnects



**Figure 4.4** The region 1 in the figure with multi-most points

itself from the neighbor in the clockwise direction along the figure boundary and broadcasts different signal “L”, “R”, “U” and “D” to the boundary segment along the counter-clockwise direction. This makes four boundary segments each of which is a region in a figure. Each boundary node has a variable REGION. Assume that the signal “L”, sent by leftmost point, is received by a boundary node, this node sets 3 in the variable REGION to indicate that it is a boundary node on region 3. If there are two topmost points in a figure, then all boundary nodes between these two nodes are eliminated. The boundary nodes between two leftmost, two rightmost and two downmost points are eliminated in a similar way.

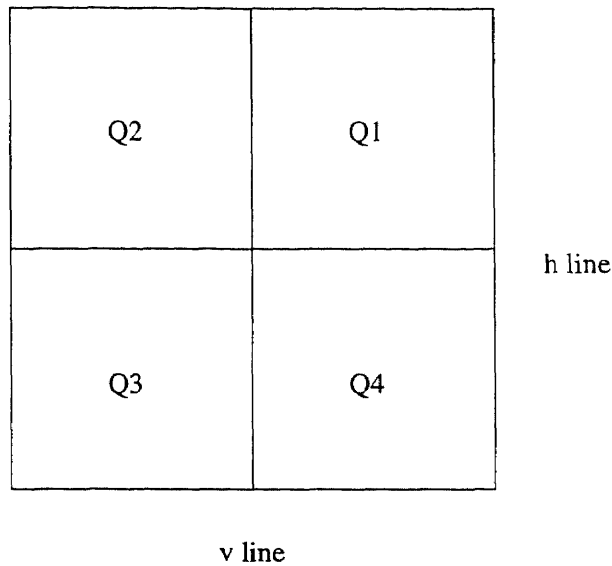
## PHASE 2:

A divide and conquer methodology is used to recognize the valleys in each boundary region. All those points in the valleys are dropped and the remaining points after phase 2 are the extreme points. The valleys are determined by computing the angle each extreme point from a previous iteration makes with its neighbor and the highest reference point (topmost, bottommost, rightmost, leftmost) in its region, for each figure. The algorithm needs to be applied four times, once for each of the regions. For convenience, we choose the region 2 segment of each figure as a sample to do the proof. Notice that any extreme point in region 2 can not be any other type of boundary node except those with  $\{SE, N, W\}$  internal switch connections when figuring boundary segment connections. This will eliminate those unnecessary

comparisons. The algorithm steps are similar for all other regions except that they would be in their corresponding directions. For an  $n \times n$  ORM (Optical Reconfigurable Mesh), where  $n = 2^m (m = 1, 2, \dots, \log N)$ , we have  $m$  iterations. In each iteration, the algorithm is applied on the submesh with the size  $2^m \times 2^m$ . Also, we assume that all boundary nodes are in the extreme point set called  $E$  set at the beginning of the algorithm. The induction method will be used to prove the correctness of the algorithm in phase 2.

*In the base, each combined submesh contains only one node (one pixel). All figure nodes are viewed as an extreme point. We assume that after the iteration  $m - 1$ , the extreme points in each submesh of size  $2^{m-1} \times 2^{m-1}$  have been found. We want to prove that after the iteration  $m$ , the extreme points in each submesh of size  $2^m \times 2^m$  can be found. Now, we look in detail at a subimage with size  $2^m \times 2^m$  in iteration  $m$ .*

The submesh containing the subimage with size  $2^m \times 2^m$  is called  $ORM_m$ . By assumptions, the extreme points in each subimage with size  $2^{m-1}$  have been found. This can be viewed as shown in Figure 4.5, where a new submesh is constructed by four quadrants (Q1, Q2, Q3 and Q4), such that the extreme points in each of the quadrants have been found in the previous iteration. The objective is to find the extreme points in the current submesh combining the four quadrants. In the combining, the vertical mesh boundary in the middle of the new submesh is named a *v line* and the horizontal one a *h line*. We assume that each processor which contains an extreme point is marked as an  $E$  node. Each  $E$  node in  $ORM_{m-1}$  is a candidate for an extreme point in  $ORM_m$ . We call them  $C$  nodes before they are confirmed as  $E$  nodes in the submesh  $ORM_m$ . All  $C$  nodes in an  $ORM_m$  are indexed as  $C(1), C(2), \dots$  clockwise along the segment. The algorithm steps for dropping nonextreme points in  $ORM_m$  consists of two super steps:



**Figure 4.5** Quadrants, the  $v$  line and the  $h$  line in a submesh of ORM

- Find the extreme points in the upper and lower halves of  $ORM_m$  separately.

All ordered extreme points of a figure within a region should have a decreasingly ordered set of angles with respect to the highest point in that region and for that figure. Super step 1 determines and eliminates those non-extreme points (valleys) which violate this rule.

- Find the extreme points in the whole  $ORM_m$ .

Merge the segments in the upper half and lower half of the mesh for each figure.

There are three substeps included in each super step. The two super steps are similar. We concentrate on describing the substeps in the first super step. The significant differences between the two super steps will also be discussed. The details of some steps are in Section 4.2. We define the following notations for describing the phase 2 of the algorithm more clearly and easily:

1.  $C(i)$  is a  $C$  node on a figure's boundary. The  $C$  node has been defined previously.  $C(i + 1)$  is a  $C$  node which is an immediate right neighbor of  $C(i)$  along the boundary in the clockwise direction.

2.  $T(i)$  is the tangent made by  $C(i)$  and  $C(i+1)$ .
3.  $P(j)$  is the cross point of  $T(i)$  and the line  $y=i_{ceil}$ . The index for  $P$  is not the same as  $T$ . It is self indexed from the left to right along the line  $y=i_{ceil}$  depending on the appearances of the cross points.
4.  $D(i)$  is the degree of the angle between  $T(i)$  and the line  $y=i_{ceil}$ .

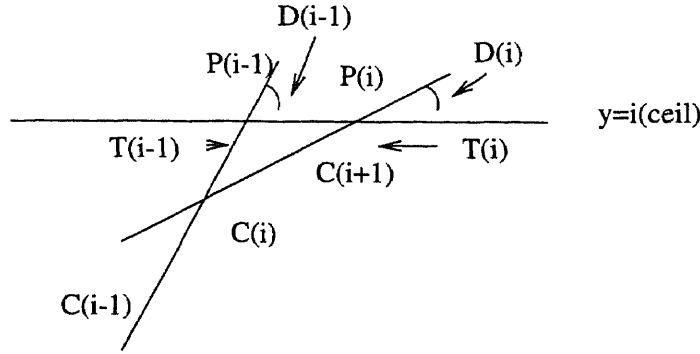
The following are the algorithm steps in phase 2:

1. **(Super step 1)** Find the extreme points in the upper half and lower half of the mesh.

(All ordered extreme points of a figure within a region should have a decreasingly ordered set of angles with respect to the highest point in that region and for that figure. Super step 1 determines and eliminates those non-extreme points (valleys) which violate this rule.)

- (a) Find the  $C$  node which has the smallest  $i$  index among all  $C$  nodes in the segment(s) of a figure  $F$  in each half mesh. This  $i$  index is called  $i_{ceil}$ , meaning ceiling point of a figure in  $ORM_m$ . We define the line  $y=i_{ceil}$  to be a ceiling line. A unique row for each figure in the current mesh to store data in later steps is also found in this step. See the details of the method for finding the unique row and ceiling line of each figure in a submesh in Section 4.2. This nontrivial method is called CONNECT. Those two data of each figure  $F$  are sent back to all figure boundary segments of  $F$  in the submesh.
- (b) Each  $C(i)$  node in a  $C$  segment which received  $i_{ceil}$  in the last step gets the index of its  $C(i+1)$  node neighbor along the clockwise direction. Then a tangent  $T(i)$  is calculated in the node by the indexes of  $C(i)$  and  $C(i+1)$ .

Further, the cross point  $P(j)$  of  $T(i)$  and the line  $y=i_{ceil}$  is found. See Figure 4.6.



**Figure 4.6** P nodes: the cross points of tangents and line  $y$

If the  $P(i)$  is out of the submesh boundary, the  $C(i+1)$  is a nonextreme point and it is dropped in this step.

- (c) Each  $C(i)$  sends the tangent  $T(i)$  (including indexes of two points) to a unique row  $y1=i_{store}$  by the order of the  $P(j)$ . Then the node that receives and stores the tangent is called a  $P$  node and the sender of  $T(i)$  is called  $Sender_{T(i)}$ . This step is done with optical routing using the reconfigurable mirrors. Each  $P$  node gets another tangent from the right  $p$  node neighbor  $P(j+1)$ . Now, each  $P(j)$  holds two tangents. We call the one originally received by  $P(j)$  is  $T(j)$  and the one from its right neighbor is  $T(j+1)$ . Further, we call the  $C$  node which sends  $T(j)$  is  $Sender_{T(j)}$  and the one which sends  $T(j+1)$  is  $Sender_{T(j+1)}$ . Then each  $P(j)$  compares two degrees  $D(j)$  and  $D(j+1)$  made by the angles of  $T(j)$  to the  $y$  line and  $T(j+1)$  to the  $y$  line. If the degrees or the sender's positions violates the the following order rule, the nonextreme points on  $C$  segments in the upper half or lower half of the submesh can be found.

The order rule for the extreme points in region 2 is as follows. The two conditions have to be satisfied at the same time:



- The degrees of two tangents  $T(k)$  and  $T(k+1)$  to the line  $y = i_{ceil}$ ,  $D(k)$  and  $D(k+1)$ , should satisfy the following condition if the sequence of  $D$  on  $y$  are indexed as  $D(1), D(2), \dots$  from left to right depending on the  $P$  node sequence:

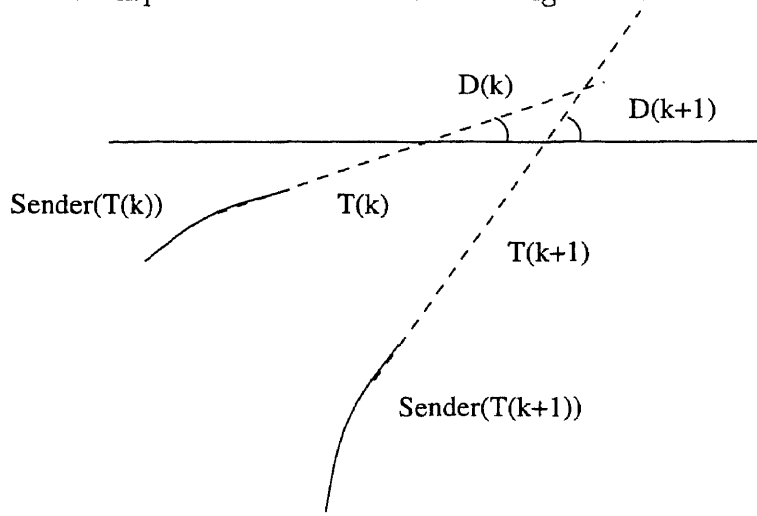
$$D(k) \geq D(k+1)$$

- The indexes of  $Sender_{T(k)}$  and  $Sender_{T(k+1)}$  should satisfy:

$$i_{Sender_{T(k)}} \geq i_{Sender_{T(k+1)}} \text{ and}$$

$$j_{Sender_{T(k)}} \leq j_{Sender_{T(k+1)}}$$

See the example of the rule violation in Figure 4.7.

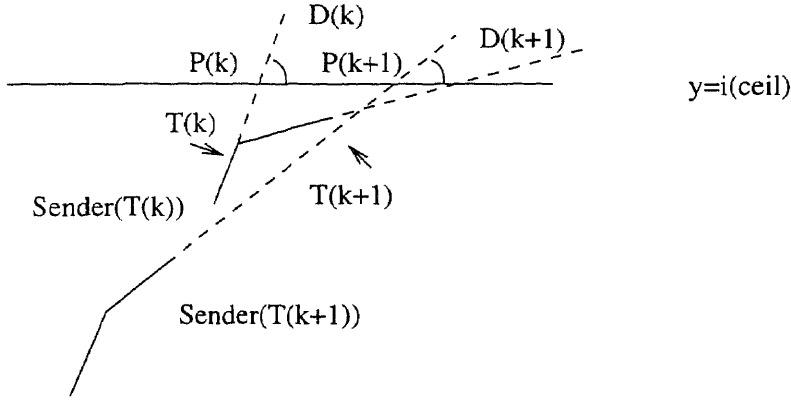


Violation:  $D(k) < D(k+1)$

**Figure 4.7** The example of the rule violation:  $D(k) \leq D(k+1)$

However, from the next example shown in Figure 4.8, we can see that the condition of  $D(k) \geq D(k+1)$  is not enough to recognize the nonextreme points.

In Figure 4.8, the degrees  $D(k)$  and  $D(k+1)$  satisfy  $D(k) > D(k+1)$ , but  $i_{Sender_{T(k)}} < i_{Sender_{T(k+1)}}$  and  $j_{Sender_{T(k)}} > j_{Sender_{T(k+1)}}$ . So, the  $C$  node  $Sender_{T(k)}$  is a nonextreme point in  $ORM_m$ . After this step, if a sequence



**Figure 4.8** The example of the rule violation:  $D(k) \geq D(k+1)$

$S$  of  $C$  nodes are nonextreme points in  $ORM_m$  and if a node  $LN$  is the leftmost node in  $S$ , the  $LN$  must have been recognized after this step.

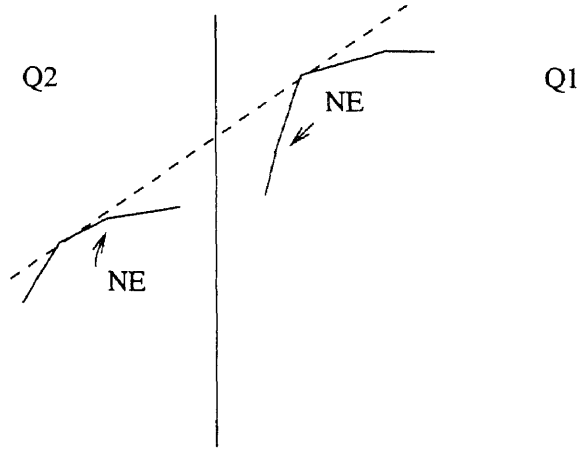
- (d) The index of  $LN$  in  $Q2$  is sent back to all segments of the figure in  $Q2$ . Any  $C$  node which is on the right side of the  $LN$  is a nonextreme point and are dropped.

Similarly, the index of the rightmost node, called  $RN$ , in sequence  $S$  and is in  $Q1$  is sent back to all segments of the figure in  $Q1$ . Any  $C$  node which is on the left side of the  $RN$  is a nonextreme point and dropped. See Figure 4.9. After this step, all nonextreme points in the point sequence are dropped. This step can be done in constant time because all nonextreme points in a segment are  $C$  nodes which are recognized in the  $ORM_{m-1}$  and are in a  $C$  node sequence.

## 2. (Super step 2) Find all extreme points in the current mesh.

(Merge the segments in the upper half and lower half of the mesh for each figure.)

- (a) Similar to 1.a except use a different direction for the  $B$  nodes in the segments which connect to the north bound or south bound of the submesh, or cross the  $h$  line. The new  $i_{ceil}$  found in this step will be



**Figure 4.9** Drop the nonextreme points depending on  $LN$  and  $RN$  nodes

broadcast to a unique row. An extra operation which should be taken by the node is that after finding the smallest  $i$  index (this is done along related rows) for a figure, the  $i$  index should be compared to the  $i_{ceil}$  found in step 1.a. This is easy because  $i_{ceil}$  of step 1 for a figure is stored in a unique column. When a node receives the  $i_{ceil}$ s in both steps 1.a and 2.a, it compares two  $i_{ceil}$ s, chooses the smaller one to be  $i_{ceil}$  in this step and sends the  $i_{ceil}$  back to the  $B$  nodes of this step and  $B$  survivals in the last step. The principle idea for setting up the communication route among different pieces of segments in one figure in the submesh is the same as the way used in step 1.(a).

- (b) Similar to 1.(b).
- (c) Similar to 1.(c).
- (d) Similar to 1.(d).

Each step above uses constant time. So the performance for the phase 2 is  $O(\log N)$ . The total performance of the algorithm is  $O(\log N) + O(\log N) = O(\log N)$ .

□

The important features of the convexity algorithm are the region concept and the tangent disorder concept that can be used to locate the first point in a sequence of nonextreme points on a  $C$  segment. The concept of regions makes it possible to set up the rules to do the comparisons in steps 1.(c) and 2.(c). In each of the  $\log N$  iterations of the algorithm, setting the electro-optical interconnections (using the reconfigurable mirrors as well as the reconfigurable buses on the mesh) based on the order of the cross points made by a sequence of tangent lines and a single ceiling line,  $y = i_{ceil}$ , makes it possible to drop the nonextreme points (valleies) in constant time. Even though most steps in the algorithm are implemented by the reconfigurable mesh, the steps in each of the  $\log N$  iteration can not be run in constant time on a purely electrical reconfigurable mesh.

#### 4.2.2 CONNECT: The Method to Find $i_{ceil}$

The goal of step 1.a in phase 2 is to find the smallest  $i$  index  $i_{ceil}$  among all  $C$  nodes of each figure  $F$  in  $ORM_m$ . Note that there may be many separate pieces of boundary segments of one figure in one  $ORM_m$ , as shown in Figure 4.10.

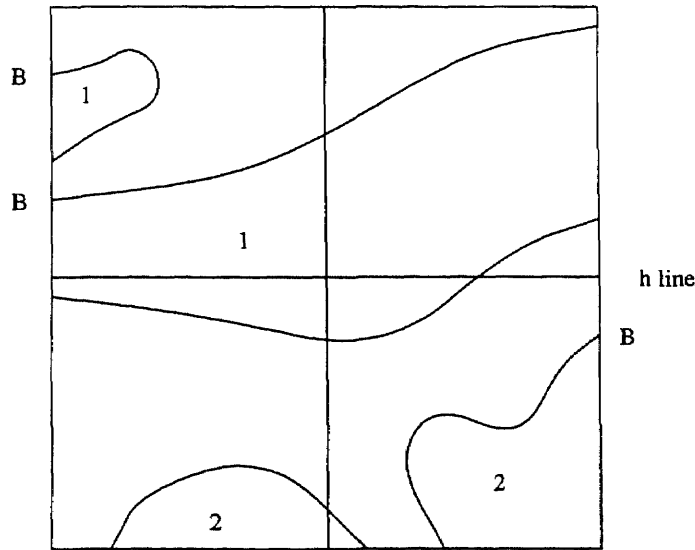


Figure 4.10 Two pieces of figure 1 and two pieces of figure 2 in one submesh

We assume that after each iteration, the node which contains  $i_{ceil}$  is marked. In each upper or lower half of  $ORM_m$ , there are at most two old  $i_{ceil}$ s existing; one in Q1 (Q3) and one in Q2 (Q4). If the segments with two old  $i_{ceil}$ s are connected in the new mesh  $ORM_m$ , it is easy to do the comparison and find the new  $i_{ceil}$  for the figure in this mesh. However, if the two segments are still separated by the mesh boundary, we need to find the route to connect those two segments and then do the comparison. On the other hand, the route to connect those  $C$  segments which do not contain the old  $i_{ceil}$  also needs to be found. The new  $i_{ceil}$  in  $ORM_m$  has to be broadcast to those segments through the route and used in the future steps of the algorithm. So, the problem we need to solve to reach the goal of the step is how to find a route to connect all  $C$  segments of a figure  $F$  in  $ORM_m$ . Once the route is found, the two old  $i_{ceil}$ s can be sent to a common place unique to figure  $F$  to do the comparison. The new  $i_{ceil}$  will be sent to all related segments along the route.

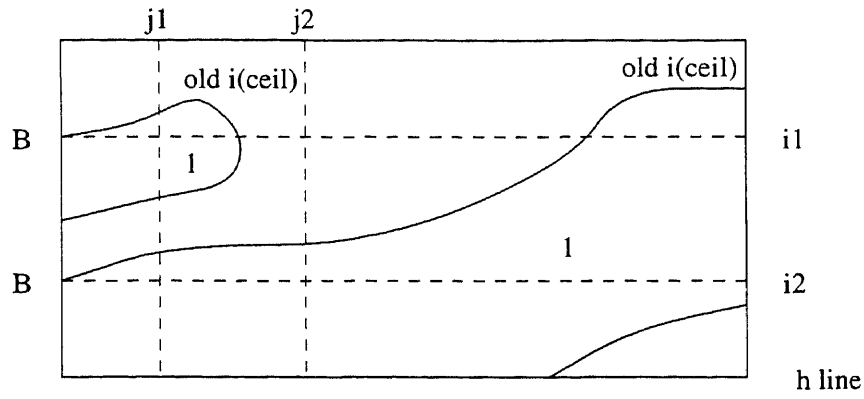
The method used here is named CONNECT. In the algorithm of phase 2, the CONNECT method is both used in step 1.(a) and 2.(a). In 1.(a), all  $C$  segments connecting the west and east submesh boundary or crossing the  $v$  line are processed. After this step, the ceiling line of each figure among those  $C$  segments in the upper part of the mesh (Q1 and Q2) is found. Then the nonextreme points are recognized and dropped. It is similar to the case for the lower part of the mesh (Q3 and Q4). The application areas are different but the processing is the same. The work is done at the same time. In 2.(a), the  $C$  segments for each figure in the upper and lower parts of the mesh are combined. The  $C$  segments which connect to the north or the south mesh boundary, or cross the  $h$  line, are processed in this step. After this step, all nonextreme points of each figure in the current submesh are dropped. The processing of this step is similar to that of step 1.(a).

Before presenting the detailed description of the CONNECT method, we need a little preparation. We know that it is easy for a figure boundary node to recognize

if it is connected to a submesh boundary or crosses the  $v$  line or is in the plane area. Further, by testing the neighbor, the node which is both in the figure and the submesh boundary knows if it is in the upper side of a segment. See Figure 4.10. So, we assume that each this kind of nodes has recognized itself and is called a  $B$  (boundary) node in the following steps. In the example of Figure 4.10, the second piece of figure 1 has only one  $B$  node because the  $B$  node recognition starts from east submesh boundary, then the west, then the  $v$  line.

1. Set row connections in the mesh. Each  $B$  node sends its figure label number  $L(F)$  (label number for figure  $F$ ) to the row. (substeps: (1) send from the west mesh boundary. (2) send from the east mesh boundary. If the  $B$  node in the east boundary has received a data in (1), it means that there is another  $B$  node in the west mesh boundary which is in the same row, say row  $k$ . In this case, the  $B$  node which is in the west mesh boundary has to send its  $L(F)$  to row  $k + 1$ . (3) send from  $v$  line if there is no west or east boundary  $B$  node on the same segment. If the row has received data in either (1) or (2), the data will be sent to the next available row in the south direction.)
2. Set column connections. The nodes on the  $h$  line disconnect the south/north ports. Each diagonal node  $d(i, i)$  which received a  $L(F)$  in the last step sends  $L(F)$  to column  $i$ . If there is another segment of figure  $F$  that sends the label  $L(F)$  from row  $j$ , the node  $d(j, i)$  on column  $i$  will receive the same label  $L(F)$  twice. The other node on column  $i$  will not. See Figure 4.11.
3. Find the northmost node which receives  $L(F)$  twice along the column  $i$ . This is the unique row for the figure  $F$  to store data in 1.(b).

Except for  $L(F)$ , the old  $i_{ceil}$ s are sent along the route in step 1 and 2 by the related  $B$  nodes. So, the two old  $i_{ceil}$ s of  $F$  can also be compared along column



In column  $j1$ , node  $p(i1, j1)$  and  $p(i2, j1)$  both received label 1 twice; once from step 1 and once from step 2. Column  $j2$  does the same.

**Figure 4.11** Find the route for each figure

$i$  in this step. In the case of Figure 4.11, row  $i1$  will be the unique row used by all  $C$  segments of figure 1 in step (c) in the basic algorithm.

4. Send  $i_{ceil}$  back to each  $B$  node. This is done by sending the data along the route in 1 and 2 by the reverse direction.

The uniqueness of the row  $y1=i_{store}$  for each figure is obvious. Since each  $B$  node uniquely occupies a row to send its figure label, and no more than one figure uses the same column to recognize the northmost  $B$  node among all  $B$  nodes of that figure, so each figure will find a unique row in the mesh.

### 4.3 Conclusion

A nontrivial  $O(\log N)$  parallel algorithm for the multi-image convexity problem on ORM was presented in this chapter. The work presented showed the advantages offered by the combination of optical and electric interconnections as compared to the traditional electrical reconfigurable systems or the pure optical interconnection systems. The algorithm obtained is the fastest known parallel convexity algorithm.

## CHAPTER 5

### APPLICATION SPECIFIC DESIGN OF THE OPTICAL COMMUNICATION TOPOLOGY IN ORM

We have mentioned in Chapter 2 that the third type of communication mechanism in ORM is slower than the other two because it requires mechanical movements per reconfiguration. It is important to find the optical interconnection topology and set it up before the execution of a given task begins. We present a methodology named CONST to construct an efficient topology for the optical interconnections in ORM for each given task.

#### 5.1 Introduction

Many studies have been done for designing various efficient parallel architectures to solve application problems. Most electrically interconnected parallel architectures designed are only suitable for certain types of problems. Normally, these problems are those whose associated data flow graphs match the topology of the architecture. Parallel architectures with optical interconnections have been recently studied by a number of scientists [8, 25] as a way for designing reconfigurable topologies that would fit any desired problem to be solved. However, reconfiguring optical interconnections is not an easy task, they have a low switching rate in case active optical switching elements are used, or require a large number of resources if passive elements are used [NPPOI 95, 96]. Designing an efficient parallel architecture with electrical and optical interconnections for any given problem is the topic of this chapter.

Now, by using ORM, we present a methodology named CONST to construct an efficient topology for the optical interconnections in ORM for each given task. We have mentioned that the third type of communication in ORM which uses reconfigurable optical interconnections is slow as compared to the other two. However, there are two advantages of this communication mechanism. It can be reconfigured as a



desired topology and it can route data in unit time on the topology once the optical interconnections are set up. The strategy is to set up an efficient optical interconnection before the execution of a task. Determining a configuration that would be suitable for the entire configuration of a task execution is studied in this chapter.

For analyzing the properties of a given task graph, a model named Cluster-M is used in the methodology. Cluster-M is a programming tool that facilitates the design and mapping of portable parallel programs [9]. Cluster-M has three main components: the specification module, the representation module and the mapping module. In the specification module, machine-independent algorithms are specified and coded using the program composition notation (PCN) [20] programming language [14]. Cluster-M specifications are represented in the form of a multilayer clustered task graph called a Spec graph. Cluster-M represents a multilayer partitioning of a system graph called a Rep graph. At every partitioning layer of the Rep graph, there are a number of clusters called Rep clusters. Each Rep cluster represents a set of processors with a certain degree of connectivity. Given a task (system) graph, a Spec (Rep) graph can be generated using one of the Cluster-M clustering algorithms. In the mapping module, a given Spec graph is mapped onto a given Rep graph. For a detailed description of Cluster-M, see the appendix in this thesis.

Using Cluster-M, we present a methodology named CONST which can determine how the optical topology in the ORM is to be set for a given problem. The goal of the construction is that for a given ORM with  $N$  processors and a given task graph with  $M$  nodes, we want to come up with a communication topology for the optical interconnections in ORM to execute the task efficiently. These optical interconnections are set before the execution begins and are not changed during the task execution.

The rest of the chapter is organized as follows. In Section 5.2, we give the preliminaries. The CONST methodology is presented in Section 5.3. Section 5.4 is for concluding remarks.

## 5.2 Preliminaries

For constructing a proper ORM machine for a given task, we need some assumptions on the computation model and tools. Those assumptions do not affect the features and performances of the original models. We also introduce the terminologies and definitions used in the methodology.

The following are the assumptions.

- An update is needed for the basic ORM to support the methodology: each PE has four free space optical transmitters to the other nodes and one optical receiver. The functions for each of them will be introduced later.
- For a given task graph, the Spec graph obtained by using the cluster-M model is the input of our methodology. Originally, there are five parameters in each cluster  $C$  after clustering:
  1. The size of the cluster  $C$  which is the number of subclusters of  $C$  that can be computed in parallel.
  2. The maximum sequential computation amounts (time) in  $C$ .
  3. The total amount of communication from layer 1 of the Spec graph to the layer  $L$  in which  $C$  resides.
  4. The average communication amount at layer  $L$  in  $C$ .
  5. The computational type of  $C$ : SIMD or MIMD.

In addition to the above five parameters in each cluster, we add one parameter  $L$  on each cluster  $C_i$ .  $L$  is a list. Each entry in the list is a cluster number

and a communication time. If any cluster  $C_i$  has a communication edge to  $C_j$ , there is an entry in  $L$  of  $C_i$  for  $j$  and the time taken to transfer the data from  $C_i$  to  $C_j$ . This can be done in the clustering easily without hurting the performance. Here, we assume that the clustering algorithm has done this work before executing CONST.

- The size of the ORM is independent of the number of tasks.

The following are terminologies and definitions used in the description:

1. BFS result graph in the algorithm

Breadth-first search (BFS) is applied to the Spec graph in the methodology to determine the relationships of clusters in the graph. In the BFS result graph  $G$ , each node is a cluster and each edge  $(C_i, C_j)$  in  $G$  represents the communication from cluster  $C_i$  to cluster  $C_j$ . This edge may be used more than once if there is more than one task in  $C_i$  requiring to communicate to the tasks in  $C_j$  in different time. There are three types of edges in the graph. A *forward edge* is the edge from a parent cluster to a child cluster. A *side edge* is the edge from a cluster to a sibling cluster (they are in the same level). A *back edge* is the edge from a child to a parent or an ancient. Any cluster which has no forward edge as an incoming edge is defined as a starting node in the graph. There may be more than one starting node in  $G$ .

The level in  $G$  is defined as follows. All starting nodes are in level 1 of  $G$ . Any child with its parent in level  $i$  is in level  $i+1$ . If a child has more than one parent, its level number follows the parent with the lowest level number among these parents. This will be recognized automatically in the breadth-first search.

In each level of the cluster graph, except the first level, the nodes are categorized into two types:

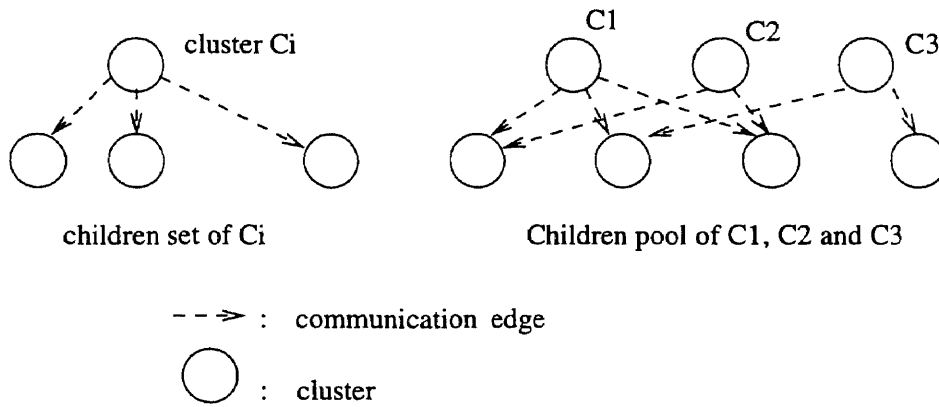
- Type 1: The node has only one parent
- Type 2: The node has more than one parent

## 2. Children set

A set of type 1 nodes. All members of the set have the same parent  $C_p$  and  $C_p$  is the only parent it has.

## 3. Children pool

A set of type 2 nodes. For each children pool, there is a unique parent set. This means that no parent node has two children in different children pools. Figure 5.1 shows examples of a children set and a children pool.



**Figure 5.1** A children set and a children pool

## 4. Time conflict and time conflict free

If two parents of level  $i$  communicate to their children in the same time period, it is a time conflict. We call a child cluster  $C$  involved in a time conflict communication a time conflict cluster. Otherwise,  $C$  is a time conflict free cluster.

### 5.3 The CONST Methodology

To come up with a suitable topology for executing a given task we first need to analyze the properties of the problem. Using the Cluster-M clustering algorithm, we can trace the communication requirement of a given problem at different steps of the execution. Clusters are created as the communications in the task graph are traced and analyzed. In this algorithm, there are two phases. In phase 1, we decluster a given Spec graph until the best cluster layer can be found for a given number of processors. In phase 2, the breadth first search methodology is performed on the Spec graph. The optical interconnection topology is then determined depending on the BFS result graph  $G$ .

#### 5.3.1 Phase 1: (Declustering)

For a given Spec graph and a given number of  $N$  processors, we do a mapping of the Spec graph to an  $N$ -node fully connected Rep graph. The mapping algorithm in [10] is used in which a test is done within the declustering for each cluster  $C$ . In the algorithm, the sequential computation time of  $C$  is compared to the computation and communication time after the declustering of  $C$ . This test will tell if the declustering of  $C$  should be done or not. After this phase, we can find the clustering results of the task graph which is the best one for the given number of processors.

#### 5.3.2 Phase 2: (Assigning)

The object processed in this phase is the cluster graph obtained from phase 1. The breadth-first search is applied on the cluster graph. The search starts from the node which contains the starting node of the whole task. After that, the forward edges, side edges and back edges are recognized. The advantages of the ORM are: the destinations of the optical interconnections can be chosen freely (but fixed during the task running) and the mesh system can be segmented during the task running.

CONST takes those advantages by using the following principle to assign processors to clusters. Once the assignment is done, the optical interconnections can be set up.

The assigning method is that the clusters are sequentially located in the processors level by level. All clusters in the second level follow the clusters of the first level and so on. We index the mesh as a linear sequence system. The first PE in the sequence is the one in the first row and the first column. It is indexed as  $P(1)$ . The sequence is forwarded to the right and then wrapped snake-like. The clusters are assigned to the processors one by one from  $P(1)$  to  $P(n^2)$ .

By the definition, each level of the graph has at least one children set and/or children pool. We want to solve two problems in our methodology:

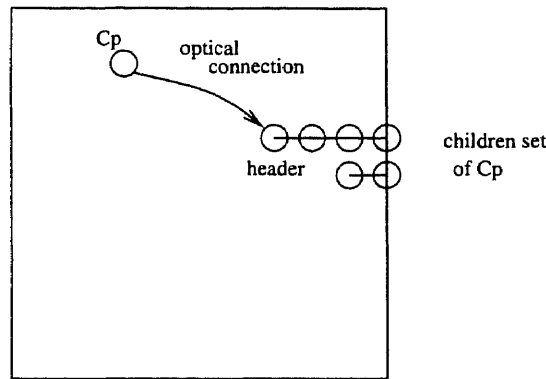
- What is the order of the clusters in a children pool or a children set?
- What is the order of children pools and children sets in each level?

We need to find good solutions for them so that the system routing delay can be minimized when the task runs. After we solve those problems, it will be clearer to see how the clusters are to be located in the system and to decide how to make the optical connections from one node to the others. The following are steps in the methodology:

#### 1. Assignment of Children Set

Conceptually, we know that each parent has at most one children set in its next level. There may be more than one children pool in each level. If each parent has a direct interconnection to its children, there would be no routing delay during the task execution. However, this kind of communication is impossible from electrical connections if the number of children of one cluster is  $O(N)$ . On the other hand, a routing delay would be introduced if the communications need go through intermediate nodes. A reconfigurable free space optical interconnection is ideal to do the global communication. The problem is that there

are still not enough optical interconnections from one node to many others in our system, and we do not want the optical connections to be changed often. So the idea is that we assign the clusters of a children set  $S$  onto a set of processors in which processors can communicate to each other through electric buses easily and efficiently. In other words, we want keep a good locality for the clusters in a children set. In CONST, the nodes in a set are assigned into a sequence of PEs connected by the row buses. Then we let each parent  $C_p$  have an optical interconnection to one of the nodes in its children set. This node is called the *header* of the set. See Figure 5.2.



**Figure 5.2** The optical connection to the header of children set

We can choose the cluster which has the smallest processor index in  $S$  as the header of that children set. Assume that the parent of  $S$  is  $C_p$ . Then one of the optical interconnections from  $C_p$  will be to the header of  $S$ . When  $C_p$  wants to send data to a child  $C_c$ , it sends data to the header first by optical interconnection. Then the system does the children set segmenting and the data can be sent to  $C_c$  on the segment. Basically, the order of clusters in a children set is not critical. However, considering the amount of communication, we want the one with the largest communication amount to be the header, the one with the second largest communication amount to be the right neighbor of

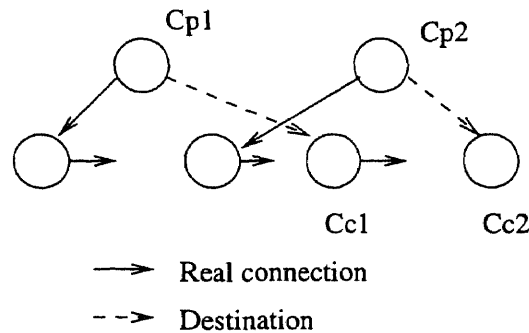
the header, and so on. The sorting by the communication time could be done in the clustering.

Summary: The order of clusters in a children set are ordered decreasingly by the communication amount in the set. Each set has a leader which is stored in the smallest indexed PE in the set. There is a fixed optical interconnection from each parent to the header of its children set.

## 2. Assignment of Children Pool

The principle is the same as the one for a children set, the clusters in a pool will be allocated in a group of PEs which are sequentially connected by the electric bus. However, the strategy to decide the order of the clusters in a pool is more complicated than the one for a children set because it needs to avoid the transmission conflict caused by more than one parent sending data to the children in the same pool.

Figure 5.3 shows an example of the conflict:



**Figure 5.3** The example of the conflict

$C_{p1}$  sends data to its child  $C_{c1}$  in the time period  $T1$  and  $C_{p2}$  to  $C_{c2}$  in  $T2$ , there is a conflict if an overlap of  $T1$  and  $T2$  exists.

For testing the existence of conflicts, CONST prescans the clusters to get the starting time of the communication(s) for each cluster in the result graph of the declustering in phase 1.



Assume that  $C_1, C_2, \dots, C_m$  are the children of the cluster  $C_p$ . They must be in the same children pool by the pool definition. Further, we assume that a group  $F$  of clusters  $C_i, i < m$ , among those  $m$  child clusters of  $C_p$  are time conflict clusters. We will assign the clusters in  $F$  onto sequentially connected processors in the mesh. The first cluster in the sequence is the header of  $F$ . In a children pool we can have several groups of time conflict clusters. Each group has a header and its own parent. This assignment of conflict groups minimizes the situation in Figure 5.3. If a cluster  $C_i$  has a time conflict in two different time period by different parents, it is put into the group with the parent which has the larger amount of communication to  $C_i$ . All time conflict free clusters in the pool are allocated following the last group of time conflict clusters in the pool. There is no routing delay for conflict free clusters anyway.

Summary: The order of the clusters in a children pool is as follows:

- (a) The grouped time conflict clusters are ordered by different parents.
- (b) If a cluster has more than one time conflict by different parents, it should be in the group with the parent which has the largest communication amount.
- (c) All time conflict free clusters follow the last time conflict group in the pool.

The order of children sets and children pools in a level is not important, since each parent has one optical connection to its children set and one for children group in the pool.

### 3. Back Edges and Side Edges

After the processor assignment, the clusters in each level are on a reconfigurable bus segment on which the processors can be easily connected. For the

case of communication to the ancestors through back edges, we need to store the assignment information (the processor address of the destination) for each outgoing edge in the cluster no matter the type of the edge. The consideration about the back edges is as follows.

- (a) If each node in level  $i$  has not more than one back edge, one of the optical interconnections will be dedicated to this edge.
- (b) If more than one back edge from one node is required, but the destinations are in the same level, the optical interconnection is built to one of the parents (or ancestors). The rest of the communications can be done by sending data through the optical connection to the destination node level, and then the mesh bus to the destination. This is easy because the nodes in the same level are in a sequentially connected bus segment and one communication occurs at one time for a cluster.
- (c) If the number of back edges in a level  $i$  is larger than the number of nodes in the level and back to many different levels, we may need to assign the optical interconnection in each node to connect to one of the above level. The result is that the total back connections can cover all levels on the level above level  $i$ .

The destination of a side edge from a cluster  $C_i$  is a cluster  $C_j$  which is in the same level as  $C_i$ . When separating the time conflict group in a level, the time conflicts caused by any side edge should be also considered with the forwarding edges. Since the destination of a side edge is in the same level as the sender, the routing can be at least restricted into a segment for a level.

#### 4. Setting Optical Interconnections

Now, we can set the optical interconnections in ORM. Each processor  $P$  can have maximally three optical interconnections to other processors except the

fixed one. Or we can say that each cluster  $C$  can have maximally three optical communication paths to other clusters:

- to the header of its children set
- to the header of its time conflict group in the children pool
- to a parent or an ancestor
- to a brother cluster

## 5.4 Conclusion

In this chapter, we presented a methodology CONST which is used to reconfigure a suitable optical interconnection in ORM for efficient execution of a given task. Cluster-M was used to analyze the properties of a given task graph, then based on that the CONST methodology produced a suitable pattern for reconfiguring the optical interconnections. These interconnections are not changed during the execution. However, they provide sufficient connectivity among all the processors for the entire duration of the execution.

## CHAPTER 6

### CONCLUSION AND FUTURE WORKS

In this thesis, we discussed the limitations of electrical interconnections and studied an emerging technique, optical interconnections, which is a remedy of those limitations. In Chapter 2, the existing optical model of computation (OMC) and three implementations were introduced. The efficient algorithms for application problems on the model were also discussed. We presented an efficient opto-electrical parallel architecture named ORM (Optical Reconfigurable Mesh) which is an implementation of OMC in chapter 3. In this architecture, processors can communicate through both optical reconfigurable interconnections and electrical reconfigurable buses. The two layers of the architecture and the data movements in the three communication mechanisms in ORM were described. The three communication mechanisms are 1. Electrical interconnections through the electrical reconfigurable mesh in ORM, 2. Electrical-optical interconnections through electrical buses and fixed passive optical interconnections and 3. Reconfigurable optical interconnections.

For illustrating the power of ORM, two efficient parallel algorithms on ORM were presented in Chapter 4 and Chapter 5. In Chapter 4, the first  $O(\log N)$  time algorithm for finding the convex hulls of all figures in an  $N \times N$  0/1 image on ORM was proposed. The algorithm consists of two phases. In phase 1, the boundary segment of each figure in the image is cut into four subboundary segments. Each subsegment represents a region of the figure. The concept of the region largely restricts the number of nodes which participate in the comparisons and makes the extreme point recognitions simple and easy. In phase 2, the divide and conquer methodology is used to recognize the valleys in each combined mesh boundary region. Chapter 5 gave a methodology named CONST to construct an efficient topology for the optical interconnections in ORM for each given task. For analyzing the properties of each given task, the Cluster-M clustering algorithm was used. There are also two

phases in the algorithm. In phase 1, we decluster a Spec graph which was obtained by the Cluster-M clustering algorithm until the best cluster layer can be found for a given number of processors. In phase 2, the breadth first search (BFS) methodology is performed on the Spec graph. The optical interconnection topology is then decided depending on the BFS result graph  $G$ .

What follows is future works for further exploring the advantages of ORM or properties of the Optical Model of Computation (OMC).

### 1. Geometric Problems

More geometric problems can be focused on. The problems include histogram computation, finding the nearest neighbor figure, component labeling, finding maximum and minimum of a set of inputs, etc. Similar research for graph problems and other image processing problems can also be done.

### 2. CRCW PRAM Simulation

Currently, CRCW can be implemented in  $N$  processors on ORM. Each PE has only constant memory cells, or  $O(N)$  cells with the restriction on the accessing range of the memory module for multiple read requests. If the number of memory cells in one module of PE/memory pair is large than  $N$ , and any memory accessing in different modules is allowed in one step, then this communication step needs  $O(N)$  time in the worst case. So, how to simulate CRCW PRAM of large memory size or on more PEs on ORM is another research topic proposed to be studied.

### 3. Using Wavelength Division

In ORM, many processing element interconnections are electric buses. We know that a single optical fiber can support a large number of independent, selectable channels to link the processing elements in a system. And from Chapter 1,

many optical versions of existing computing models are implemented by using wavelength division through optical fiber to substitute for the electrical buses.

The questions are as follows:

- (a) Can we find some advantages for using wavelength division to connect the mesh?
- (b) How do we do the connections?
- (c) What are the performance issues after using the wavelength division?
- (d) How can we efficiently use optical free space and wavelength division in one system?

#### 4. Systolic ORM

A systolic reconfigurable mesh (SRM) [16] is a variant of the reconfigurable mesh. The SRM combines aspects of systolic arrays with that of a general reconfigurable mesh model. The implementation of an optical systolic reconfigurable mesh is another interesting research topic that we propose to study in the future.

## APPENDIX A

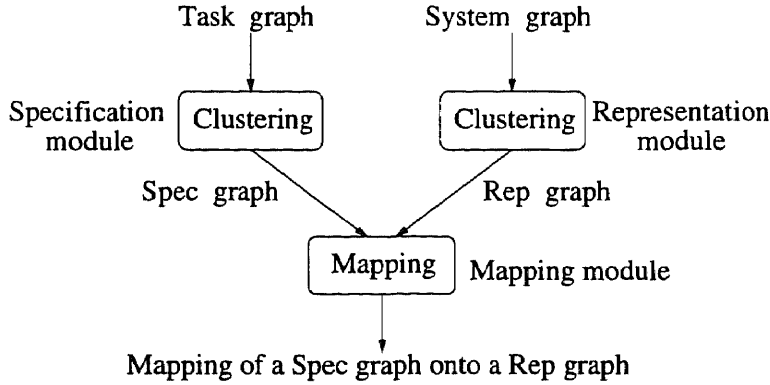
### CLUSTER-M PRELIMINARIES

In an earlier publication [9] a set of clustering and mapping algorithms was presented for the preliminary version of the Cluster-M mapping module. Those algorithms can handle only “uniform” arbitrary task and system graphs. The algorithms presented in this paper are nontrivial extensions of the Cluster-M uniform algorithms for mapping “nonuniform” arbitrary task graphs onto “nonuniform” arbitrary system graphs. In the following, we first give an overview of the Cluster-M tool and then present basic concepts used both in uniform and nonuniform Cluster-M clustering and mapping algorithms. A set of parameters used in the nonuniform clustering and mapping algorithms is presented in the Section Clustering Parameters.

#### Cluster-M

Cluster-M is a programming tool that facilitates the design and mapping of portable parallel programs [9]. Cluster-M has three main components: the specification module, the representation module and the mapping module. In the specification module, machine-independent algorithms are specified and coded using the Program Composition Notation (PCN [20]) programming language. Cluster-M specifications are represented in the form of a multilayer clustered task graph called the Spec graph. Each clustering layer in the Spec graph represents a set of concurrent computations called Spec clusters. A Cluster-M Representation represents a multilayer partitioning of a system graph called the Rep graph. At every partitioning layer of the Rep graph, there are a number of clusters called Rep clusters. Each Rep cluster represents a set of processors with a certain degree of connectivity. Given a task (system) graph, a Spec (Rep) graph can be generated using one of the Cluster-M clustering algorithms. The clustering is done only once for a given task (system) graph independent of any system (task) graphs. It is a machine-independent (application-independent)

clustering; therefore it is not necessary for it to be repeated for different mappings. For this reason, the time complexities of the clustering algorithms are not included in the time complexity of the Cluster-M mapping algorithm. In the mapping module, a given Spec graph is mapped onto a given Rep graph. This process is shown in Figure A.1. In an earlier publication [9] two Cluster-M clustering algorithms and a mapping algorithm were presented for uniform graphs.



**Figure A.1** Cluster-M mapping process.

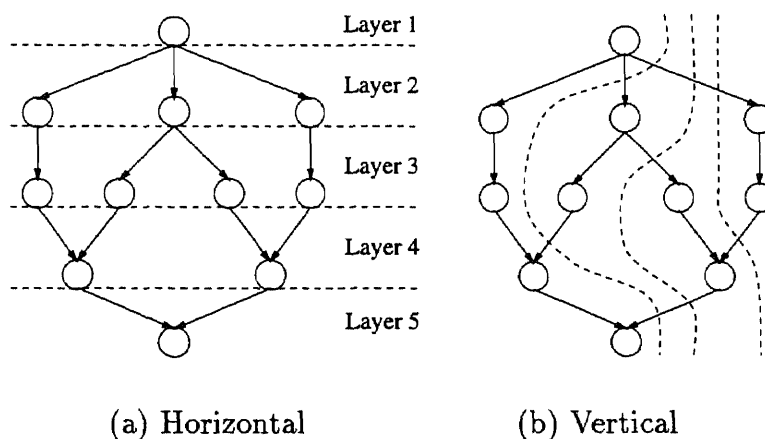
### Basic Concepts

There are a number of reasons and benefits in clustering task and system graphs in the Cluster-M fashion. Basically Cluster-M clustering causes both task and system graphs to be partitioned so that the complexity of the mapping problem is simplified and good mapping results can be obtained. In clustering an undirected graph, completely connected nodes are grouped together forming a set of clusters [9, 14]. Clusters are then grouped together again if they are completely connected. This is continued until no more clustering is possible. When an undirected graph is a task graph, then doing this clustering essentially identifies and groups communication-intensive sets of task nodes into a number of clusters called Spec clusters. Similarly for a system graph, doing the clustering identifies well-connected sets of processors into a number of clusters called Rep clusters. In the mapping process, each of the communication intensive sets of task nodes (Spec clusters) is to be mapped



onto a communication-efficient subsystem (Rep cluster) of suitable size. Note that mapping of undirected task graphs onto undirected system graphs is referred to as the allocation problem. An earlier publication [9] showed that Cluster-M clustering and mapping algorithms can lead to good allocation results. It compared its results with Bokhari's  $O(N^3)$  algorithm and showed that its algorithm has a lower time complexity of  $O(MN)$ , where  $M$  and  $N$  are the number of nodes in the task and system graphs, respectively.

Clustering directed graphs (i.e., directed task graphs) produces two types of graph partitioning: horizontal and vertical. Horizontal partitioning is obtained because, as part of clustering, we divide a directed graph into a layered graph such that each layer consists of a number of computation nodes that can be executed in parallel and a number of communication edges incoming to these nodes. This is shown in Figure A.2(a). The layers are to be executed one at a time. Therefore, the mapping is done one layer at a time. This significantly reduces the complexity of the mapping problem since the entire task graph need not to be matched against the entire system graph.



**Figure A.2** Horizontal and vertical partitioning of a task graph.

Vertical graph partitioning is obtained because, as part of the clustering, the nodes from consecutive layers are merged or embedded. All the nodes in a layer are merged to form a cluster if they have a common parent node in the layer above or

a common child node in the layer below. Doing this traces the flow of data. This information will be used later as part of the mapping so that the tasks are placed onto the processors in a way that total communication overhead is minimized. For example, to avoid unnecessary communication overhead, the task nodes along a path may be embedded into one another so that they are assigned to the same processor. The effect of this type of partitioning is shown in Figure A.2(b).

Both horizontal and vertical graph partitionings are accomplished by performing the clustering in a bottom-up fashion. The Cluster-M mapping will then be performed in a top-down fashion by mapping the Spec clusters one layer at a time onto the Rep clusters. The next two sections show how these clustering and mapping ideas work for nonuniformly weighted graphs. The nonuniform algorithms shown in this chapter are nontrivial extensions of the Cluster-M uniform algorithms presented in an earlier publication [9].

### Clustering Parameters

In the following, we present a set of parameters needed for nonuniform version of Cluster-M clustering and mapping. The first set is for representing a portable parallel program and the other for specifying the organization of the underlying heterogeneous architecture or suite.

#### Machine-Independent Program Parameters:

A given parallel program consists of a sequence of steps such that in each step a number of computations can be done concurrently. Each step is called a layer. These concurrent computations for a given step (layer) can each be presented by a cluster called a Spec cluster. The  $m$ th Spec cluster at layer  $u$  is denoted by  $S_m^u$  and associated with the following parameters.

- $\sigma S_m^u$  The size of  $S_m^u$  which is the maximum number of nodes in this cluster that can be computed in parallel.
- $\delta S_m^u$  The maximum sequential computation amounts (i.e., the maximum number of clock cycles required to execute all the instructions sequentially using a baseline computer) in  $S_m^u$ .
- $\Pi S_m^u$  The total amount of communication from layer 1 to layer  $u$  of  $S_m^u$ .
- $\pi S_m^u$  The average communication amount at the layer  $u$  in  $S_m^u$ .
- $\rho S_m^u$  The computational type of  $S_m^u$ . Its value is set to 0 for single instruction stream, multiple data stream (SIMD) type and 1 for multiple instruction stream, multiple data stream (MIMD) type<sup>1</sup>.

### Program-Independent Machine Parameters:

Any heterogeneous architecture can similarly be represented in a multilayered format such that each layer presents a set of processing units which are completely connected. Each processing unit is represented by a cluster called a Rep cluster. The  $n$ th Rep cluster at layer  $v$  is denoted by  $R_n^v$  and associated with the following parameters.

- $\sigma R_n^v$  The number of processors contained in  $R_n^v$ .
- $\delta R_n^v$  The average computation speed of the processors in  $R_n^v$ .
- $\Pi R_n^v$  The total data transmission rate including the transmission rate over the links (communication bandwidth) and over the nodes (switching latency) from layer 1 to  $v$  in  $R_n^v$ .
- $\pi R_n^v$  The average data transmission rate at layer  $v$  of  $R_n^v$ .

---

<sup>1</sup>All the examples of the problems and systems studied in this paper are assumed to be of MIMD-type. However, in heterogeneous computing, it is possible to have a mix of SIMD and MIMD nodes both in the task and the system.

$\rho R_n^v$  The computational type of the Rep cluster. Its value is set to 0 for SIMD type and 1 for MIMD type.

## NON-UNIFORM CLUSTERING

This section first presents a clustering algorithm to be used for directed task graphs independent of any system graphs and then presents another one for undirected system graphs independent of any task graphs. Both algorithms are done only once for any given task or system graph and are not repeated as part of the mapping process.

### Clustering Directed Task Graphs

A task can be represented by a directed graph  $G_t(V_t, E_t)$ , where  $V_t = \{t_1, \dots, t_M\}$  is a set of task modules to be executed and  $E_t$  is a set of edges representing the partial orders and communication directions between task modules. A directed edge  $(t_i, t_j)$  represents a data communication from module  $t_i$  to  $t_j$  and  $t_i$  must be completed before  $t_j$  can begin, where  $1 \leq i, j \leq M$ . Each edge  $(t_i, t_j)$  is associated with  $D_{ij}$ , the amount of data required to be transmitted from module  $t_i$  to module  $t_j$ , where  $D_{ij} \geq 1$ . Each task module  $t_i$  is associated with its amount of computation  $A_i$ , that is, the number of instructions contained in  $t_i$ . Note that  $A_i \geq 1$  and  $D_{ij} \geq 1$  if there exists an edge  $(t_i, t_j)$ , for  $1 \leq i, j \leq M$ . If a directed edge  $(t_i, t_j)$  exists,  $t_i$  is called a parent node (module) of  $t_j$  and  $t_j$  a child node (module) of  $t_i$ . If a node has more than one child, it is called a fork-node. If a node has more than one parent, it is called a join-node. A task graph is divided into a number of layers, so that all nodes in a layer can be executed concurrently.

A clustering algorithm called clustering nonuniform directed graphs (CNDG) is shown in detail in Figure A.3. This nonuniform algorithm is designed as an extension to the uniform clustering algorithm presented in an earlier publication [9]. The

```

Clustering Nonuniform Directed Graphs (CNDG) Algorithm
Divide the directed graph into horizontal layers in a top-down fashion
For each node at layer 1 do
    Make it into a cluster and calculate its parameters
For each of the other layers in top-down sequence do
begin
    For all edges  $(t_i, t_j)$  do
    begin If  $t_i$  is a fork-node then
        begin Embed the child node with the largest edge weight to  $t_i$ 
            If the child nodes of  $t_i$  are not in a cluster then
            begin Merge them with  $t_i$  into a cluster
                Calculate the parameters of the new cluster
            end
        end
    end
    If  $t_j$  is a join-node then
    begin Embed the child node with the largest edge weight to  $t_j$ 
        If the parent nodes of  $t_j$  are not in a cluster then
        begin Merge them with  $t_j$  into a cluster
            Calculate the parameters of the new cluster
        end
    end
    end
end
end
end

```

**Figure A.3** Clustering Nonuniform Directed Graphs (CNDG) algorithm

nonuniform algorithm has been designed in such a way that it is a generalization of the uniform algorithm. For clustering nonuniform directed graphs, a quintuple of parameters  $(\sigma S_m^u, \delta S_m^u, \Pi S_m^u, \pi S_m^u, \rho S_m^u)$  from the Cluster-M model described in the last section is associated with the  $m$ th Spec cluster at layer  $u$  denoted by  $S_m^u$ . The clustering is done layer by layer. At layer 1, a node with computation amount  $A_i$  is a cluster by itself with parameters  $(1, A_i, 0, 0, 0)$  for SIMD type or  $(1, A_i, 0, 0, 1)$  for MIMD type. Then for other layers, the nodes are clustered as follows. If a node is a join-node, we first embed it onto one of its parent nodes that has the largest weighted edge connecting to this join-node. If multiple parent nodes have edges with the same largest weight, we randomly select one of them. When a node with a computation

amount  $A$  is to be embedded onto  $S_m^u$ , then these parameters are updated to  $\sigma S_m^u$ ,  $\delta S_m^u + A_i$ ,  $\Pi S_m^u$ ,  $\pi S_m^u$ , and  $\rho S_m^u$ . We then merge all its parent nodes into a new cluster denoted by  $S_1^{u+1}$ . This is shown in Figure A.4, where a join-node at layer  $(u + 1)$  with computation amount  $A$  has  $n$  parent nodes  $S_1^u, S_2^u, \dots, S_n^u$  at layer  $u$ . The communication amount between the join-node and one of its parent nodes  $S_i^u$  is denoted by  $D_i$ , where  $1 \leq i \leq n$ . Also,  $D_1 = \max_{1 \leq i \leq n} D_i$ . The new cluster  $S_1^{u+1}$  is generated by embedding the join-node to  $S_1^u$  and merging it with all the other parent nodes. The first four parameters of  $S_1^{u+1}$  can be computed as follows.

$$\sigma S_1^{u+1} = \sum_{i=1}^n \sigma S_i^u \quad (\text{A.1})$$

$$\delta S_1^{u+1} = \max(\delta S_1^u + A, \delta S_2^u, \dots, \delta S_n^u) \quad (\text{A.2})$$

$$\Pi S_1^{u+1} = \sum_{i=1}^n (\Pi S_i^u + D_i) - D_1 \quad (\text{A.3})$$

$$\pi S_1^{u+1} = \frac{\sum_{i=2}^n D_i}{n - 1} \quad (\text{A.4})$$

If a node is a fork-node, we will embed one of its child nodes to this fork-node. The child node is selected so that it has the largest weighted edge connecting to the fork-node. If multiple child nodes have edges with the same largest weight, we randomly select one of them. We then merge the rest of the child nodes with the fork-node into a new cluster. As shown in Figure A.5, a fork-node  $S_1^u$  at layer  $u$  has  $n$  child nodes at layer  $(u + 1)$ . These child nodes have computation amounts  $A_1, A_2, \dots, A_n$ , and the communication amounts between the fork-node and each of them are  $D_1, D_2, \dots, D_n$ , respectively. Similar to the case of the join-node,  $D_1 = \max_{1 \leq i \leq n} D_i$ . Then the node with the computation amount  $A_1$  is embedded into the fork-node before we merge the fork-node with all the other child nodes to generate the new cluster  $S_1^{u+1}$ . The first four parameters of  $S_1^{u+1}$  are then computed as follows.

$$\sigma S_1^{u+1} = \max(\sigma S_1^u, n - 1) + 1 \quad (\text{A.5})$$

$$\delta S_1^{u+1} = \max(\delta S_1^u + A_1, A_2, \dots, A_n) \quad (\text{A.6})$$

$$\Pi S_1^{u+1} = \Pi S_1^u + \sum_{i=2}^n D_i \quad (\text{A.7})$$

$$\pi S_1^{u+1} = \frac{\sum_{i=2}^n D_i}{n-1} \quad (\text{A.8})$$

For both fork and join nodes, the fifth parameter,  $\rho S_m^u$ , is determined as follows. As an MIMD cluster is merged with an SIMD or MIMD cluster, the computation type of the new generated cluster is MIMD. When two SIMD clusters are merged then the computation type of the new cluster is decided by their computational form (addition, subtraction, multiplication, etc.). If the two SIMD clusters have exactly the same computation form then the computational type of the new cluster is SIMD, otherwise, it is MIMD. We denote the computation form of  $S_m^u$  by  $CF(S_m^u)$ . Then the computational type of a new cluster  $S_m^u$  generated from embedding or merging  $n$  clusters,  $S_1^u, S_2^u, \dots, S_n^u$ , can be formulated as follows.

$$\rho S_m^u = \begin{cases} 0 & \text{if } (\rho S_i^u = 0, \text{ for all } i) \text{ and } (CF(S_1^u) = CF(S_2^u) = \dots = CF(S_n^u)) \\ 1 & \text{otherwise} \end{cases} \quad (\text{A.9})$$

Note that since our task graphs are independent of any system graphs (unlike [64, 54, 67]), they do not contain the information about computation time and communication delay. Therefore, we can only embed one node into another as part of clustering for reducing communication overhead. The embedding of multiple nodes onto one node is done as part of the mapping, as explained in the next section.

The time complexity of the CNDG algorithm is bounded by the number of edges in the task graph, which is  $O(|E_t|)$ . For the worst case, we have an upper bound for this algorithm, that is,  $O(M^2)$ , where  $M$  is the number of nodes. However, note that most graphs are not completely connected, therefore, in practice, the time complexity of this algorithm will be  $O(M)$  if the number of edges is proportional to the number of nodes. To illustrate this algorithm, consider the task graph of seven modules and its Spec graph, as shown in Figure A.6. Each module is labeled with its computation amount and each edge is labeled with the amount of data communication. The

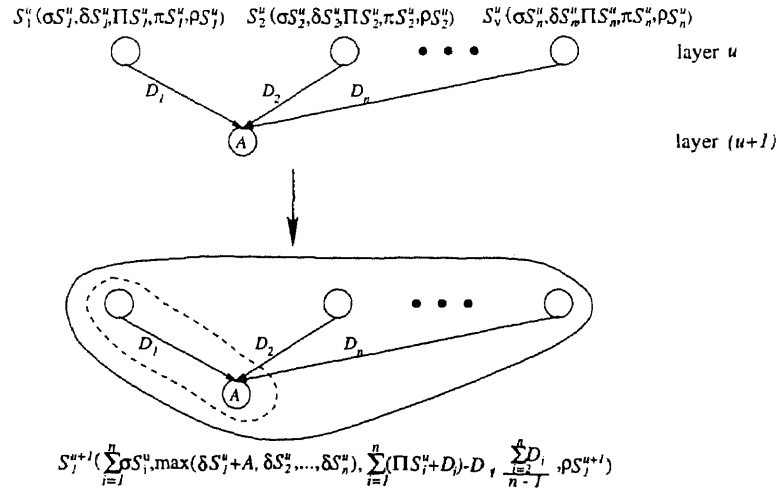


Figure A.4 Clustering on a join-node: a general case

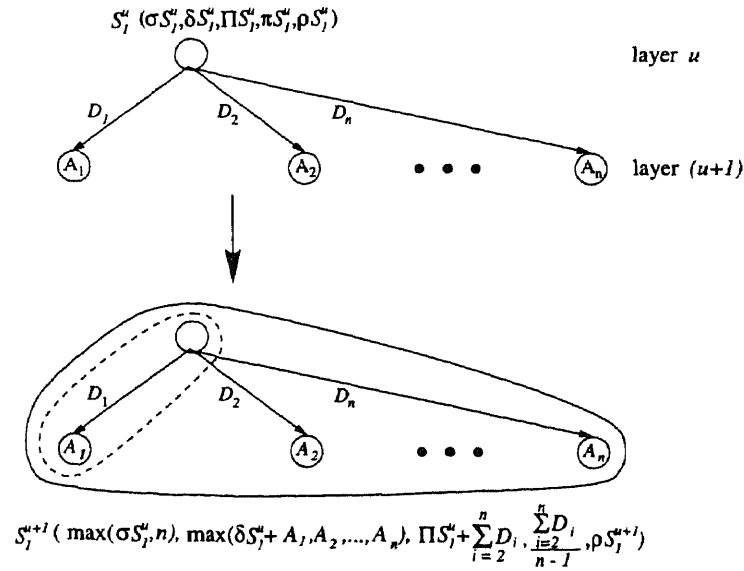
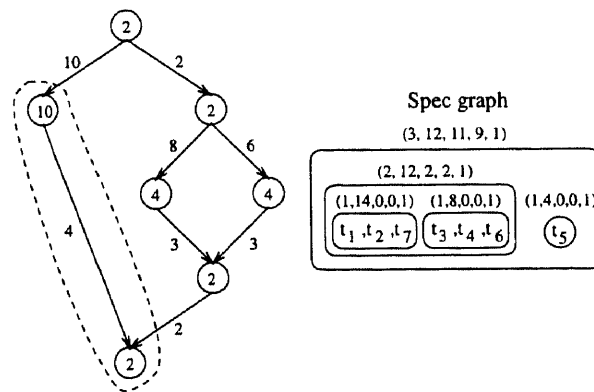
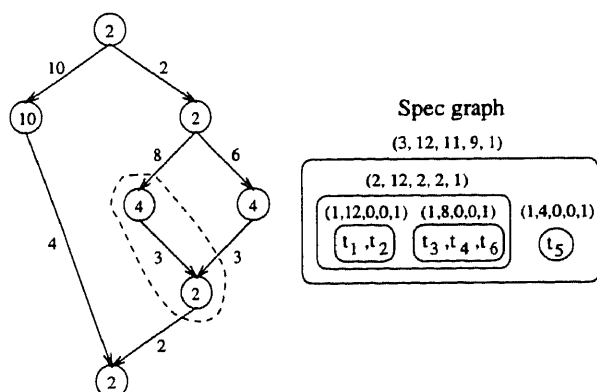
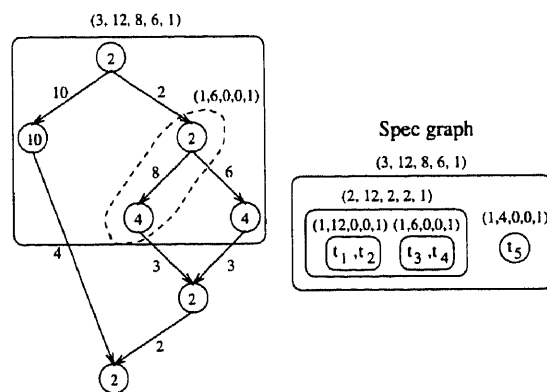
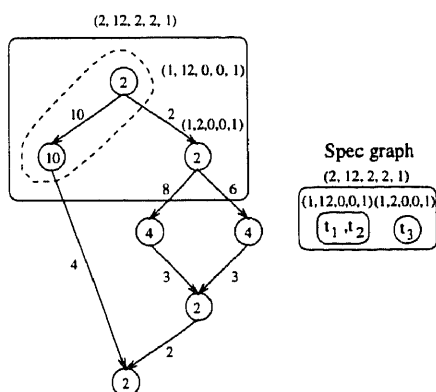
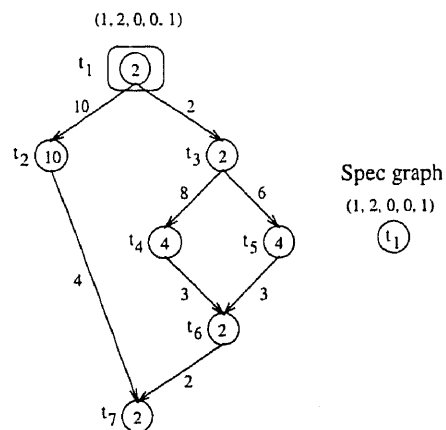


Figure A.5 Clustering on a fork-node: a general case





**Figure A.6** A task graph and steps for obtaining the Spec graph

Spec graph is constructed by embedding/merging the clusters layer by layer and is a multi-layer clustered graph as shown.

### Clustering Undirected System Graphs

A parallel system that can be modeled as an undirected system graph  $G_p(V_p, E_p)$ . In  $G_p$ ,  $V_p = \{p_1, \dots, p_N\}$  is the set of processors forming the underlying architecture, while  $E_p$  is the set of edges representing the interconnection topology of the parallel system. We assume that the connections between adjacent processors are bidirectional. Therefore, an edge  $(p_i, p_j)$  represents that there is a direct connection between processor  $p_i$  and  $p_j$ . The computational speed of processor  $p_i$  is denoted by  $B_i$ , and the communication bandwidth between two processors  $p_i$  and  $p_j$  is denoted by  $C_{ij}$ . The transmission rate is a function of the communication bandwidth between  $p_i$  and  $p_j$  and the node latencies at  $p_i$  and  $p_j$ . Both the computational speeds of different processors and the transmission rates of different communication links may be nonuniform. This makes the Cluster-M approach more general than approaches such as PYRROS, Hypertool, and PARSAs, which assume fully connected uniform systems.

Similar to Spec clusters, the  $n$ th Rep cluster at layer  $v$ ,  $R_n^v$ , is associated with the quintuple  $(\sigma R_n^v, \delta R_n^v, \Pi R_n^v, \pi R_n^v, \rho R_n^v)$  defined as part of the Cluster-M model in the last section. To construct a Rep graph from an undirected system graph, initially, every node with computation speed of  $B_i$  forms a cluster by itself with parameters  $(1, B_i, 0, 0, 1)$ , assuming that these nodes are all MIMD type. Then clusters that are completely connected are merged to form a new cluster, and the parameters of the new cluster are calculated, as explained below. This process is repeated until no further merging is possible. Three clusters  $R_x^v, R_y^v$ , and  $R_z^v$  are completely connected if  $R_x^v$  contains a node  $p_x$ ,  $R_y^v$  contains a node  $p_y$ , and  $R_z^v$  contains a node  $p_z$ , so that nodes  $p_x, p_y$ , and  $p_z$  form a clique. This definition can be extended for  $N$  completely

connected clusters. To calculate the values of the first four parameters for a new cluster, consider a new cluster  $R_n^{v+1}$ , which is generated at layer  $(v+1)$  by merging  $N$  completely connected clusters  $R_1^v, R_2^v, \dots, R_N^v$  at layer  $v$ . Then the values of  $\sigma R_n^{v+1}$  and  $\delta R_n^{v+1}$  can be easily computed as follows.

$$\sigma R_n^{v+1} = \sum_{i=1}^N \sigma R_i^v \quad (\text{A.10})$$

$$\delta R_n^{v+1} = \frac{\sum_{i=1}^N \sigma R_i^v \delta R_i^v}{\sigma R_n^{v+1}} = \frac{\sum_{i=1}^N \sigma R_i^v \delta R_i^v}{\sum_{i=1}^N \sigma R_i^v} \quad (\text{A.11})$$

We denote the transmission rate between  $R_i^v$  and  $R_j^v$  to be  $C_{ij}^v$ , which is defined as the sum of the transmission rate (as a function of communication bandwidth and switching latency) of each pair of processors (subclusters)  $p_i$  and  $p_j$  such that  $p_i$  is in  $R_i^v$  and  $p_j$  is in  $R_j^v$ , that is,  $C_{ij}^v = \sum_{p_i \in R_i^v, p_j \in R_j^v} C_{ij}$ . Then  $\Pi R_n^{v+1}$  and  $\pi R_n^{v+1}$  can be calculated as follows.

$$\Pi R_n^{v+1} = \sum_{i=1}^N \Pi R_i^v + \sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij}^v \quad (\text{A.12})$$

$$\pi R_n^{v+1} = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij}^v}{\frac{N(N-1)}{2}} = \frac{2(\sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij}^v)}{N(N-1)} \quad (\text{A.13})$$

The fifth parameter,  $\rho R_n^{v+1}$ , is computed per (A.9).

The algorithm for clustering undirected graphs is shown in Figure A.7. Instead of using an optimal algorithm for finding cliques, we use a heuristic so that, for every cluster, we examine the set of edges connected to it in the following manner. The edges are sorted in descending order based on the value of  $C_{ij}$ . The edges are then examined one at a time from this list. If more than one of the edges have the same weight, then an arbitrary one is selected. A simple example is shown in Figure A.8.

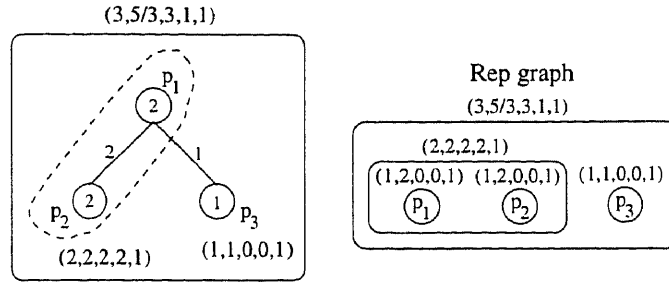
We now analyze the running time of this implementation. For each layer, we first sort all the edges between clusters. This sort takes  $O(|E_p| \log |E_p|)$  time, where  $|E_p|$  is the number of edges in the system graph. Then, we keep merging clusters into the next layers. Suppose at a certain layer, there are  $m$  clusters  $c_1, \dots, c_m$ . The time for finding cliques among these clusters is at most  $m \times m \leq N^2$ , where  $N$  is the

```

Clustering Nonuniform Undirected Graphs (CNUG) Algorithm
For all nodes  $p_i$  do
begin Make a cluster for  $p_i$  at clustering layer 1
    Set the parameters of the cluster to be  $(1, B_i, 0, 0)$ 
end
Set cluster layer  $L$  to be 1
While there is at least one edge linking two clusters of layer  $L$  do
begin Sort all edges linking any two clusters in descending order
    While sorted edge list is not empty, do
begin Take the first edge  $(c_i, c_j)$  from sorted edge list
    Delete the edge from the list
    Merge  $c_i$  and  $c_j$  into cluster  $c'$  at layer  $(L + 1)$ 
    Calculate the parameters of  $c'$ 
    Delete clusters  $c_i$  and  $c_j$  from current layer  $L$ 
    For each edge  $(c_x, c_y)$  in sorted edge list
begin
    If  $(c_x$  is a sub-cluster of  $c')$  and
     $(c_y$  is not a sub-cluster of any cluster) and
     $(c_y$  is connected to all other sub-clusters of  $c')$  then
begin Merge  $c_y$  into  $c'$ 
        Recalculate the parameters of  $c'$ 
        Delete  $(c_x, c_y)$  from edge list
end
    Else if  $c_x$  and  $c_y$  are sub-clusters of two different clusters at layer  $(L + 1)$ , then
begin Add the weight of  $(c_x, c_y)$  to the edge between the two super-clusters
        Delete  $(c_x, c_y)$  from edge list
end
end
end
Increment clustering layer  $L$  by 1
end

```

**Figure A.7** Clustering Nonuniform Undirected Graphs (CNUG) algorithm.



**Figure A.8** A nonuniform system graph and its Rep graph.

number of processors in the system graph. The most number of layers there can be is  $N - 1$ . Therefore the total time complexity of this algorithm is  $O(N(|E_p| \log |E_p| + N^2))$ . Consider the worst case, where the system graph is completely connected (i.e.,  $|E_p| = O(N^2)$ ), then the time complexity of this algorithm will be  $O(N^3 \log N)$ . Note that most system graphs are not completely connected. Therefore, in practice the time complexity of this algorithm will be  $O(N^3)$  if the number of edges is proportional to the number of nodes.

## REFERENCES

1. A. Agrawal and Lena Nekludova. A parallel  $O(\log N)$  algorithm for finding connected components in planar images. In *Proc. IEEE International Conference on Parallel Processing*, 1987.
2. R. Anderson and G. L. Miller. Optimal parallel algorithms for list ranking. Technical report, Dept. of Computer Science, University of Southern California, 1987.
3. R. Barakat and J. Reif. Lower bounds on the computational efficiency of optical computing systems. *Journal of Applied Optics*, 26(6):1015–1018, March 15 1987.
4. Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13:139–153, 1991.
5. V. Benes. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, 1965.
6. L. A. Bergman, W. H. Wu, A. R. Johnston, R. Nixon, C. C. Esener, S. C. Guest, P. Yu, T. J. Drabik, M. Feldman, and S. H. Lee. Holographic optical interconnects for VLSI. *Optical Engineering*, October 1986.
7. K. H. Brenner and T. M. Merklein. Implementation of an optical crossbar network based on directional switches. *Applied Optics*, 31:2446–2451, 1991.
8. A. M. Butrym, N. Craft, D. Guise, M. Murdocca, and F. Sauer. A model for a reconfigurable fine-grained optoelectronic processor. In *Massively Parallel Processing with Optical Interconnections*, 1994.
9. S. Chen and M. Eshaghian. A fast recursive mapping algorithm. *Concurrency: Practice and Experience*, 7(5):391–409, August 1995.
10. S. Chen, M. M. Eshaghian, and Y. Wu. Mapping arbitrary non-uniform task graphs onto arbitrary non-uniform system graphs. In *1995 International Conference on Parallel Processing*, volume II, pages 191–195, August 1995.
11. A. Chiou and P. Yeh. Energy efficiency of optical interconnections using photorefractive holograms. *Applied Optics*, 29:1111–1117, 1990.
12. B. D. Clymer and J. W. Goodman. Optical clock distribution to silicon chips. *Optical Engineering*, October 1986.
13. P. W. Dowd. Wavelength division multiple access channel hypercube processor interconnection. *IEEE Transactions on Computers*, Oct. 1991.

14. M. Eshaghian and M. Shaaban. Cluster-M parallel programming paradigm. *International Journal of High Speed Computing*, 6(2):287-309, June 1994.
15. M. M. Eshaghian. Parallel algorithms for image processing on OMC. *IEEE Transactions on Computers*, 40(7):827-833, July 1991.
16. M. M. Eshaghian and R. Miller. The systolic reconfigurable mesh. In *1995 International Conference on Parallel Processing*, August 1995.
17. M. M. Eshaghian, J. G. Nash, M. E. Shaaban, and D. B. Shu. Heterogeneous algorithms for image understanding architecture. *Parallel Algorithms and Applications*, 1:273-284, 1993.
18. M. M. Eshaghian and V. K. Prasanna-Kumar. A VLSI electro-optical crossbar for signal processing. In *SPIE's 33rd annual International Symposium on Optical and Optoelectronic Applied Science and Engineering*, 1989.
19. T. Y. Feng. A survey of interconnection networks. *IEEE Computer*, December 1981.
20. I. Foster and S. Tuecke. Parallel programming with PCN. Technical report, Argonne National Laboratory, University of Chicago, January 1993.
21. J. A. Fried. Optical I/O for high speed CMOS systems. *Optical Engineering*, October 1986.
22. J. W. Goodman and K. M. Johnson. Optical computing research. In *Final report to AFOSR*, 1981.
23. J. W. Goodman, S. Y. Kung, F. J. Leonberger, and R. A. Athale. Optical interconnections for VLSI systems. In *Proceedings of IEEE*, volume 72, July 1984.
24. R. I. Greenberg and Charles E. Leiserson. Randomized routing on fat-trees. In *Proc. 26th Annual Symposium on Foundations of Computer Science*, pages 241-249, October 1985.
25. T. S. Guan and S. P. V. Barros. Reconfigurable multiple-behavioural architecture using free-space optical communication. In *Massively Parallel Processing with Optical Interconnections*, 1994.
26. D. H. Hartman. Digital high speed interconnects: A study of the optical alternative. *Optical Engineering*, 25(10), October 1986.
27. P. R. Haugen, S. Rychnovsky, A. Hussain, and L. D. Hutcheson. Optical interconnects for high speed computing. *Optical Engineering*, 25(10), October 1986.
28. E. Hecht. *Optics*. Addison-Wesley, Reading, MA, 1987.

29. A. Huang and J. Goodman. Number theoretic processors, optical and electronic. In *Optical Processing Systems, Proc. SPIE 185*, 1979.
30. H. Ito, N. Komagata, H. Yamada, and Humio Inaba. New structure of laser diode and light emitting diode based on coaxial transverse junction. Technical report, Research Institute of Electrical Communication, Tohoku University, Sendai 980, Japan.
31. B. K. Jenkins and C. L. Giles. Superposition in optical computing. In *Proc. ICO International Conference on Optical Computing*, Toulon, France, 1988.
32. S. Kawai. Free-space multistage optical interconnection networks using micro lens arrays. *Journal of Lightwave Technology*, 9(2):1774–1779, Dec. 1991.
33. I. Lee, S. Goldwasser, and D. Smitely. Synthesis and mapping algorithms for a reconfigurable optical interconnection network. In *Proc. IEEE International Conference on Parallel Processing*, 1986.
34. F. T. Leighton. New lower bound techniques for VLSI. *IEEE FOCS*, 1981.
35. Tom Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, U. S. A., 1992.
36. H. Li and M. Maresca. Polymorphic-torus network. *IEEE Trans. on Computers*, 38:1345–1351, Sep. 1989.
37. Y. Li, T. Wang, and J. Sharony. Free-space optical interconnections using connectivity-enhanced, mesh-based networks. *Optical Engineering*, 33(5):1532–1542, May 1994.
38. G. E. Lohman and K. H. Brenner. Space-invariance in optical computing systems. *Optik*, 89:123–134, 1992.
39. A. Louri and H. Sung. 3D optical interconnects for high-speed interchip and interboard communications. *IEEE Computer*, Oct. 1994.
40. E. S. Maniloff, K. M. Johnson, and J. H. Reif. Holographic routing network for parallel processing machines. In *SPIE International Congress on Optical Science and Engineering, Paris, France*, 1989.
41. A. McAulay. Optical crossbar interconnected digital signal processor with basic algorithms. In *SPIE Real Time Signal Processing Conf. VIII*, 1985.
42. R. Miller, V. K. P. Kumar, D. I. Reisis, and Q. F. Stout. Parallel computations on reconfigurable meshes. *IEEE Trans. on Computers*, 42(6):678–692, Jun 1993.



43. R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. F. Stout. Meshes with reconfigurable buses. In *Proc. MIT Conf. Advanced Research in VLSI*, pages 163–178, Mar. 1988.
44. R. Miller and Q. F. Stout. Convexity algorithms for pyramid computers. In *Proc. 7th International Conference on Pattern Recognition*, 1984.
45. V. N. Morozov, J. A. Neff, H. Temkin, and A. S. Fedor. Analysis of a three-dimensional computer optical scheme based on bidirectional free-space optical interconnections. *Optical Engineering*, 34(2):523–534, Feb. 1995.
46. D. Nassimi and S. Sahni. Parallel algorithms to set up the Benes permutation network. *IEEE Transactions on Computers*, C-31(2), February 1982.
47. J. A. Neff. Optical crossbar switch to be developed for strategic computers. In *International Solid State Circuits Conference*, 1985.
48. M. Nigam and S. Sahni. Sorting  $n$  numbers on  $n \times n$  reconfigurable meshes with buses. In *Univ. of Florida Dept. of CIS, Tech. report TR-92-35*, 1992.
49. J. W. Parker. Optical interconnection for advanced processor systems: a review of the ESPRIT II OLIVES program. *Journal of Lightwave Technology*, 9(12):1764–1773, Dec. 1991.
50. V. K. Prasanna-Kumar and M. Mary Eshaghian. Parallel geometric algorithm for digitized pictures on mesh of trees. In *Proc. IEEE International Conference on Parallel Processing*, 1986.
51. A. G. Ranade. How to emulate shared memory. In *Proc. Annual Symposium on Foundations of Computer Science*, 1987.
52. J. H. Reif and A. Yoshida. Optical expanders with applications in optical computing. *Applied Optics*, 32:159–165, 1993.
53. J. H. Reif and A. Yoshida. Free space optical message routing for high performance parallel computers. In *Massively Parallel Processing with Optical Interconnections*, 1994.
54. V. Sarkar. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*. The MIT Press, Boston, MA, 1989.
55. A. A. Sawchuk, B. K. Jenkins, C. S. Raghavendra, and A. Varma. Optical crossbar networks. *IEEE Computer*, June 1987.
56. H. J. Siegel. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*. Lexington Books, Lexington, MA, 1984.
57. T. Szymanski. Hypermeshes: optical interconnection networks for parallel computing. *Journal of Parallel and Distributed Computing*, 26:1–23, 1995.

58. R. K. Thiruchelvan, J. L. Trahan, and R. Vaidyanathan. On the power of segmenting and fusing buses. In *Proc. International Parallel Processing Symposium*, pages 79–83, April. 1993.
59. C. D. Thompson. *A Complexity Theory for VLSI*. PhD thesis, Dept. of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1980.
60. R. Vaidyanathan. Sorting on PRAMs with reconfigurable buses. *Info. Processing Letters*, 42:203–208, 1992.
61. L. G. Valiant and G. J. Brebner. Universal schemes for parallel computations. In *Proc. ACM Symposium on Theory of Computing*, pages 263–277, 1981.
62. T. S. Wailes and D. G. Meyer. Multiple channel architecture: a new optical interconnection strategy for massively parallel computers. *Journal of Lightwave Technology*, Dec 1991.
63. B. F. Wang and G. H. Chen. Constant time algorithms for the transitive closure problem and some related graph problems with reconfigurable bus systems. *IEEE Trans. on Parallel and Distributed Systems*, 1:500–507, April. 1991.
64. M. Y. Wu and D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Trans. on Parallel and Distributed Systems*, 1(3):101–119, 1990.
65. S. Wu, W. Song, A. Mayers, D. A. Gregory, and F. T. S. Yu. Reconfigurable interconnections using photorefractive holograms. *Applied Optics*, 29:1118–1125, 1990.
66. J. C. Wyllie. *The complexity of parallel computations*. PhD thesis, Dept. of Computer Science, Cornell University, Ithaca, NY, 1979.
67. T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. on Parallel and Distributed Systems*, 5(9):951–967, September 1994.
68. P. Yeh, A. E. T. Chiou, and J. Hong. Optical interconnection using photorefractive dynamic holograms. *Applied Optics*, 27:2093–2096, 1988.