UNCLASSIFIED

DTIC FILE COPY

(2)

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION: Unclassified

AD-A223 479

1b. RESTRICTIVE MARKINGS:

2a. SECURITY CLASSIFICATION AUTHORITY: None

3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: None

JUN 27 1990 S D

4. PERFORMING ORGANIZATION REPORT NUMBER(S):

5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 90 0675

6a. NAME OF PERFORMING ORGANIZATION: University of Southern California

6b. OFFICE SYMBOL (If applicable):

7a. NAME OF MONITORING ORGANIZATION: Department of the Air Force, Air Force Office of Scientific Research

6c. ADDRESS (City, State, and ZIP Code): Institute for Robotics & Intelligent Systems, Powell Hall Room 204, Los Angeles, CA 90089-0273

7b. ADDRESS (City, State, and ZIP Code): Bolling Air Force Base, D.C. 20332-6448 (BLDG 410)

8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Air Force Office of Scientific Research

8b. OFFICE SYMBOL (If applicable): NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Grant No. AFOSR-89-0032

8c. ADDRESS (City, State, and ZIP Code): Building 410, Bolling AFB, D.C. 20332-6448

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
61102F	2304	A7	

11. TITLE (Include Security Classification): Parallel Architectures and Algorithms for Image Understanding
Final Technical Report

12. PERSONAL AUTHOR(S): R. Nevatia and V.K. Prasanna-Kumar

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical Report	FROM 11/1/88 TO 9/30/89	1990, May 17	39

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
			Parallel Architectures, Parallel Algorithms, Latin Squares, Image Understanding, Array Access, Processor Time, Optimal Solutions, Image Labelling, Low Level Vision. (KR)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Several issues in using parallel processing for image understanding are addressed. First, efficient schemes for parallel image array access are developed. New class of latin squares called perfect latin squares are defined. Several construction methods are shown and some useful properties of such squares are identified. A generic parallel model of computation employing electro optical devices is developed. Parallel techniques for image computations are developed on this model. Finally, processor time optimal solutions to several low and intermediate level image understanding tasks are designed on orthogonal access parallel architectures.

Keywords:

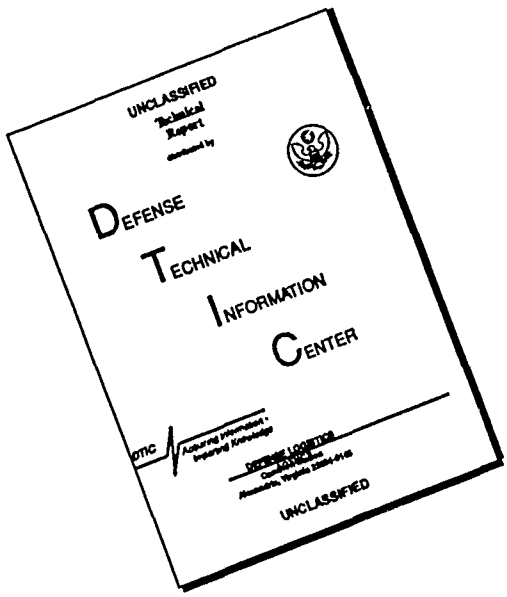
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
Abraham Waksman, Program Manager	(202) 767-5025	NM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE: Unclassified

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

INSTITUTE FOR ROBOTICS AND INTELLIGENT SYSTEMS

UNIVERSITY OF SOUTHERN CALIFORNIA
School of Engineering
Powell Hall Room 204
Los Angeles California 90089-0273
(213) 743-6377

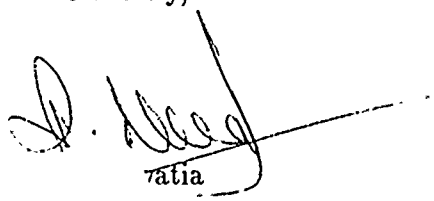
May 29, 1990

Dr. Abraham Waksman
Program Manager
AFOSR/NM
Building 410
Bolling AFB, DC 20332-6448

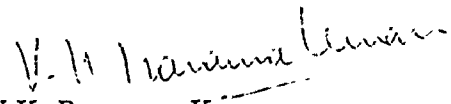
Dear Dr. Waksman:

Please find enclosed six copies of our Final Technical Report for the period of November 1, 1988 to September 30, 1989 on our AFOSR Grant #89-0032.

Sincerely,



Co-Principal Investigator



V.K. Prasanna-Kumar
Co-Principal Investigator

cc: Kathleen Wetherell, Contracting Officer
Harriet Vigoren, USC Contracts and Grants

encl: 6 copies of report

Parallel Architectures and Algorithms for Image Understanding

AFOSR Grant # AFOSR-89-0032

Final Technical Report

R. Nevatia and V.K. Prasanna Kumar
Institute for Robotics and Intelligent Systems
University of Southern California

May 17, 1990

Summary of Results

This report describes the major accomplishments under the grant AFOSR-89-0032. The main focus of research was on two major areas: study of parallel architectures, and design of efficient parallel algorithms for image understanding. In the following, we summarize the research. The appendix includes copies of publications supported under this grant.

1 Memory Systems for Image Processing

Communication between memory and processors always has been a bottleneck in high performance computer systems. Low memory bandwidths can lead to a significant performance degradation of these systems. Particularly, in real time image processing and vision applications, where the amount of data movement is enormous, very fast memory operations are needed. A new efficient memory system has been proposed for image processing and vision. In this system, Latin Squares are adopted as the skewing scheme. A new type of Latin squares, *perfect Latin squares* with special properties are introduced for image processing. A simple construction method for perfect Latin squares is also shown. The resulting memory system provides access to various subsets of image data (rows, columns, diagonals, main subsquares etc.) without memory conflict. The memory modules are fully utilized for most frequently used subsets of image data. The address generation can be performed in constant time. This memory system is the first known system that achieves constant time access to rows, columns, diagonals and subarrays using minimum number of memory modules [5].

2 Electro-optical Architectures

Arrays of processors with optical interconnection networks for fine grain image processing are studied. It has been shown that the unit time interconnection medium made possible with the use of free space optics results in efficient solutions to communication intensive image computations. Using these models, $O(\log N)$ pointer based algorithms for several tasks in low and medium level vision are presented. We also presented a generic solution that can be used to simulate PRAM algorithm for image computation on the proposed electro-optical arrays [4].

3 Orthogonal-access Parallel Organizations

We have investigated a class of *orthogonal-access* parallel organizations for applications in image and vision analysis. These architectures consist of a massive memory and reduced number of processors which access the shared memory. The memory can be envisaged as an array of memory modules in the k-dimensional space, with each row of modules along a certain dimension connected to one bus. Each processor has access to one bus along each dimension. It is shown that these organizations are communication-efficient and can provide processor-time optimal solutions to a wide class of image and vision problems. In the two dimensional case, the basic organization has n processors and $n \times n$ memory arrays which can hold $n \times n$ image, and it provides $O(n)$ time solution to several image computations including: histogram, histogram equalization, computing connected components, convexity problems, and computing distances, etc. [6].

References

[1] H.M. Alnuweiri, and V.K. Prasanna Kumar, *Fast Image Labeling Using Local Operators on Mesh-Connected Computers,* Proceedings of International Conference on Parallel Processing (ICPP), 1989.

[2] H.M. Alnuweiri, *Communication efficient Parallel Architectures and Algorithms for Image Computations,* Ph.D. thesis, August 1989, USC.

[3] M. Mary Eshaghian *Parallel Computing Using Optical Interconnection Networks,* Ph.D. thesis, December 1988, USC.

[4] M. Mary Eshaghian and V.K. Prasanna Kumar, *Fine Grain Image Computations on Electro-optical Arrays,* Proceedings of IEEE conference on Computer Vision and Pattern Recognition (CVPR), 1989.

[5] Kichul Kim and V.K. Prasanna Kumar, *Parallel Memory Systems for Image Processing,* Proceedings of IEEE conference on Computer Vision and Pattern Recognition (CVPR), 1989.

[6] V.K. Prasanna-Kumar, *Parallel Image Computations on Reduced VLSI Arrays,* Indo-US Workshop on high speed signal processing, 1989.

Appendix

Appears in Proceedings of the IEEE Computer Vision and Pattern Recognition Conference, 1989.

Parallel Memory Systems for Image Processing[1]

Kichul Kim and V. K. Prasanna Kumar

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781
(213)-743-5236

Abstract

A novel memory system is proposed for image processing. Latin squares, which are well known combinatorial objects for centuries, are used as the skew function of the memory system. We introduce a new Latin square with desired properties for image array access. The resulting memory system provides access to various subsets of image data(rows, columns, diagonals, main subsquares etc.) without memory conflict. The memory modules are fully utilized for most frequently used subsets of image data. The address generation can be performed in constant time. This memory system is the first known memory system that achieves constant time access to rows, columns, diagonals and subarrays using minimum number of memory modules.

1 Introduction

With recent advances in VLSI and parallel processing, large number of processors can be interconnected to yield high performance systems for image processing and vision. Most image processing tasks not only perform intensive computations but also process enormous amount of data. Low effective bandwidth of the memory system can lead to significant performance degradation. In order to fully utilize the processing power available in parallel systems, special attention should be given to memory organizations as well as mapping of image arrays.

In image processing, an image can be represented by a two dimensional array of size $M \times N$. It is well understood that access to row vectors, column vectors, diagonals and subarrays are required heavily in image processing[10,17]. A memory system for image processing should provide efficient access to rows, columns, diagonals and subarrays. Also, while implementing partitioned VLSI arrays and special purpose arrays, access to various subarrays is needed.

Today's computers use multiple memory modules either in interleaved memory form or in parallel memory form. The effective bandwidth of these memory systems not only depends on the number of modules and the speed of operation but also depends on the occurrence of memory conflicts, which occur when more than one memory request is given to the same memory module during a memory cycle. Hence, the memory system should provide conflict free access to as many subsets of interest as possible.

Many researchers have addressed the issue of designing memory systems which provide efficient access to rows, columns, diagonals or subarrays[1,2,9,11,12,16,17,18,19]. Budnik and Kuck introduced linear skewing scheme and prime memory system which allows conflict free access to rows, columns, diagonals and $N^{1/2} \times N^{1/2}$ subarrays of an $N \times N$ array, in which the number of memory modules is a prime number greater than N[2,9]. However, their method needs modulo operation of a prime number in address generation, which can not be performed in constant time with a reasonable amount of circuitry. Lawrie[11] generalized the linear skewing scheme and proposed an array storage scheme with a simple address generation. His scheme provides conflict free access to rows, columns, diagonals and $N^{1/2} \times N^{1/2}$ subarrays of an $N \times N$ array using $2N$ memory modules. He also introduced the omega network to perform data routing and data alignment. However, his scheme suffers from low utilization of memory modules. Only half of the memory modules are used in a memory fetch. Memory organizations with linear skewing schemes suffer either low memory utilization or difficulty in address generation. Another way of achieving efficient access to various subsets of interest is by using nonlinear skewing schemes[1,3,12,16]. Lee[12] presented a nonlinear skewing scheme(called scrambled storage scheme) and a new network which allows conflict free access to rows, columns, square blocks and distributed blocks. The scrambled storage scheme does not need modulo operations for address generation. However, it can not provide efficient access to diagonals.

Even though a lot of work has been done in the context of array storage schemes for general purpose parallel computers, not much has been done with respect to access patterns needed for image processing and vision. Voorhis and Morrin[17] proposed a linear skewing scheme and an address generation circuitry for image processing in which only one memory module is not used during a memory

[1]This research was supported in part by the National Science Foundation under grant IRI-8710836 and in part by AFOSR under grant AFOSR-89-0032.

fetch. Their scheme provides conflict free access to rows, columns and subarrays. However, their scheme requires modulo operation with a number which is not a power of two. Later, their method was improved by Park[14] using a simpler address generation circuitry.

To achieve conflict free access to various subsets of data, we propose to use Latin squares which are well known combinatorial objects for centuries[4]. We introduce a new Latin square(denoted perfect Latin square) which has properties useful for image processing and vision. We also show a simple construction method for perfect Latin squares. Using perfect Latin square as the skewing scheme, various subsets of image data are accessible without memory conflict while memory modules are fully utilized for most interesting subsets of image data. Furthermore, it is possible to show that the address generation can be done in constant time.

In the next section, we describe memory access requirements for image processing. In section 3, we introduce perfect Latin squares. We also show a construction method for perfect Latin squares. In section 4, we describe how perfect Latin squares can be used for storing and retrieving image data. Section 5 concludes the paper.

2 Preliminaries

A memory module can be characterized by the amount of data it can store and the cycle time which is the minimum time between two consecutive read or write operations. Clearly, the cycle time determines bandwidth of the memory system. To overcome the limitation imposed by cycle time, multiple memory modules are used in today's computers. One way of using multiple memory modules is by organizing them as parallel memory banks. In this memory system, multiple memory banks are accessed at the same time, therefore, the effective memory bandwidth is multiplied by the number of memory banks. This scheme is commonly used in SIMD or MIMD computers. Another way of using multiple memory modules is by organizing them as interleaved memories which are commonly employed in pipelined vector computers. In this memory system, memory access mechanism is divided into several stages and used in a pipelined manner. Cray X-MP uses a 32-way interleaved memory system. Some memory systems employ both parallel memory banks and interleaved memories. In such a system, each parallel memory bank is composed of interleaved memories.

Even if multiple memory modules are used to increase the effective bandwidth of the memory system, memory conflict can greatly reduce the effective bandwidth. For an extreme example, if all data elements to be accessed are stored in the same memory module, the effective bandwidth is same as using only one memory module irrespective of the number of memory modules in the memory system.

Budnik and Kuck[2] showed that, in general, memory conflicts can not be avoided for arbitrary data access using a limited number of memory schemes. However, if a limited number of subsets of data are used frequently, an efficient memory scheme can be provided for access to the subsets of interest resulting in high effective bandwidth.

In image processing, row vectors, column vectors, subarrays and diagonals(or diagonals of subarrays) are heavily used. The importance of row access is obvious considering that image inputs and outputs are usually in row major order(left to right, top to bottom). There are also many algorithms which need row accesses like one dimensional convolution algorithms. Desirability for column access is justified by the commonly used 90 degree rotation of image and by algorithms which need column accesses like the two dimensional FFT algorithms. Subarrays play an important role in two dimensional template matching and other algorithms. Partitioning is an essential step in mapping algorithms to fixed size VLSI arrays[13]. Access to subarrays of image data is needed in such implementations. Diagonals or diagonals of subarrays are also useful in matrix computation which is an indispensable tool in image processing[10].

From the above arguments, it is obvious that a memory system for image processing should provide efficient, hopefully conflict free, access to row vectors, column vectors, subarrays, and diagonals for image processing.

In this paper, we restrict our attention to the cases where the number of memory modules is an even power of two. The desirability of having number of memory modules to be an even power of two is clear in terms of address generation, utilization of address space and interconnection network.

3 Latin Squares for Memory Access

A *Latin square* of order n is an $n \times n$ square composed with symbols from 0 to $n-1$ such that no symbol appears more than once in a row or in a column[4]. The rows are numbered from 0 to $n-1$, top to bottom. The columns are also numbered from 0 to $n-1$, left to right. The square A shown below is an example of Latin square of order 4.

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}$$

A *diagonal Latin square* of order n is a Latin square such that no symbol appears more than once in any of its two main diagonals. A simple construction method is known for diagonal Latin squares of order n when n is a power of two[4]. Gergely showed a general method to construct

diagonal Latin squares of any order $n \geq 5$. The square B shown below is an example of diagonal Latin square of order 4.

$$B = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \\ 1 & 0 & 3 & 2 \end{bmatrix}$$

Two Latin squares C and D of order n are *orthogonal* to each other if the set of ordered pairs $CD = \{(c_{i,j}, d_{i,j}),\ 0 \leq i,j \leq n-1\}$ is equal to the set of all possible ordered pairs $(i,j),\ 0 \leq i,j \leq n-1$. The squares C and D shown below are orthogonal to each other.

$$C = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

A *self-orthogonal Latin square* is a Latin square orthogonal to its transpose. We define a *doubly self-orthogonal Latin square* as a Latin square orthogonal to its transpose and to its antitranspose. Notice that a doubly self-orthogonal Latin square is also a diagonal Latin square. The square B shown above is also an example of doubly self-orthogonal Latin square. We define a *subsquare* $S_{i,j}$ of a Latin square of order n^2 as an $n \times n$ square whose top left cell has the coordinate (i,j). When $i \equiv 0 \bmod n$ and $j \equiv 0 \bmod n$, the subsquare $S_{i,j}$ is called a *main subsquare*. Subsquare $S_{0,1}$ of the square B is $\begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$ and main subsquare $S_{2,0}$ of the square B is $\begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix}$. We define a *perfect Latin square* of order n^2 as a diagonal Latin square of order n^2 such that no symbol appears more than once in any main subsquare. Hence, in a perfect Latin square, no symbol appears more than once in any row, in any column, in any diagonal or in any main subsquare. The square E shown below is a perfect Latin square of order 9.

$$E = \left[\begin{array}{ccc|ccc|ccc} 0 & 3 & 6 & 1 & 4 & 7 & 2 & 5 & 8 \\ 2 & 5 & 8 & 0 & 3 & 6 & 1 & 4 & 7 \\ 1 & 4 & 7 & 2 & 5 & 8 & 0 & 3 & 6 \\ \hline 3 & 6 & 0 & 4 & 7 & 1 & 5 & 8 & 2 \\ 5 & 8 & 2 & 3 & 6 & 0 & 4 & 7 & 1 \\ 4 & 7 & 1 & 5 & 8 & 2 & 3 & 6 & 0 \\ \hline 6 & 0 & 3 & 7 & 1 & 4 & 8 & 2 & 5 \\ 8 & 2 & 5 & 6 & 0 & 3 & 7 & 1 & 4 \\ 7 & 1 & 4 & 8 & 2 & 5 & 6 & 0 & 3 \end{array}\right]$$

In the rest of this section, we present a construction method for perfect Latin squares of order n^2 for the case $n = 2^k,\ k \in \{1,2,3\ldots\}$. The interested reader can refer to [8] for the construction of perfect Latin squares of order n^2 where n is odd or $n = 2^l m^2,\ l \in \{2,3,4,\ldots\}, m$ is odd.

For the construction of perfect Latin squares, we start with the following lemma.

Lemma 3.1 *If there exist two doubly self-orthogonal Latin squares A of order m and B of order n, then there exists a doubly self-orthogonal Latin square C of order mn.*

Proof: From the doubly self-orthogonal Latin squares A and B, we construct a square C of order mn with the following rule:

$$c_{i,j} = n \times a_{\lfloor i/n \rfloor, \lfloor j/n \rfloor} + b_{i \bmod n, j \bmod n},\quad 0 \leq i,j \leq mn-1$$

where $\lfloor i/n \rfloor$ represents the largest integer not exceeding i/n and $i \bmod n$ represents the nonnegative remainder of i/n. The construction of C can be viewed as follows:

1. Construct a square of order mn using m^2 B's. Let $B_{i,j}$ be the square B whose position is $i+1$ from top and $j+1$ from left.
2. Add $n \times a_{i,j}$ to all members of $B_{i,j}$.

An example is shown below for the case $m = n = 2^2$.

$$A = B = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \\ 1 & 0 & 3 & 2 \end{bmatrix}$$

$$C = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 & 10 & 11 & 8 & 9 & 14 & 15 & 12 & 13 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 & 11 & 10 & 9 & 8 & 15 & 14 & 13 & 12 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 & 9 & 8 & 11 & 10 & 13 & 12 & 15 & 14 \\ \hline 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 10 & 11 & 8 & 9 & 14 & 15 & 12 & 13 & 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 11 & 10 & 9 & 8 & 15 & 14 & 13 & 12 & 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \\ 9 & 8 & 11 & 10 & 13 & 12 & 15 & 14 & 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ \hline 12 & 13 & 14 & 15 & 8 & 9 & 10 & 11 & 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 14 & 15 & 12 & 13 & 10 & 11 & 8 & 9 & 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 \\ 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 13 & 12 & 15 & 14 & 9 & 8 & 10 & 11 & 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \\ \hline 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 & 12 & 13 & 14 & 15 & 8 & 9 & 10 & 11 \\ 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 & 14 & 15 & 12 & 13 & & & 8 & 9 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 15 & 14 & 13 & 12 & & & 9 & 8 \\ 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 & 13 & 12 & 15 & 14 & 9 & & 11 & 10 \end{array}\right]$$

From the construction, it is clear that no symbol appears more than once in any row or in any column. Suppose C is not orthogonal to its transpose. There should exist i,j,k and l, $i \neq k$ or $j \neq l$, such that $c_{i,j} = c_{k,l}$ and $c_{j,i} = c_{l,k}$. Since $0 \leq a_{p,q} \leq m-1$, $0 \leq p,q \leq m-1$, and $0 \leq b_{r,s} \leq n-1$, $0 \leq r,s \leq n-1$, from $c_{i,j} = c_{k,l}$, we have

$$a_{\lfloor i/n \rfloor, \lfloor j/n \rfloor} = a_{\lfloor k/n \rfloor, \lfloor l/n \rfloor} \tag{1}$$

$$b_{i \bmod n, j \bmod n} = b_{k \bmod n, l \bmod n} \tag{2}$$

From $c_{j,i} = c_{l,k}$, we have

$$a_{\lfloor j/n \rfloor, \lfloor i/n \rfloor} = a_{\lfloor l/n \rfloor, \lfloor k/n \rfloor} \tag{3}$$

$$b_{j \bmod n, i \bmod n} = b_{l \bmod n, k \bmod n} \tag{4}$$

(1) (4) contradict the assumption that A and B are doubly self orthogonal Latin squares. Hence, C' is orthogonal to its transpose. The same argument can be applied to C' and its antitranspose. Therefore, C' is a doubly self-orthogonal Latin square. □

Now, we are ready for the following lemma.

Lemma 3.2 *For all $k \in \{2,3,4,\ldots\}$, there exists a doubly self-orthogonal Latin square D^k of order 2^k.*

Proof: We use induction on k.

Bases: For $k = 2$ and $k = 3$, the following squares are doubly self-orthogonal Latin squares of order 2^2 and 2^3. D^3 can be found in [1].

$$D^2 = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \\ 1 & 0 & 3 & 2 \end{bmatrix} \qquad D^3 = \begin{bmatrix} 2 & 3 & 0 & 1 & 7 & 6 & 5 & 4 \\ 0 & 1 & 2 & 3 & 5 & 4 & 7 & 6 \\ 6 & 7 & 4 & 5 & 3 & 2 & 1 & 0 \\ 4 & 5 & 6 & 7 & 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 & 6 & 7 & 4 & 5 \\ 1 & 0 & 3 & 2 & 4 & 5 & 6 & 7 \\ 7 & 6 & 5 & 4 & 2 & 3 & 0 & 1 \\ 5 & 4 & 7 & 6 & 0 & 1 & 2 & 3 \end{bmatrix}$$

Hypothesis: There exists a doubly self-orthogonal Latin square D^k of order 2^k.

Induction Step: We construct a doubly self-orthogonal Latin square D^{k+2} of order 2^{k+2}, using D^2 and D^k, whose existence is proven in the previous lemma. □

Notice that D^2 in the above lemma is also a perfect Latin square. Now, we present the main theorem for the construction of perfect Latin squares.

Theorem 3.1 *For all $k \in \{1,2,3,\ldots\}$, there exists a perfect Latin square P^{2k} of order 2^{2k}, which is also a doubly self-orthogonal Latin square.*

Proof: When $k = 1$, $P^2 = D^2$. When $k \geq 2$, from the doubly self-orthogonal Latin square D^k of order 2^k, we construct a Latin square C'^{2k} of order 2^{2k} such that

$$c'^{2k}_{i,j} = 2^k \times d^k_{[i/2^k],[j/2^k]} + d^k_{i \bmod 2^k, j \bmod 2^k}, \quad 0 \leq i,j \leq 2^{2k}-1$$

Notice that the construction method for C'^{2k} is the same as the construction method appeared in lemma 3.1. From C'^{2k}, we construct perfect Latin square P^{2k} of oreder 2^{2k} such that

$$p^{2k}_{i,j} = c'^{2k}_{2^k \times i \bmod 2^k + [i/2^k], j}.$$

Notice that P^{2k} is obtained from C'^{2k} by a row exchange operation such that the first k most significant bits of i are swapped with the k least significant bits of i, i.e., the rows are shuffled k times. As an example, we build a perfect Latin square of order 2^4. Since the square A in lemma 3.1 is the doubly self-orthogonal Latin square D^2, the matrix C' in lemma 3.1 is exactly the intermediate matrix C'^4 we want. We get a perfect Latin square P^4 by the row exchange operation on C'^4.

$$P^4 = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 12 & 13 & 14 & 15 & 8 & 9 & 10 & 11 & 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 & 12 & 13 & 14 & 15 & 8 & 9 & 10 & 11 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 & 10 & 11 & 8 & 9 & 14 & 15 & 12 & 13 \\ 10 & 11 & 8 & 9 & 14 & 15 & 12 & 13 & 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 14 & 15 & 12 & 13 & 10 & 11 & 8 & 9 & 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 \\ 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 & 14 & 15 & 12 & 13 & 10 & 11 & 8 & 9 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 & 11 & 10 & 9 & 8 & 15 & 14 & 13 & 12 \\ 11 & 10 & 9 & 8 & 15 & 14 & 13 & 12 & 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \\ 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 \\ \hline 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 & 9 & 8 & 11 & 10 & 13 & 12 & 15 & 14 \\ 9 & 8 & 11 & 10 & 13 & 12 & 15 & 14 & 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ 13 & 12 & 15 & 14 & 9 & 8 & 10 & 11 & 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \\ 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 & 13 & 12 & 15 & 14 & 9 & 8 & 11 & 10 \end{array}\right]$$

From the construction, it is clear that, in the square P^{2k}, no symbol appears more than once in any row, in any column or in any main subsquare. Suppose P^{2k} is not orthogonal to its transpose. There should be i,j,m and n, $i \neq m$ or $j \neq n$, such that $p^{2k}_{i,j} = p^{2k}_{m,n}$ and $p^{2k}_{j,i} = p^{2k}_{n,m}$. From the construction, $p^{2k}_{i,j}$, $0 \leq i,j \leq 2^{2k}-1$ can be expressed as

$$p^{2k}_{i,j} = 2^k \times d^k_{i \bmod 2^k, [j/2^k]} + d^k_{[i/2^k], j \bmod 2^k}$$

Since $0 \leq d^k_{i,j} \leq 2^k - 1$, $0 \leq i,j \leq 2^k - 1$, from $p^{2k}_{i,j} = p^{2k}_{m,n}$, we have

$$d^k_{[i/2^k], j \bmod 2^k} = d^k_{[m/2^k], n \bmod 2^k} \quad (5)$$

$$d^k_{i \bmod 2^k, [j/2^k]} = d^k_{m \bmod 2^k, [n/2^k]} \quad (6)$$

From $p^{2k}_{j,i} = p^{2k}_{n,m}$, we have

$$d^k_{[j/2^k], i \bmod 2^k} = d^k_{[n/2^k], m \bmod 2^k} \quad (7)$$

$$d^k_{j \bmod 2^k, [i/2^k]} = d^k_{n \bmod 2^k, [m/2^k]} \quad (8)$$

(5)-(8) contradict the assumption that D^k is a doubly self-orthogonal Latin square. Hence P^{2k} is orthogonal to its transpose. The same argument can be applied to P^{2k} and its antitranspose. Hence, P^{2k} is a doubly self-orthogonal Latin square and no symbol appears more than once in any of its two main diagonals. Hence, P^{2k} is a perfect Latin square of order 2^{2k}. □

4 Parallel Access to Image Array

In this section, we show in detail how to use perfect Latin squares in storing and retrieving image data. We also show the properties of perfect Latin squares built from our construction method which are useful for image data storage.

Suppose we have two dimensional image data $I(M,N)$, where M and N are multiples of 2^{2k} and we also have 2^{2k} memory modules. Then, we can use the perfect Latin square P^{2k} as the skewing scheme. The image element $I(i,j)$ will be stored in the memory module $p^{2k}_{i \bmod 2^{2k}, j \bmod 2^{2k}}$. Figure 1 shows an example for the case $M = 12, N = 16$ and $k = 1$, i.e., we are using 4 memory modules to store image data of size 12×16.

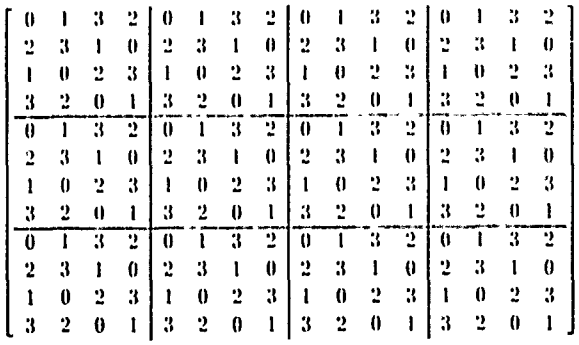


Figure 1: Memory assignment of 12×16 array into 4 memory modules.

In figure 1, it is easy to see that every 1×4 rows and every 4×1 columns are accessible without memory conflict. 2×2 main subarrays are also accessible without memory conflict. Data on the diagonals of 4×4 perfect Latin squares are also accessible without memory conflict. These properties come directly from the definition of perfect Latin squares. However, figure 1 does not fully demonstrate the properties of the perfect Latin squares built from our construction method, because we are using, as the skewing scheme, P^2 which is not built from our method. The perfect Latin squares built from our construction method ($P^{2^k}, k \geq 2$) have other properties which are very useful in image processing. It is easy to verify the following property satisfied by perfect Latin square of order 2^{2^k}, $k \geq 2$, built from theorem 3.1.

Lemma 4.1 *No symbol appears more than once in a subsquare $S_{i,j}$ such that $i \equiv 0 \bmod 2^k$ or $j \equiv 0 \bmod 2^k$.*

The above lemma shows that not only the main subsquares are accessible without memory conflict but also every subsquares on the vertical or horizontal strips of width 2^k are accessible without memory conflict.

It is easy to see that there does not exist a Latin square such that no symbol appears more than once in any subsquare, which means there is no skewing scheme which guarantees conflict free access to rows, columns and $N^{1/2} \times N^{1/2}$ subarrays when N memory modules are used to store an $N \times N$ array. However, we have the following lemma which assures that the maximum degree of memory conflict is two when an arbitrary subsquare is accessed.

Lemma 4.2 *No symbol appears more than two times in any subsquare $S_{i,j}$.*

Lemma 4.2 is a direct result of lemma 4.1 because every subsquare can be partitioned into two parts such that each of them belong to a subsquare on a strip of width 2^k.

Another subset of possible interest in image processing is the subset of elements whose coordinates are the same within each main subsquare. This subset can be used in algorithms which need a value within each main subarray. Let a *same position* $SP_{i,j}$ of a perfect Latin square A of order n^2 be defined as follows,

$$SP_{i,j} = \{a_{k,l} \mid k \equiv i \bmod n, l \equiv j \bmod n\}, \quad 0 \leq i, j \leq n-1.$$

Now, we have the following lemma which shows no memory conflict occurs when a same position is accessed.

Lemma 4.3 *No symbol appears more than once in any same position $SP_{i,j}$.*

5 Conclusion

A new efficient memory system for image processing and vision is described in this paper. Latin squares have been adapted as the skewing scheme. Perfect Latin squares which are Latin squares with special properties useful for image processing have been introduced. A simple construction method for perfect Latin squares is also shown. Using perfect Latin squares as the skewing scheme, many interesting subsets(rows, columns, diagonals, main subarrays etc.) of image data can be accessed without memory conflict.

In this memory system, it is possible to show that the address generation is very easy for both memory module address and local address within memory modules. Memory module address can be generated in constant time with a simple logic circuitry and local address generation does not need any additional hardware.

The efficient skewing scheme along with the fast address generation circuitry makes it possible that rows, columns, diagonals, same positions, and subarrays can be accessed in constant time. This memory system achieves this goal using minimum number of memory modules.

Table 1 shows several known memory schemes along with the scheme proposed in this paper. Comparisons are based on storing $N \times N$ data. An address generation is considered to be hard if it requires modulo operations of a number which is not a power of two.

Results in	# of modules	subsets supported	skew function	address generation
[2]	a prime, $P > N$	row, column, diagonal, subarray	linear	hard
[11]	$2N$	row, column, diagonal, subarray	linear	easy
[17]	$N + 1$	row, column subarray	linear	hard
[12]	N	row, column, *blocks*	nonlinear	easy
This paper	N	row, column, diagonal, subarray same position	nonlinear	easy

Table 1: Comparison of memory systems (Blocks include square blocks and distributed blocks. Square blocks are same as main subsquares and distributed blocks are same as same positions discussed in this paper.)

References

[1] M. Balakrishnan, R. Jain and C. S. Raghavendra, "On Array Storage For Conflict-Free Memory Access For Parallel Processors," *Proc. Intl. Conf Parallel Processing*, vol. 1, pp. 103-107, 1988.

[2] P. Budnik and D. J. Kuck, "The Organization and Use of Parallel Memories," *IEEE Trans. Comput.*, vol. C-20, pp. 1566-1569, 1971.

[3] A. Deb, "Conflict-free Access of Arrays-A counter Example," *Inf. Proc. Letters*, vol. 10, No. 1, pp. 20, 1980.

[4] J. Denes and A. D. Keedwell, *Latin Squares and Their Applications*, Academic Press, New York, 1974.

[5] E. Gergely, "A Simple Method for Constructing Doubly Diagonalized Latin Squares," *J. Combinatorial Theory*, A 16, pp. 266-272, 1974.

[6] A. Hedayat, "A Complete Solution to the Existence and Nonexistence of Knut Vik Designs and Orthogonal Knut Vik Designs," *J. Combinatorial Theory*, A 22, pp. 331-337, 1977.

[7] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

[8] K. Kim and V. K. Prasanna Kumar, "Perfect Latin Squares and Parallel Array Access," Technical Report IRIS #239, University of Southern California, 1988.

[9] D. J. Kuck, "ILLIAC IV Software and Application Programming," *IEEE Trans. Comput.*, vol. C-17, pp. 758-770, 1968.

[10] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988.

[11] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145-1155, 1975.

[12] D. Lee, "Scrambled Storage for Parallel Memory Systems," *Intl. Symp. Computer Architecture*, pp. 232-239, 1988.

[13] J. N. Navarro, J. M. Llaberia and M. Valero, "Partitioning: an essential step in mapping algorithms into systolic array processors," *IEEE Computer*, pp. 77-89, July 1987.

[14] J. W. Park, "An Efficient Memory System for Image Processing," *IEEE Trans. Comput.*, vol. C-35, pp. 669-674, 1986.

[15] A. Rosenfeld and A. Kak, *Digital Image Processing*, Academic Press, 1976.

[16] H. D. Shapiro, "Theoretical Limitations on the Efficient Use of Parallel Memories," *IEEE Trans. Comput.*, vol. C-27, pp. 421-428, 1978.

[17] D. C. V. Voorhis and T. H. Morrin, "Memory Systems for Image Processing," *IEEE Trans. Comput.*, vol. C-27, pp. 113-125, 1978.

[18] H. A. G. Wijshoff and J. V. Leeuwen, "The Structure of Periodic Storage Schemes for Parallel Memories," *IEEE Trans. Comput.*, vol. C-34, pp. 501-505, 1985.

[19] H. A. G. Wijshoff and J. V. Leeuwen, "On Linear Skewing Schemes and d-Ordered Vectors," *IEEE Trans. Comput.*, vol. C-36, pp. 233-239, 1987.

Appears in Proceedings of the IEEE
Computer Vision and Pattern Recognition
Conference, 1989.

Fine Grain Image Computations on Electro-optical Arrays*

M. Mary Eshaghian and V. K. Prasanna-Kumar
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781

Abstract

In this paper, we consider a set of array processors with optical interconnection networks for fine grain image processing. The unit time interconnection medium made possible with the use of free space optics, results in efficient solutions to communication intensive image computations. Using these models, we present a summary of our results in designing $O(\log N)$ pointer based algorithms for several tasks in low and medium level vision. We also show a generic subroutine that can be used to simulate PRAM algorithms for image computations on the proposed electro-optical arrays.

1 Introduction

During past decade, several mesh based VLSI architectures have been considered for fine grain image computations [13, 12, 1, 14]. The underlying interconnection medium of these designs has a significant impact on the performance of the desired parallel algorithms. For example, solving many image processing problems on a $N \times N$ array of processors takes as much as $O(N)$ time [12]. This can be reduced to a logarithmic time complexity for a limited class of problems when organizations such as pyramids, mesh of trees, or reconfigurable meshes is used [13, 14, 11].

The Parallel Random Access Machine (PRAM) is an abstract shared memory model which has been employed in designing parallel algorithms for computational geometry and computer vision, among others. The major drawback of this model is in its unrealistic unit time communication assumption. The unit time simulation of a N processor PRAM using electronic interconnects is not possible unless one allows unbounded fan-in and or fan-out for each of the processors. In fact, a lower bound on the time to simulate one step of a PRAM on any bounded degree network of N nodes using electrical interconnects is $\Omega(\log N)$. Even though several networks have been designed to efficiently simulate PRAM [7], many cost/performance issues limit their potential application in high performance parallel systems.

Recently, there has been an emerging interest in using optical communications in parallel processing. The replacement of the electrical interconnects with optical beams has a significant impact on the performance of parallel computing systems implemented using VLSI technology [4, 10]. This is due to the following two important properties of free space optics. First, free space optical beams can cross each other without any interference. Also, the connections need not be fixed and can be redirected [3]. This implies that using optical interconnects, one can design area efficient bounded degree VLSI architectures that can simulate a unit delay interconnection network.

In this paper, we consider a class of parallel architectures which use free space optical beams as means of interprocessor communications. The organization of these networks is based on a proposed generic parallel model of computation. This model called OMC, allows unit delay interconnects and can efficiently simulate PRAM algorithms. We introduce possible physical realizations of this model. Each of the proposed designs reflects the capabilities and limitations of the device technology used for the redirection of optical beams. Using these models, we present efficient pointer based algorithms for finding geometric properties of digitized images, and parallel implementation of iterative methods used in high level vision processing. We will also present a simple $O(\log N)$ algorithm for simulating each step of an N processor PRAM on these models using $O(N/\log N)$ pro-

*This research was supported in part by the National Science Foundation under grant IRI-8719836 and by AFOSR under grant AFOSR-89-0032

cessors. This can be used as a generic subroutine in translating PRAM algorithms for image computations.

The rest of the paper is organized as follows. In the next section, we discuss an optical model of computation, and present three physical implementations of the model. In section 3, we study a generic subroutine for simulating PRAM algorithms. A set of parallel algorithms for image computations using electro-optical arrays is shown in section 4.

2 Electro Optical Arrays

In the first part of this section, we define an Optical Model of Computation (OMC) which is an abstraction of currently implementable optical and electro-optical computers. Similar to the VLSI model of computation [18], this generic model can be used to understand the limits on computational efficiency in using optical technology. In the second part of this section, we present possible physical realizations of the optical interconnection network defined by OMC. Each of the proposed designs reflects the capabilities and limitations of the device technology used for the redirection of optical beams.

2.1 A Generic Model

OMC is defined as follows:

Definition 1 *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, at that time 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 returned to successful writers. We assume that the operation mode is synchronous, and all processors are connected to a central control unit. The above definition is supplemented with a set of assumptions for an accurate analysis in [6].

Using those assumptions, the following space time trade-off can be shown:

$$AT = \Omega(I)$$

where A is the area occupied by the processing layer, and the time (T) required to solve a problem, is the number of cycles necessary to exchange the minimum required information (I). When compared with three dimensional VLSI model of computation the following proposition 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 $Nd \leq v$.*

In the following section three different parallel architectures are presented as possible efficient upper bounds for v.

2.2 Physical realizations

In this section, we present a class of optical interconnection networks as a realization of the OMC presented in the previous section. Each of the proposed 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.

2.2.1 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. These layers are aligned so that each of the mirrors is located directly above its associated processor. 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, 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-seconds reconfiguration time. Therefore this architecture is suitable for applications where the interconnection topology does not have to be changed frequently.

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. With current technological advancements, the above assumption may not be the most accurate one but we do not see any technological limitations preventing us from making our assumptions. In fact, our assumptions are as valid as those in VLSI; the constant propagation delay assumption regardless of wire's 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 ($O(N)$).

2.2.2 Electro-optical Linear Array

In this organization, N processors are arranged to form a one-dimensional processing layer and the corresponding acousto-optics devices are similarly located on a one-dimensional deflecting layer. 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 is made 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 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.

2.2.3 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. 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 nano-seconds using Gallium Arsenide (GaAs) technology [9].

This architecture is faster than the previous designs and further it compares well with the clock cycle of the current supercomputers. One of the advantages of this simple design is in its implementability in VLSI, using GaAs technology. Unlike the previous designs, this can be fabricated with very low cost and is highly suitable for applications where full connectivity is required. In such applications, the processor layer area can be fully utilized by placing N optical beam receivers in each of the vacant areas to simultaneously interconnect with all the other processors. This design can be easily adopted to implement a neural network of processes with optical interconnects [6].

3 Mapping and Simulation of Parallel Algorithms

An OMC with N processors can simulate, in real time, an Exclusive Read Exclusive Write (EREW) PRAM having P processors and M memory locations, where N= maximum $\{P, M\}$. On the other hand, a P processor EREW PRAM can simulate in real time the computations of a P processor OMC. Thus, the interesting cases are when the memory is "large". In this section, we present a simple efficient algorithm for simulation of a N processor EREW PRAM using an OMC with $N/\log N$ processors. This algorithm can be used as a subroutine for simulating and mapping PRAM algorithms for image processing onto OMC.

The input is assumed to be of the following form. Every processor is responsible for routing the $O(\log N)$ messages that reside in its local memory. Each message has a destination tag attached to it. The destination address is made up of two components x, y, where x denotes the address of the memory module, $x \leq N/\log N$, and y denotes its position within the module, $y \leq \log N$.

In the first step of the algorithm, all elements in each PE are sorted based upon the position to which they will write within the memory module, with duplicate positions being sent to a second queue. Next, each of the elements is sent out one by one. After this, the duplicates from the last iteration are brought forward and the process is repeated. This algorithm works in $O(\log^2 N)$ time by running through each $O(\log N)$ time iteration a total of $\log N$ times, to insure that all elements are transferred to the correct memory locations. This leads to:

Lemma 1 *One step of an N processor EREW PRAM can be simulated on an OMC with $N/\log N$ processors in $\Theta(\log^2 N)$ time. Further, there exists an input sequence for which this is the best possible bound.*

The $O(\log N)$ stages of a slightly modified version of the algorithm in the proof of Lemma 1 [6], leads to a randomized algorithm with a running time of $O(\log N \log\log N)$ with high probability, and a worst case time complexity of $O(\log^2 N)$. Our routing algorithm differs from others in the literature in the way randomization is used. Unlike the algorithms of [20] and [19], it does not randomize with respect to paths taken by messages. For example, Valiant's classic scheme for routing on a hypercube sends each message to a randomly chosen intermediate destination and, from there, to its true destination. On OMC such a technique would lead to queues of size $O(\log^2 N)$. In [8], instead of choosing random paths for messages to traverse, their algorithm repeatedly attempts to deliver a randomly chosen subset of the messages. A by-product of their strategy is that their algorithm requires no intermediate buffering of messages, and hence works under the general situation where each processor can send and receive polynomially many messages. In [15], the only use of randomization is in selecting a hash function to distribute the shared address space of the PRAM onto the nodes of the butterfly. (Note that the direct application of the techniques of [15] to our problem will lead to $O(\log^2 N)$ running time, and $O(\log^2 N)$ local memory requirement for each processor.) Similarly, we only use randomization in assuming a random distribution for input data. This random distribution can also be obtained in selecting a hash function to distribute the shared address space of the PRAM onto an OMC with $N/\log N$ processors, each having $\log N$ local memory. The routing itself is deterministic as explained in the proof of Lemma 1, and has worst case running time of $O(\log^2 N)$.

Theorem 1 *The probability that the simulation of one step of an N processor EREW PRAM using an OMC with $N/\log N$ processors would take more than $O(\log N \log\log N)$ time is given by $\beta e^{-\log^2 N\beta}$, for some constant β independent of N.*

In the above, as opposed to the original algorithm, we check each of the iteration for the number of elements that are left to be sent. If there are none, the algorithm exits else the program will run again, but will only iterate according to the number of elements left to be sent. This leads to $O(\log N \log\log N)$ running time with a high probability. For more details see [6].

4 Fine Grain Image Computing

The algorithms described in the last section, can be used as a subroutine in mapping and simulating any PRAM algorithm for image processing onto the proposed models. In this section, we present a summary of our results in designing algorithms directly on these models for fine grain image computing. For full details see [6].

4.1 Low and Medium Level Vision Processing

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 to which figure (label) each "1" belongs to.

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

Convexity plays an important role in image processing and in vision; many other problems can be solved once the convex hull of figures is obtained. It has applications to normalizing pattern in image processing, obtaining triangulations of sets of points, topological feature extraction, shape decomposition in pattern recognition, testing for linear separability, etc. [13].

Lemma 3 *Given a $N \times N$ 0/1 image, the extreme points of all the figures can be enumerated in $O(\log N)$ time using an $(N/\log^{1/2} N \times N/\log^{1/2} N)$-OMC.*

Another interesting problem is to identify and to compute the distance to a 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.

Lemma 4 *Given a $N \times N$ 0/1 image, the nearest figure to all figures can be computed in $O(\log N)$ time using an $(N/\log^{1/2} N \times N/\log^{1/2} N)$-OMC.*

Template matching is another basic operation in image processing and computer vision [16, 17]. Template matching can be described as comparing a template (window) with all possible windows of the image. Each position of the image will store the result of the window operation for which it is the top-left corner. Note that the computation can be done in $O(N \times k^2)$ time using a uniprocessor. Using $N/\log N$ processors, the above computation can be done in $O(k^2 \log N)$ time.

4.2 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 is possible with OMC.

Any iterative matrix structure can be realized by OMC using devices such as holograms (or those described previously). Although the reconfiguration time for the holograms can be in the order of seconds it only has to be set once during the preprocessing phase. The structure of the coefficient matrix is used to define the holographic connections. The interconnection pattern remains the same throughout the computation. An optimal $O(\log m)$ time can be achieved by this design and the number of processors depends only on the number of non-zero elements in the matrix [6]. This method is attractive when many computations are to be performed in which the structure of the coefficient matrix is fixed. It is well suited for implementation of many iterative methods such as Gauss-Jordan, Gauss-Siedel and the Conjugate method [5].

Consider the iterative method

$$x^{k+1} = Mx^k + g$$

where M is sparse and nonsingular. Let n_i be the number of nonzero element in the ith row of M, and let $j_1, j_2, \ldots, j_{n_i}$ be the columns corresponding to these elements. Thus, the above equation can be rewritten as

$$x_i^{k+1} = \sum_{s=1}^{n_i} m_{ij_s} x_{j_s}^k + g_i.$$

Suppose there are n processors and each of the processors stores exactly one nonzero element in matrix M. Then the following can be shown:

Lemma 5 *The OMC can be used to solve each iteration of the iterative solution to any general sparse linear systems with m variables and n nonzero elements in $O(\log m)$ time.*

5 Conclusion

We studied a set of electro-optical arrays for efficient image computations. Using these, we showed $O(\log N)$ algorithms for determining several properties of images. Furthermore, we introduced a generic subroutine that can be used to map and simulate the existing PRAM algorithms on these models.

References

[1] H. Alnuweiri and V. K. Prasanna-Kumar. Optimal image computations on VLSI architectures with reduced hardware. In *Proc. of IEEE Workshop on Computer Architecture for Pattern and Machine Intelligence*, 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] 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.

[4] B. D. Clymer and J. W. Goodman. Optical clock distribution to silicon chips. *Optical Engineering*, October 1986.

[5] D.J. Evans, Ed. *Sparsity and its applications.* Cambridge University Press, Cambridge, London, 1985.

[6] M. M. Eshaghian. *Parallel Computing with Optical Interconnects.* PhD thesis, Dept. of Computer Engineering, University of Southern California, 1988.

[7] T. Y. Feng. A survey of interconnection networks. *IEEE Computer magazine*, December 1981.

[8] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In *Proc. of the 26th Annual Symposium on Foundations of Computer Science*, pages 241–249, October 1985.

[9] H. Y. H. Ito, N. Komagata and H. 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, 1988.

[10] M. K. Kilcoyne, S. Beccue, K. D. Pedrotti, R. Asatourian, and R. Anderson. Optoelectronic integrated circuits for high speed signal processing. *Optical Engineering*, October 1986.

[11] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout. Data movement operations and applications on reconfigurable VLSI arrays. In *Proc. of IEEE International Conference on Parallel Processing*, August 1988.

[12] R. Miller and Q. F. Stout. Parallel geometric algorithms for digitized pictures on mesh connected computers. In *IEEE transactions on Pattern Analysis and Machine Intelligence*, March 1985.

[13] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. In *IEEE transactions on Computers*, December 1988.

[14] V. K. Prasanna-Kumar and M. M. Eshaghian. Parallel geometric algorithm for digitized pictures on mesh of trees. In *Proc. of IEEE International Conference on Parallel Processing*, 1986.

[15] A. G. Ranade. How to emulate shared memory. In *Proc. of Annual Symposium on Foundations of Computer Science*, 1987.

[16] A. Rosenfeld and A. Kak. *Digital Picture Processing.* Academic Press, 2nd edition, 1982. 2 Volumes.

[17] L. J. Siegel, H. J. Siegel, and A. Feather. Parallel processing approaches to image correlation. *IEEE Trans. on Computers*, C–31, March 1982.

[18] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Dept. of Computer Science, Carnegie Mellon University, 1980.

[19] E. Upfal. Efficient schemes for parallel communication. *JACM*, 31(3):507–517, July 1984.

[20] L. G. Valiant and G. J. Brebner. Universal schemes for parallel computations. In *Proc. of ACM Symposium on Theory of Computing*, pages 263–277, 1981.

To appear in International
Conference on Pattern Recognition
(ICPR), June 1990.

Optimal Image Algorithms on an Orthogonally-Connected Memory Architecture [1]

Hussein M. Alnuweiri and V. K. Prasanna Kumar
SAL-344, Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-0781

September 25, 1989

Abstract

We present processor-time optimal parallel algorithms for several image and vision problems on a novel architecture which combines an orthogonally accessed memory with linear array structure. The organization has p processors and a memory of size $O(n^2)$ locations. The number of processors p can vary over the range $[1, n^{3/2}]$ while providing optimal speedup for several problems in image processing and vision. Such problems include labeling connected components and computing the convex hull, diameter, smallest enclosing rectangle, and nearest neighbors of each region. Histogramming and computing the Hough Transform are also considered. Such problems arise in medium-level vision and require global operations on image pixels. For these problems, it is shown that the proposed organization is superior to the mesh and pyramid organizations.

[1] This research was supported in part by the National Science Foundation under grant IRI-8710836 and in part by AFOSR under grant AFSOR-89-0032.

Contents

List of Figures

List of Tables

1 Introduction

Several parallel techniques for image computations have been implemented on distributed models of computation. In particular, mesh-connected computers have been widely used for image processing since the nearest-neighbor interconnection among the processors preserves the spatial relation among image pixels while maintaining a low hardware implementation cost [7, 11, 31]. Mesh-connected arrays can efficiently handle low level-image processing applications in which only local operations are performed on image pixels [6], or computationally intensive tasks (such as image template matching) for which suitable mappings can result in optimal speedup on the array. However, a wide class of problems in vision and image analysis involves the computation of geometric properties of image regions. Such problems require global or dense data movement operations on the image. The time needed to solve such problems on an $n \times n$ mesh is lower bounded by n (proportional to the diameter of the mesh) even if the problem considered is not computationally intensive. For example, a wide class of problems on $n \times n$ images can be solved sequentially in $O(n^2)$ time, but take $O(n)$ time on an $n \times n$ mesh (i.e. the speedup is not optimal). The mesh-diameter problem can be alleviated by using broadcast buses [24], a reconfigurable bus [15], or a hierarchy of processors and buses such as in the Pyramid Computer (PC) [10, 19], and the Mesh of Trees (MOT) organization [14, 22, 23]. However, even with such enhancements, the number of processors used in such organizations is usually excessive for practical implementations. Furthermore, algorithms which require dense data movement requirements (such as image rotation and sorting pixels) can not be performed efficiently on these organizations.

In this paper, we present a mesh-connected array with special memory-access and communication capabilities to provide processor-time optimal algorithms to several image problems which involve global computations on image pixels. The problems we consider include histogramming, Hough Transform, component labeling, computing convexity and related properties, and computing nearest neighbors. Although such problems are not computationally intensive (except for computing the Hough Transform, they all can be solved sequentially in $O(n^2)$ time for an $n \times n$ image), they incur a high communication overhead when implemented on a parallel distributed-memory computer. However, for such problems, divide-and-conquer and data-reduction techniques can be used to reduce the number of computations and the communication overhead.

The proposed array consists of a memory array of size $O(n^2)$ locations (onto which an $n \times n$ image can be mapped) and p Processing Elements (PEs), where p can vary over the range 1 to n^2. The PEs are interconnected in a mesh fashion and have special access to the memory. The organization is called Mesh-Connected Modules (MCM) because it can be looked upon as a mesh of *basic-modules*, where each basic-module is a parallel organization in which the PEs have row and column access to a subarray of the memory. The MCM supports global data operations efficiently and is thus, highly suited to the class of image computations considered here. One major distinguishing feature of the MCM from other mesh-based parallel organizations is that communication among processors is done via a shared memory as well as by interprocessor links. Most of our algorithms take $O(n^2/p)$ time, using p processors, where p is in the range $[1, n^{3/2}]$. As compared to the performance of a pyramid computer for such problems [19], the MCM performance is clearly superior. For example, the problems of computing the connected components of an image and computing all closest figures take $O(n^{1/2})$ time, while computing the convexity of multiple figures takes $O(n^{1/2} \log n)$ time, on a pyramid computer with $O(n^2)$ PEs. The MCM can solve all these problems in $O(n^{1/2})$ time using $n^{3/2}$ PEs only.

The rest of the paper is organized as follows. Section 2 introduces the proposed MCM organization; section 3 presents a summary of results and comparisons to other organizations; and section 4 presents processor-time optimal parallel algorithms for several image problems.

2 Architecture

The proposed organization is based on a class of *orthogonal memory-access* architectures which have been studied by several authors [1, 5, 12, 27, 29]. A VLSI-based multiprocessor architecture with orthogonal access to a memory array was first proposed by Tseng, Hwang, and Prasanna Kumar [29], to provide optimal speed up for matrix-based computations and sorting. Ja'Ja' and Owens [13], have proposed a somewhat similar array for computing the Fast Fourier Transform and sorting; also, Scherson and Sen [27] have considered the organization for sorting. Finally, Alnuweiri and Prasanna Kumar [1, 4, 5] have proposed efficient data movement techniques for this organization which lead to processor-time optimal solutions to a wide class of image and vision problems and graph theoretic problems.

In [1], the organization has been called a Reduced Mesh of Trees (RMOT) because it can be viewed as mesh of trees organization in which the base PEs are replaced by memory modules, and each row (column) tree of PEs is replaced by one PE with a row (column) bus.

2.1 A Review of the RMOT Organization

An RMOT of size m consists of m PEs each having row and column access to an $m \times m$ array of memory modules such that PE_i can access the modules in the ith row and the ith column of the array. The organization of an RMOT with 4 PEs is shown in figure 1, where the dashed lines represent data-links whose purpose will be explained in the next section.

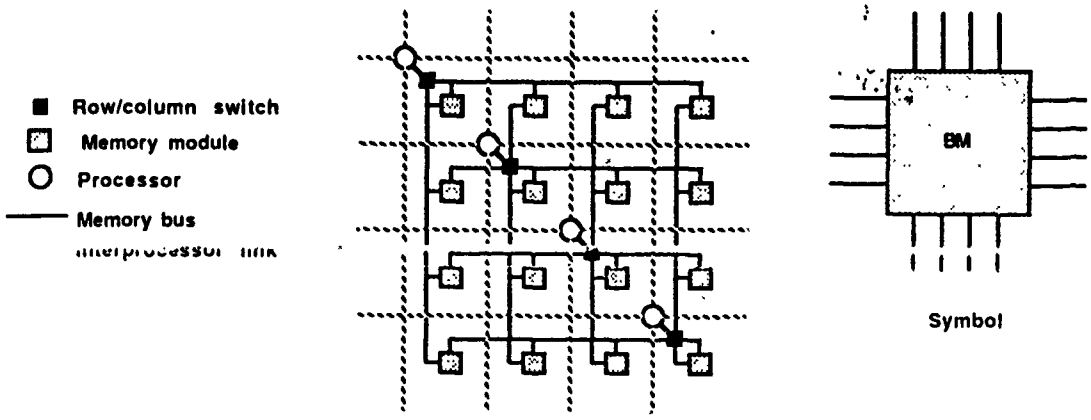


Figure 1: Organization of an RMOT (Basic Module) with 4 PEs

The PEs have basic arithmetic/logic capabilities and all their internal registers and data paths are $O(\log n)$-bit wide. Also, each memory module consists of a number of $O(\log n)$-bit registers each of which is considered to be one *memory location* within the memory module. The distinction between memory *locations* and memory *modules* should be noted. Each PE can read or write one unit ($\log n$ bits) of data in a memory location in its row or in its column in $O(1)$ time. This organization has the advantage that the processing area and the storage area are identified as two distinct components of the design. Further architectural

and operational details can be found in [5].

2.2 The Mesh-Connected Modules (MCM) Organization

A computation on the RMOT can be done by decomposing it into alternating *row-phases* and *column-phases*, where in a row-phase (column-phase) the entire computation is performed on data within the same row (column) of memory. In nontrivial applications, the time to perform a computation is dominated by the time taken by each PE to perform the computation on its memory row and memory column. The proposed architecture enhances the time performance of the RMOT by augmenting it with more PEs. Each PE with its row or column bus in the RMOT is replaced by a linear array of PEs, and the memory modules within each row or column are partitioned equally among these PEs, as shown in figure 2. Note that now we have two types of communication links, *interprocessor links* and *memory buses*. To save on the number of processors, we let each row and column array of

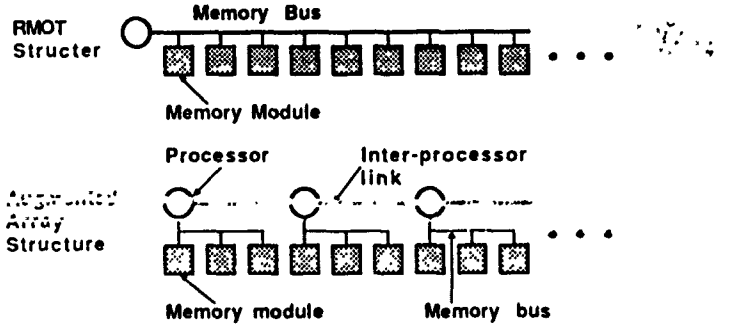


Figure 2: Augmenting an RMOT row with more PEs

PEs share one PE. The resulting organization is shown in figure 3 (the distinction between interprocessor links and memory buses is not shown in this figure). It is interesting to note that the proposed organization consists of smaller RMOT arrays (modules) which are connected as a mesh. A formal definition of the architecture and some additional notation are given below.

Definitions and Notation

The Mesh-Connected Modules (MCM) organization consists of p PEs and $O(n^2)$ memory locations. The PEs operate in a synchronous mode under the supervision of one control

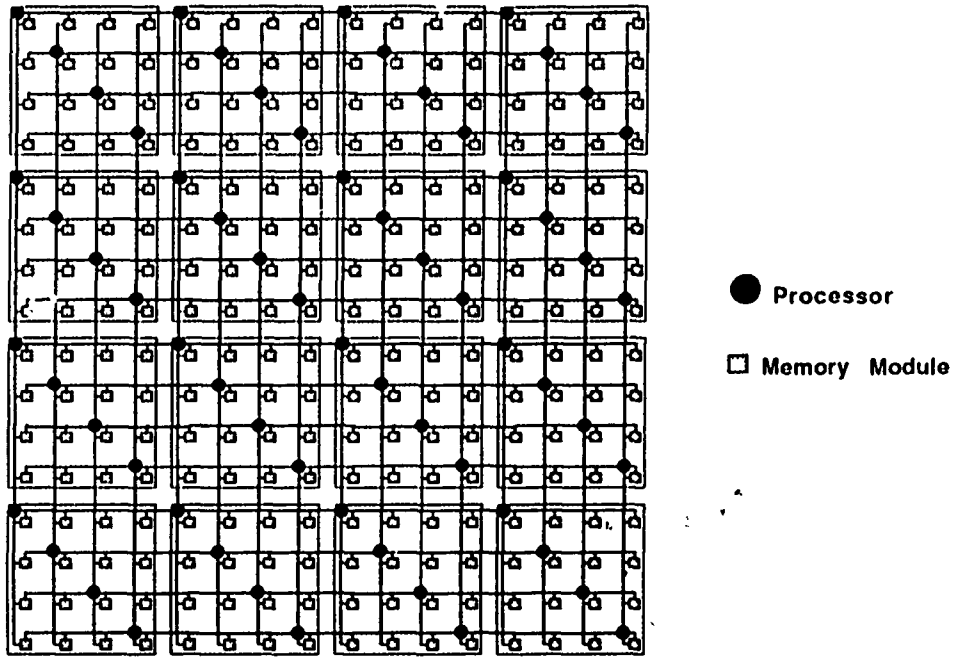


Figure 3: The Mesh-Connected Modules Organization

unit. This mode of operation is usually known as SIMD (Single-Instruction Multiple Data). The MCM is organized as a $k \times k$ mesh-connected array of *basic modules* (BMs), where $1 \leq k \leq n$. Each BM is an RMOT organization with q PEs $(PE_0, \cdots, PE_{q-1})$ where $1 \leq q \leq n/k$; and a $q \times q$ array of memory modules each of size $O((n/kq)^2)$. In addition, each PE_i of a BM is connected to PE_i in each adjacent BM. The memory module in the ith row and jth column of the array is denoted by MM_{ij}. Also, a memory row is denoted by RM, and a memory column is denoted by CM. It is assumed that $O(\log n)$ bits can be moved across each bus or data link in a constant amount of time. The organization of a BM was shown in figure 1, where the dashed lines represent the interprocessor links between each PEs and its four possible neighbor PEs.

An MCM can be specified by the three parameters n, k and q, described above, where $1 \leq k \leq n$ and $1 \leq q \leq n/k$. We use the notation MCM(n, k, q) to denote an MCM having $O(n^2)$ words of memory and organized as a $k \times k$ array of BMs, each having q PEs. The total number of PEs in the MCM is $p = k^2 q$, and the total number of memory modules is $k^2 q^2$ (i.e. a total of $O(n^2)$ memory locations). Let $m = n/k$, then each memory module has $O(\frac{m^2}{q^2})$ memory locations, and the memory array of each BM has a total of $O(m^2)$ memory locations. Several problems can be solved by recursively partitioning the MCM into submeshes of BMs, solving the subproblem in each submesh, then merging the results.

A *j-partition* of the MCM, $1 \leq j \leq k$, is a partition of the MCM into k^2/j^2 submeshes, each consisting of an array of $j \times j$ BMs. Each such submesh is called a *j-mesh* and denoted by $M^j(i)$, where i is the index of the submesh in the partition. Note that a *j-mesh* is also an MCM(jm, j, q), where $m = n/k$. Also, by definition, a BM s a 1-*mesh*.

3 Main Results

It was mentioned earlier that an MCM(n, k, q) can solve several image and vision problems in $O(n^2/p)$ time using $p = k^2q$ PEs, $1 \leq p \leq n^{3/2}$. For a large number of vision and image problems it can be shown that the MCM is superior to the standard mesh computer [16] and the pyramid computer [19]. Table 1 compares the performance of an MCM$(n, \sqrt{n}, \sqrt{n})$ having $O(n^{3/2})$ PEs with that of the pyramid computer and the standard mesh-connected computer each having $O(n^2)$ PEs. Definitions and algorithmic details for these problems are given in section 4.

	2MCC	Pyramid	MCM$(n, \sqrt{n}, \sqrt{n})$
Number of processors	n^2	$O(n^2)$	$n^{3/2}$
Labeling figures	n	$n^{1/2}$	$n^{1/2}$
Convex hull:			
1. Set of pixels	n	$\log^2 n/\log\log n$	$n^{1/2}$
2. Multiple figures	n	$n^{1/2}\log n$	$n^{1/2}$
Properties:			
Diameter	n	$n^{1/2}\log n$	$n^{1/2}$
Smallest enclosing rectangle	n	$n^{1/2}\log n$	$n^{1/2}$
Distances:			
1. All-point closest neighbor	n	$\log n$	$n^{1/2}$
2. Closest figure	n	$n^{1/2}$	$n^{1/2}$

Table 1: Performance comparison of an MCM with pyramid and mesh computers

As an enhancement to the standard mesh, Miller and Stout [18] considered a *variable-diameter* mesh which consists of a $\sqrt{p} \times \sqrt{p}$ array of PEs, each having a local memory of size n^2/p locations. They have shown that the problems listed in table 1 can be solved in $O(n^2/p)$ time on the variable-diameter mesh, where $1 \leq p \leq n^{4/3}$. The MCM is superior to this mesh since it provides $O(n^2/p)$ time solutions for $1 \leq p \leq n^{3/2}$. In fact, it is easy to see that the variable-diameter mesh is a special instant of the MCM(n, k, q) obtained

by choosing $q = 1$ (thus $p = k^2$, and $1 \leq k \leq n^{2/3}$). Thus, the MCM algorithms can be implemented directly on the variable-diameter mesh.

4 Parallel Vision Algorithms on the MCM

This section presents optimal solutions to several image problems on the MCM. Our approach is based on combining the divide-and-conquer paradigm with the optimal parallel techniques that can be implemented on the basic modules. The MCM techniques basically consist of performing parallel merge on the results from the basic modules. Optimality is achieved by carefully balancing the BM techniques and the merge operations as will be shown later. Processor-time optimal parallel algorithms for solving several image and vision problems on a BM have been presented in [1, 5]. Table 2 summarizes these results. These algorithms form the basic subtechniques for the MCM algorithms.

Radix-Sorting of m^2 integers
Parallel Prefix Computations on m^2 elements
Problems on an $m \times m$ Image:
1. Histogram computations
2. Histogram modification (equalization)
3. Labeling connected components
4. Convex hull of each figure
5. Diameter of each figure
6. Smallest enclosing rectangle of each figure
7. All-point closest neighbors
8. Closest figures

Table 2: Problems that can be solved in $O(m^2/q)$ time on a BM with q PEs

(Alnuweiri and Prasanna Kumar [1, 5])

Before we proceed further, it is necessary to introduce some additional notation which will clarify the description of our algorithms. The image is initially partitioned into k^2 disjoint squares each of size $m \times m$ pixels, where $m = n/k$. Each such square is called a *basic-square* and stored in a distinct BM. Given an $n \times n$ image F, a *j-partition* of the F is defined to be the partitioning of F into disjoint squares each consisting of $j \times j$ basic-squares of the image. Each such square is called a *j-square* and denoted by $Q^j(i)$, where

$i = 0, 1, \cdots, (k/j)^2 - 1$ is the index assigned to each square. Note that the size of a *j-square* is $jm \times jm$ pixels. The squares of a *j-partition* of the image are indexed such that $Q^j(i)$ is stored in $M^j(i)$. For $j \leq 2$, each $Q^j(j)$ can be partitioned into four equal quadrants, denoted by Q_0^j, Q_1^j, Q_2^j, Q_3^j. Also, note that $Q^0(i)$ denotes the basic-square in $BM(i)$. The *boundary* of a *j-square* consists of topmost and bottom-most rows of pixels and the leftmost and rightmost columns of pixels within the *j-square*. The boundary of an image square Q is denoted by $\delta(Q)$. The pixels on the boundary of a *j-square* $Q^j(i)$ are stored in the PEs of the boundary BMs of $M^j(i)$.

4.1 Histogramming on the MCM

Histogramming is a widely used operation in the early processing of images, and in image enhancement applications. The histogramming problem can be defined as follows [26]: Given an $n \times n$ image I having H gray-level values $0, 1, \cdots, H-1$, compute the number of pixels having the gray-level value g for each $g = 0, 1, \cdots, H-1$. Each gray-level value can be represented by a binary number $g_{h-1}, g_{h-2}, \cdots, g_0$, where $h = \lceil \log H \rceil$. The histogramming problem for an $m \times m$ image can be solved in $O(m^2/q)$ time on a BM with q PEs by using the parallel radix-sort technique described in [5] (see table 1). This result is processor-time optimal since, sequentially, this problem requires $\Omega(m^2)$ time. Histogramming can be performed efficiently on the MCM by using data reduction and divide-and-conquer techniques. Our technique consists of two stages: a *sorting* stage, and a *key-counting* stage. Suppose that the given $n \times n$ image consists of H different gray-levels, the two stages can be performed as follows:

- **Sorting Stage:** Partition the MCM into *j-meshes* each having a total of at least H memory locations, i.e. $j^2m^2 \geq H$ (for simplicity, assume that $H = j^2m^2$). Each memory location in such a *j-mesh* corresponds to a distinct key (gray-level value). Within each *j-mesh*, sort the pixels by their gray-level values, then compute the sum of the pixels in each gray-level, and finally, store the sum into the memory location corresponding to that gray-level key. It is clear the time complexity of this stage is dominated by sorting the pixels within each *j-mesh*, which is $O(j\frac{m^2}{q})$.

- **Key-Counting Stage:** This stage is performed using a divide-and-conquer technique. Since the initial size of each submesh is $j \times j$ BMs, this stage takes $\log k - \log j =$

$\log(k/j)$ iterations. In iteration i, the size of each submesh is $(j2^i \times j2^i)$ BMs. Because the number of different keys is fixed and equal to H, After each iteration the keys are divided equally among the RMs of each ℓ-*mesh*, $\ell = j2^i$. Furthermore, the counts (sum of elements) of the same key in different ℓ-*meshes*, are stored in the same memory location within each ℓ-*mesh*. Step i of this stage is performed as follows:

1. In this iteration, the sum of pixels for each key in a 2ℓ-*mesh*, $\ell = j2^i$, is to be computed by adding the four sums for that key in the four quadrants of the 2ℓ-*mesh*. Since these four sums are stored in the same memory location within each quadrant, this operation can be performed in $O(j2^i + \lceil \frac{m}{q} \cdot \frac{\sqrt{H}}{2^i} \rceil)$ time by rotating the data within each row and column of PEs in a 2ℓ-*mesh*.
2. Now we partition the keys among the rows of each 2ℓ-*mesh*, so that each RM will contain a smaller number of keys. This can be done in $O(j2^i + \lceil \frac{m}{q} \cdot \frac{\sqrt{H}}{2^i} \rceil)$ time by a simple rotation of data in each row and column of memory within each 2ℓ-*mesh*.

The total time taken by the $\log(k/j)$ iterations is

$$\sum_{i=0}^{\log(k/j)} j2^i + \lceil \frac{m}{q} \cdot \frac{\sqrt{H}}{2^i} \rceil$$

which is bounded by $O(k + \frac{m}{q} \cdot \sqrt{H})$. For processor-time optimality, we must have

$$\frac{m}{q}\sqrt{H} \leq \frac{m^2}{q} \Longrightarrow H \leq m^2,$$

i.e., the memory size of each BM must be equal or greater than the maximum number of keys (gray-levels). Thus, we have the following theorem:

Theorem 1 *Given an image with H or less gray-levels, computing the histogram of the image can be done in $O(k + \frac{m}{q} \cdot \sqrt{H})$ time on an MCM(n, k, q) with $p = k^2 q$ processors. If $H = O(m^2)$, where $m = n/k$, then the histogram can be computed on this MCM in $O(k + \frac{m^2}{q}) = O(k + \frac{n^2}{p})$ time, which is optimal.*

Histograms are used in a large number of applications such as image enhancement. For example, in a certain image some gray-levels may be heavily populated while other levels

may have a much lower number of pixels. To enhance the dynamic range of the image, a redistribution of gray-level values may be desired. This can be done by a technique called *histogram modification* [26], in which given a histogram H_M for an image F, it is required to transform the gray-scale of the image so as to give the image a specified histogram H_M^{new}. The new histogram has the same number of gray-levels as the old histogram but has a different number of pixels in each level. It is easy to show that image enhancement by this method can be performed on an MCM(n,k,q) in $O(k+\frac{m}{q}\sqrt{H})$ time, where H is the number of the gray levels. Again, if we choose $H=m^2$, then this time becomes $O(k+\frac{n^2}{p})$ which is optimal for $1 \leq p \leq n^{3/2}$ and $1 \leq k \leq n^{2/3}$.

4.2 Line Detection by the Hough Transform

The Hough Transform is a general technique for feature detection in images. The method involves accumulating evidence for parametrized objects from the image feature space [9]. The principle of line detection using the Hough Transform is based on transforming each pixel (i,j) in the image (also called the feature space) into a sinusoidal curve in the parameter space. This transformation is defined by the *normal representation* of a line (passing through (i,j)), which is given by

$$\rho = i\sin\theta + j\cos\theta. \tag{1}$$

The computation of the Hough Transform proceeds by associating an *accumulator cell* $A(s,t)$ with each point (ρ_s,θ_t) (in the parameter space) which keeps a count of the number of curves intersecting at that point. For computational purposes the ranges of ρ and θ are quantized into a number of levels. If the number of quantized levels of θ (respectively, ρ) is L (respectively, M), then we have ML accumulator cells each associated with a pair (ρ_s,θ_t) of quantized values, where $1 \leq s \leq M$ and $1 \leq t \leq L$.

Suppose that $\theta_1,\cdots,\theta_L$, are the L given samples of θ, then the Hough Transform can be computed using L histogramming steps. In Step i, each image pixel (i,j) increments the count of $A(\rho_s,\theta_t)$ by one, where $\rho_s = i\sin\theta_t + j\cos\theta_t$. Implementing this technique directly on the MCM(n,k,q) leads to $O(Lk+Ln^2/p)$ time. The term Ln^2/p in the time complexity is within the optimal bound; however, the term Lk is not for sufficiently large values of k and L, e.g. $L=O(n), k=O(n^{1/2})$. The method can be improved by modifying the key-counting stage of our histogramming technique. The complete algorithm is rather

lengthy to describe here. We will consider a special case which shows how optimality can be achieved on the MCM.

Consider computing the Hough Transform of an $n \times n$ image. Let L be the number of values of θ and M be the number of values of ρ, and let $H = LM$ be the total number of accumulator cells needed to compute the transform. Now let $M = m^2$ and $L = k^2$, where as before $m = n/k$. Each BM_i, $1 \leq i \leq k^2$, has m^2 memory locations which can store the $M = m^2$ values of ρ for a particular θ_i. In the computation each BM computes the Hough Transform for its image pixels for a certain θ. This can be done in $O(m^2/q)$ time by histogramming. The m^2 values of ρ are then shifted to an adjacent BM, so that values of ρ for a new value of θ can be computed. Using this technique each BM computes the values of ρ for the $L = k^2$ values of θ. Since each such computation takes $O(m^2/q)$ time, the total time taken by each BM is $O(k^2m^2/q)$. The data transfer time is also $O(k^2m^2/q)$. Since $H = LM = k^2m^2$, the computation time on an MCM(n, k, q) is $O(H/q) = O(L\frac{n^2}{p})$ using $p = k^2q$ PEs, which is optimal since a sequential computation of the HT would take $O(Ln^2)$ time.

Theorem 2 *Computing the Hough Transform can be done in $O(k + L\frac{n^2}{p})$ time on an MCM(n, k, q) with $p = k^2q$ processors, where L is the number of discrete values of θ in the parameter space.*

4.3 Computing Image Connected Components

A black and white (binary) image may consist of several connected regions of black pixels. A *connected component* (or a *figure*) is a maximally connected region of black pixels. The *image labeling* problem is to identify a unique label with each figure in the image. The labeling problem can be solved by using a bottom-up recursive procedure in which the figures in a $k \times k$ square are labeled first by labeling the connected regions within each $\frac{k}{2} \times \frac{k}{2}$ quadrant of the square, then by using the adjacency information along the common boundaries of the quadrants to label the connected regions within the $k \times k$ square. Nassimi and Sahni have proposed this approach for meshes [20], and since then it was implemented on several other parallel organizations such as the pyramid computer [19], MOT [23], and hypercube computers [8]. The same technique can be implemented on the MCM as follows:

Initially, the figures within each basic-square can be labeled in $O(\frac{m^2}{q})$ time by using a module-algorithm. Next, a recursive merging algorithm is used to compute the labels of the figures. The algorithm consists of $\log k$ iterations. In iteration i, $1 \leq i \leq \log k - 1$, the figures within each *j-square*, $j = 2^i$, are labeled by considering the pixels on the boundaries of its four quadrants. The merge is done first by constructing an auxiliary graph $G^j(h)$ representing the connectivity among the figures in the four quadrants of each $Q^j(h)$, then computing the connected components of each such graph. Each vertex in $G^j(h)$ represents a figure which intersects the boundary of a quadrant. The edges of $G^j(h)$ are constructed as follows: $(v_x, v_y) \epsilon E$ if two figures labeled x and y in two adjacent quadrants have at least two pixels which are 4-neighbors along the common boundary of the two quadrants. It is clear that G^j has $O(jm)$ vertices and $O(jm)$ edges.

Lemma 1 *Given the graph G^j constructed as above, and stored in a j-*mesh*, the connected components of this graph can be computed in $O(j + \sqrt{j} \cdot \frac{m}{q} \cdot \log(jm))$ time.*

Proof: The $O(jm)$ edges of the given graph G^j are stored in a column (or row) of boundary BMs. Let these edges be stored in the leftmost column of BMs in the *j-mesh*. The connectivity algorithm we use is basically an implementation of the parallel connectivity algorithm in [28]. For a graph with jm vertices, this algorithm requires $O(\log(jm))$ iterations on a CROW PRAM with $O(jm)$ PEs. In each step, each PE performs a constant time operation on its local data. To implement this algorithm on the MCM, each read/write step on the PRAM is simulated by a *random access read/write* operation [21] on the MCM. Once each PE has the appropriate data in its RM, it performs the desired operations on this data.

To implement this algorithm on a *j-mesh* in which G^j is stored, we first move the $O(jm)$ edges of the graph into a $2^{\lceil i/2 \rceil}$*-mesh* within the *j-mesh*, where as before $j = 2^i$. This can be done as follows: First partition the 2^i BMs into $2^{\lceil i/2 \rceil}$ groups, each consisting of $2^{\lfloor i/2 \rfloor}$ BMs. Move the data within each such group into a distinct column of BMs, such that the $2^{\lceil i/2 \rceil}$ groups are stored in $2^{\lfloor i/2 \rfloor}$ adjacent columns of BMs. This can be in $O(2^{\lceil i/2 \rceil} + m/q)$ time by moving the data within each row of PEs. The data within each such column of BMs is moved upward (within the *j-mesh*) so that all the edges are now in a $2^{\lfloor i/2 \rfloor} \times 2^{\lceil i/2 \rceil}$ array of BMs. If i is even then this array constitutes a *$\sqrt{j}$-mesh*. If i is odd then this array constitutes the upper half of a $2^{\lceil i/2 \rceil}$*-mesh*. Each PE within the smaller mesh has $O(m/q)$ edges only in its RM. This data movement is illustrated in figure 4, and it can be completed

in $O(2^i + 2^{\lceil i/2 \rceil} m/q)$ time.

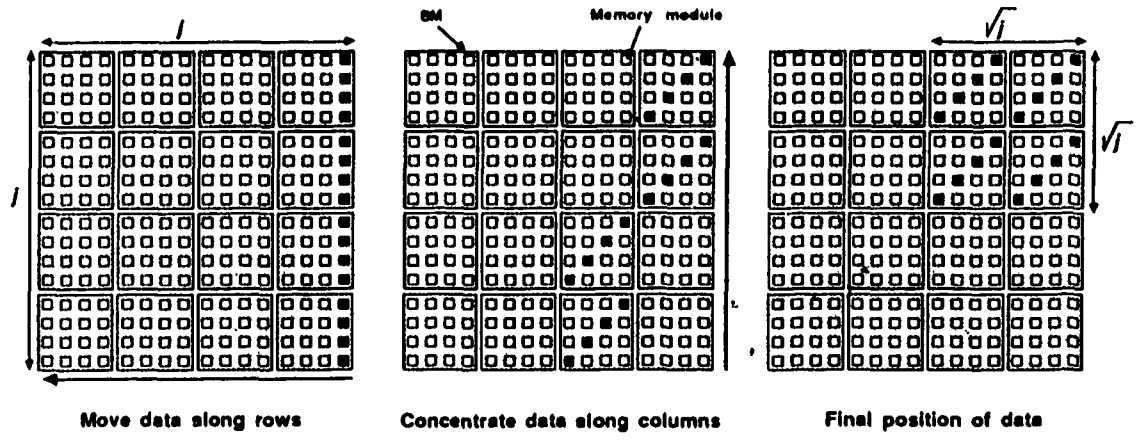


Figure 4: Concentrating data into a smaller submesh

Within the $2^{\lceil i/2 \rceil}$-*mesh*, each random access read/write operation can be performed in $O(2^{\lceil i/2 \rceil} m/q) = O(\sqrt{j} \cdot (m/q))$ time [2]. Thus, the $\log(jm)$ iterations of the connectivity algorithm in [28] can be performed in $O(\sqrt{j} \cdot (m/q) \cdot (\log jm))$ time. □

After computing the connected components of G^j, the correct labels of the pixels on the boundary of each j-*square* (and also on the boundary of its four quadrants) can be computed. The procedure is then repeated for each $(2j)$-*square*, $(4j)$-*square*, and so on until the entire image has been processed. The time for iteration i of the algorithm is $O(j + \sqrt{j} \cdot \frac{m}{q} \cdot \log(jm))$, where $j = 2^i$, and the time for the initial module computation is $O(m^2/q)$. Thus, the time for all $\log k$ iterations is,

$$m^2/q + \sum_{i=1}^{\log k} \left(2^i + 2^{i/2}(m/q)(\log 2^i m)\right)$$

which is bounded by $O(k + m^2/q) = O(k + n^2/p)$, where $p = k^2 q$.

Since only the labels of the pixels on the boundary of each quadrant are updated in each iteration, some pixels may still have old incorrect labels at the end of the computation. These pixels can be labeled with the correct labels by applying a top-down recursive procedure similar to the above procedure but in the reverse direction. That is, use the

labels of the pixels on the boundary of each *j-square* to label the pixels on the boundaries of its four quadrants. This is done starting from the square Q^k which consists of the entire $n \times n$ image, and then applying the technique until the size of each quadrant is $m \times m$ pixels. Each such quadrant is local to one BM and its pixels can be labeled with the correct labels in $O(m^2/p)$ time using a module-algorithm. The following theorem summerizes this result.

Theorem 3 *Given an $n \times n$ black and white image, all connected regions of black pixels in the image can be labeled in $O(k + n^2/p)$ time on an MCM(n, k, q) with p PEs, where $1 \leq p \leq n^{3/2}$, and $k \leq n^{2/3}$.*

Besides being optimal, our result is superior to previous known results. For example, using $n^{3/2}$ the MCM can provide $O(n^{1/2})$ time solution to the above problem. A pyramid computer can provide the same solution time but using n^2 PEs [19]. A 2MCC with n^2 PEs provides $O(n)$ time solution to this problem [20]. The hypercube and shuffle-exchange networks can provide $O(\log^2 n)$ time solutions using n^2 PEs [8], but they require a much larger area if implemented in VLSI.

4.4 Convexity of Multiple Figures

We now consider the problem of computing the convex hull of each figure (connected component) in an $n \times n$ image in which all figures have been labeled. A set S of pixels is said to be *convex* if the smallest convex polygon containing them contains no other pixels. Such polygon is called the *convex hull* of S, and it is denoted by CH(S). The MCM algorithm is based on the divide-and-conquer paradigm for merging disjoint convex hulls [25]. Two convex hulls are said to be *disjoint* if there exists a vertical line in the plane such that all the vertices of one hull lie on or to the left of this line, and all the vertices of the second hull lie to the right of this line. By merging two convex hulls A_1 and A_2, we mean computing the convex hull $CH(A_1, A_2)$ of A_1 and A_2. Merging two disjoint convex polygons A_1 and A_2 can be done by computing the two *tangents* common to A_1 and A_2, and by eliminating the vertices which become internal to the resulting polygon. To compute these tangent lines, it is useful to partition A_1 (A_2) into an *upper hull* and a *lower hull*. Let ℓ and r be the two vertices with, respectively, the smallest and largest x-coordinates among all the vertices of

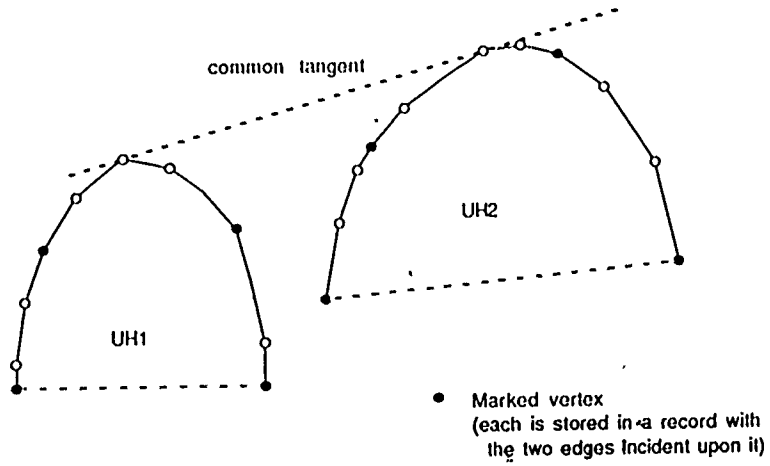


Figure 5: Partitioning convex hull

A_1 (A_2). The upper hull consists of all the vertices of A_1 (A_2) which lie on or above the line $\overline{\ell r}$, and the lower hull consists of all the vertices below this line.

Our merging technique is based on the following procedure: Suppose we have two convex hulls (polygons), CH1 and CH2. We consider the merge procedure for the upper two hulls of CH1 and CH2 (denoted UH1 and UH2, respectively). The same procedure can be applied to merge the lower hulls. Let UH1= $(v_1, v_2, \cdots, v_s)$ and UH2= $(u_1, u_2, \cdots, u_s)$, where the v_i's (u_i's) are the vertices UH1 (UH2) given in the increasing order of their $x-$coordinate (see figure 5). Mark j vertices $V = (v_{i_1}, v_{i_2}, \cdots, v_{i_j})$ from UH1 and j vertices $U = (u_{i_1}, u_{i_2}, \cdots, u_{i_j})$ from UH2, where $v_{i_1} = v_1, v_{i_j} = v_s, u_{i_1} = u_1$, and $u_{i_1} = u_s$. The marked vertices are also ordered by their $x-$coordinate. The sets V and U partition UH1 and UH2 into $j-1$ *segments*, where a segment $[v_{i_r}, v_{i_{r+1}}]$ ($[u_{i_r}, u_{i_{r+1}}]$) consists of all the vertices of UH1 (UH2) in between, and including, v_{i_r} and $v_{i_{r+1}}$ (u_{i_r} and $u_{i_{r+1}}$). Note that the segments do not necessarily contain the same number of vertices. Finally, we assume each vertex in V (U) is stored in a record together with the two edges from UH1 (UH2) incident upon it. The two hulls UH1 and UH2 can be merged in two stages as follows:

Procedure MERGE-HULL

- **Stage 1:** This stage performs the binary search procedure only on the vertices in V

and U and the edges incident upon them. Each iteration of the algorithm consists of choosing two vertices v_{i_k} and u_{i_ℓ}, constructing the line joining them, and then testing if this line is tangent to both UH1 and UH2 [25]. Thus, this stage terminates in at most $(\log j)$ iterations. Upon termination, either a common tangent is found (and we are done) or two segments of vertices, one from UH1 and one from UH2, are identified. These two segments contain the two vertices through which the common tangent passes.

- **Stage 2:** This stage computes the common tangent by performing the binary search procedure on vertices of the two segments identified in the previous stage. This stage terminates in at most $(\log t)$ iteration, where t is the total number of vertices in the two segments.

To merge pairs of convex hulls in two adjacent image squares, the above two stages must be applied simultaneously to all pairs of convex hulls which touch along the common boundary of the two squares. The following steps show how the above procedure can be implemented on the MCM:

1. **Initial module computation:** We start by computing the convex hulls for all the figures within each basic square. Since each basic square is local to one BM, this computation can be completed in $O(m^2/q)$ time using a BM algorithm. Note that if a figure extends over several basic squares, then this figure will have a convex hull in each one of these basic squares. All the convex hulls of one figure must be merged in order to obtain the final convex hull of the figure. For the merge phase, we need only to consider convex hulls of figures which have at least one pixel on the boundary of a basic square.

2. **Recursive-merge:** This phase consists of a recursive procedure which runs in $\log k$ iterations. In iteration i, $1 \leq i \leq \log k$, we compute the convex hulls of the figures within each *j-square*, Q^j, where $j = 2^i$. At the beginning of this iteration, the convex hulls of the figures within each quadrant of Q^j have been computed. So we need to merge each group of (at most four) convex hulls of the same figure in the four quadrants. It can be shown that all pairs of hulls can be merged in $\frac{2}{3}\log(jm)$ iterations by using a binary-search procedure [25]. Our merge procedure is based on implementing this binary search technique. The details of the implementation are

rather tedious and are left out to the full version of this paper. The following theorem summerizes our result.

Theorem 4 *Given an $n \times n$ image in which all figures have been labeled, the convex hull of each figure can be computed, for all figures simultaneously, in $O(n^2/p)$ time on an MCM(n,k,q) with p PEs, where $1 \leq p \leq n^{3/2}$ and $k \leq n^{2/3}$.*

Using a similar computation, several other geometric properties can be computed for each figure in the image. For example, computing the diameter and a smallest enclosing rectangle for each figure can be computed in $O(n^2/p)$ time on an MCM(n,k,q) with p PEs, where $1 \leq p \leq n^{3/2}$ and $k \leq n^{2/3}$. Again for $p = n^{3/2}$, the above problems can be solved in $O(n^{1/2})$ time, which is superior to the $O(n)$ time solution on a standard $n \times n$ mesh of PEs [16], and the $O(n^{1/2}\log n)$ solution on a pyramid computer of size n^2 [19].

4.5 Computing All Closest Neighbors

We consider two problems related to computing distances among image regions. The first problem we consider is the *closest neighbor* problem. In this problem we are given an $n \times n$ black and white image, and we are required to compute for each pixel p (either black or white) a closest black pixel. The computed black pixel will be called a *closest neighbor* of p. As a measure of distance, we will use the L_1 metric. In this metric the distance between two pixels (i,j) and (u,v) (denoted by DIST[(i,j),(u,v)] is given by $|i-u|+|j-v|$. A bottom-up recursive approach can be used for this problem. To compute the closest neighbor for each pixel in a *j-square* square Q^j, we first solve the problem within each quadrant of Q^j and then merge the results from the quadrants. It can be shown that in the merge step only the boundary pixels of the quadrants need to be considered.

The divide-and-conquer procedure proceeds as follows: Initially, the closest neighbors are computed within each basic square. Since each basic square is local to one BM, this can be done in $O(m^2/q)$ time using a module-algorithm. A bottom-up recursive algorithm is then used. In the beginning of iteration i of this algorithm, $1 \leq i \leq \log k$, the boundary pixels of each quadrant of a *j-square*, $j = 2^i$, have computed their closest neighbors within that quadrant. Iteration i consists of computing the closest neighbors of the boundary

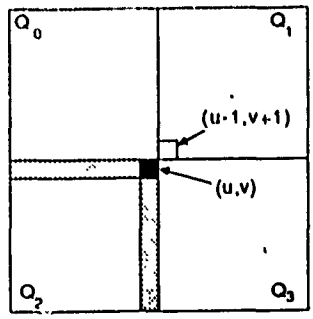


Figure 6: Closest neighbor computations

pixels of each *j-square*, by merging the information along the boundaries of its quadrants.

Theorem 5 *Given an $n \times n$ black and white image, the closest neighbor of each pixel in the image can be computed in $O(k + n^2/p)$ time on an MCM(n, k, q) with p PEs, $1 \leq p \leq n^{3/2}$ and $k \leq n^{2/3}$.*

Proof: Let M^j be the *j-mesh* containing the *j-square*, Q^j, and let $M^{(j/2)}(\ell)$, be the $(j/2)$-*mesh* (within M^j) which contains quadrant Q^j_ℓ, $0 \leq \ell \leq 3$, of Q^j (see figure 6). The major step in the merge procedure is the data transfer needed to enable each pixel on the boundary of a quadrant to compute its nearest neighbors in the other three quadrants. Once a pixel has this information, computing its nearest neighbors can be done in $O(1)$ time. Consider the pixels on the boundary of quadrant Q^j_2. Each PE in the rightmost column (topmost row) of BMs in $M^{(j/2)}(2)$ reads the closest neighbors of the boundary pixels in its neighbor PE in the leftmost column (bottom-most row) of BMs in $M^{(j/2)}(3)$ ($M^{(j/2)}(0)$). Since each such PE has $O(m/q)$ boundary pixels in its RM, this operation can be completed in $O(m/q)$ time. The pixel $(u-1, v+1)$ in the south-western corner of Q^j_1 can be also moved into the PE containing pixel (u, v) in the north-eastern corner of Q^j_2, in $O(1)$ time. Now the PE containing (u, v) can compute the closest neighbors of (u, v) within quadrants Q^j_0, Q^j_1, and Q^j_3. These three closest neighbors are then broadcast to all boundary PEs in $M^{(j/2)}(2)$. This can be done in $O(j + \log q + m/q)$ time. Now each PE in the rightmost column and topmost row of BMs in $M^{(j/2)}(2)$, has all the information necessary to compute the correct distances for each pixel in its portion of the boundary. Since each boundary PE has $O(m/q)$ such pixels, this can be done in $O(m/q)$ time. A similar computation is performed to update

the closest neighbors of the pixels on the boundary of Q^j. Thus, iteration i of the merge algorithm requires $O(j + \log q + m/q)$ time, where $j = 2^i$. The time taken by the entire algorithm is

$$\frac{m^2}{q} + \sum_{i=1}^{\log k} \left(2^i + \frac{m}{q} + \log q\right) = O(k + \frac{m^2}{q}).$$

Where the $O(m^2/q)$ term is the time taken by the initial module-algorithm. Since $m = n/k$ and $p = k^2 q$, the time taken by the algorithm is $O(k+n^2/p)$ which is optimal for $1 \leq p \leq n^{3/2}$ and $k \leq n^{2/3}$. □

A closely related problem is the problem of computing for each figure F, the label of the nearest figure to it in the image. As before, define DIST(p, q) to be the distance between two pixels, p and q according to L_1 metric. Then the label of the nearest figure to figure F, is the label of the pixel q such that

$$\text{DIST}(p, q) = \min_{p,q} \{\text{DIST}(p, q) | p \epsilon F, q \notin F\}$$

Corollary 1 *Given an $n \times n$ image in which all figures have been labeled, a nearest figure to each figure can be identified, for all figures simultaneously, in $O(n^2/p)$ time on an MCCM(n, k, q) with p PEs, where $1 \leq p \leq n^{3/2}$ and $k \leq n^{2/3}$.*

Proof: Let $\mathcal{L}(p)$ denote the label assigned to a pixel p, this problem can be solved in two major steps. In the first step, we compute for each pixel p, a closest neighbor q such that $\mathcal{L}(q) \neq \mathcal{L}(p)$. This can be done by using a simple modification of the above algorithm for computing closest neighbors. In the second step, the nearest figure to each figure is computed. This can be done by using a slight variation of the component labeling algorithm. In addition to keeping track of the minimum indexed pixel in the component, we also keep track of the label of the closest neighbor with the minimum distance from a pixel in the component. From theorem 3 we know that this can be done in $O(n^2/p)$ time. At the end of the computation, each pixel labeled F knows the label of the nearest figure to F. □

5 Concluding Remarks

We have presented several processor-time optimal algorithms for digitized images on a mesh-connected processor array having p PEs and $O(n^2)$ memory. The organization uses novel communication capabilities and can be implemented in current VLSI technology. The proposed organization has several features which distinguish it from other mesh-based parallel computers. First, the PEs communicate through a shared memory as well as direct interprocessor links. Second, the MCM is, in fact, a hybrid of a mesh-connected array and the communication-efficient organization of the basic modules (BMs). This two-level organization of the MCM implies two types of parallel algorithms: basic-module algorithms and communication-algorithms. Balancing the size of each BM and the number of BMs in the MCM results in processor-time optimal algorithms over a wide range of processor counts. The image problems considered here require global operations on image pixels, but are not computationally intensive (they require $O(n^2)$ operations for an $n \times n$ image). To provide optimal solutions to such problems, several efficient data movement operations have been developed for the MCM. All of these operations are efficient for applications in which the amount of data in the memory array is reduced (i.e. less than $O(n^2)$), which is the case for several of the image problems that we have considered. The MCM organization has been considered to provide optimal speedups for several classes of problems such as sorting [2] and adjacency matrix based graph computations [4].

References

[1] H. M. Alnuweiri and V. K. Prasanna Kumar, "Efficient Image Computations on VLSI Architectures with Reduced Hardware", Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, Seattle, WA, 1987.

[2] H. M. Alnuweiri and V. K. Prasanna Kumar, "Optimal VLSI Sorting with Reduced Number of Processors", Technical Report, IRIS 225, USC, 1987.

[3] H. M. Alnuweiri and V. K. Prasanna Kumar, "Optimal Geometric Algorithms on Fixed-Size Linear Arrays and Scan Line Arrays", Proc. IEEE Conference on Computer Vision and Pattern Recognition, 1988.

[4] H. M. Alnuweiri and V. K. Prasanna Kumar, "A Reduced Mesh of Trees Organization for Efficient Solutions to Graph Problems", Proc. the 22nd Annual Conference on Information Science and Systems (CISS), Princeton University, March 1988.

[5] H. M. Alnuweiri and V. K. Prasanna Kumar, "Optimal Image Computations on Reduced VLSI Architectures", to appear in *IEEE Transactions on Circuits and Systems*, October, 1989.

[6] H. M. Alnuweiri and V. K. Prasanna Kumar, "Fast Image Labeling Using local Operators on Mesh-Connected Computers", Proc. International Conference on Parallel Processing, 1989.

[7] K. E. Batcher, "Design of A Massively Parallel Processor", *IEEE Transactions on Computers*, c-29, September 1980.

[8] R. Cypher, J. L. C. Sanz, L. Snyder, "Hypercube and Shuffle-Exchange Algorithms for Image Component Labeling", Proc. IEEE Workshop on Pattern Analysis and Machine Intelligence, Seattle, WA, 1987.

[9] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures", *Communications of the ACM*, 15, 1, 1972.

[10] C. R. Dyer, "A VLSI Pyramid Machine for Hierarchical Parallel Image Processing", Proc. IEEE Conference on Pattern Recognition and Image Processing, 1981.

[11] C. R. Dyer and A. Rosenfeld, "Parallel Image Processing by Memory Augmented Cellular Automata", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1981.

[12] K. Hwang and P.S. Tseng and D. Kim, "An Orthogonal Multiprocessor for Parallel Scientific Computations", *IEEE Transactions on Computers*, c-38, January 1989.

[13] J. Ja'Ja' and R. M. Owens, "An Architecture for a VLSI FFT Processor", *INTEGRATION the VLSI Journal*, 1, 1983.

[14] F. T. Leighton, "Parallel Computations on Meshes of Trees", Technical Report, MIT 1982.

[15] R. Miller, V. K. Prasanna Kumar, D. Reisis, Q. F. Stout, "Meshes with reconfigurable buses", Proc. MIT Conference on Advanced Research on VLSI, Cambridge, MA, 1988.

[16] R. Miller and Q. F. Stout, "Geometric Algorithms for Digitized Pictures on A Mesh-Connected Computer", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, March 1985.

[17] R. Miller and Q. F. Stout, "Varying Diameter and Problem Size on Mesh-Connected Computers", Proc. International Conference on Parallel Processing, 1985.

[18] R. Miller and Q. F. Stout, "Varying Diameter and Problem Size on Mesh-Connected Computers", Proc. International Conference on Parallel Processing, 1985.

[19] R. Miller and Q. F. Stout, "Data Movement Techniques for the Pyramid Computer", *SIAM Journal on Computing*, 2, 1987.

[20] D. Nassimi and S. Sahni, "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", *SIAM Journal on Computing*, 9, 4, 1980.

[21] D. Nassimi and S. Sahni, "Data Broadcasting in SIMD Computers", *IEEE Transactions on Computers*, c-30, 2, February 1981.

[22] D. Nath, S. N. Maheshwari, P. C. P. Bhatt, "Efficient VLSI Networks for Parallel Processing Based on Orthogonal Trees", *IEEE Transactions on Computers*, c-32, 6, June 1983.

[23] V. K. Prasanna Kumar and M. Eshaghian, "Parallel Geometric Algorithms for Digitized Pictures on Mesh of Trees Organization", Proc. International Conference on Parallel Processing, 1986.

[24] V. K. Prasanna Kumar and C. S. Raghavendra, "Array Processor with Multiple Broadcasting", *Journal of Parallel and Distributed Computing*, 4, 173-190 (1987).

[25] F. Preparata and S. J. Hong, "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", *Communications of the ACM*, February, 1977.

[26] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.

[27] I. D. Scherson and S. Sen, "Parallel Sorting in Two-Dimensional VLSI Models of Computation", *IEEE Transactions on Computers*, c-38, 2, 1989.

[28] Y. Shiloach and U. Vishkin, "An $O(logn)$ Parallel Connectivity Algorithm", *Journal of Algorithms*, 3, 1982.

[29] P. S. Tseng, K. Hwang, V. K. Prasanna Kumar, "A VLSI Based Multiprocessor Architecture for Implementing Parallel Algorithms", Proc. International Conference on Parallel Processing, 1985.

[30] J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.

[31] S. H. Unger, "A Computer Oriented towards Spatial Problems", *Proceedings of the IRE*, v. 46, pp. 1744-1754, 1958.