

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2004

Efficient algorithms for the optical multi-trees (OMULT) architecture.

Mohammad Rabiul Islam
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Islam, Mohammad Rabiul, "Efficient algorithms for the optical multi-trees (OMULT) architecture." (2004).
Electronic Theses and Dissertations. 1879.
<https://scholar.uwindsor.ca/etd/1879>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Efficient Algorithms for the Optical Multi-Trees (OMULT) Architecture

by
Mohammad Rabiul Islam

A Thesis

Submitted to the Faculty of Graduate Studies and Research through the School of
Computer Science in Partial Fulfillment of the Requirements for the Degree of
Master of Science at the University of Windsor

Windsor, Ontario, Canada

2004



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-494-00125-9

Our file Notre référence

ISBN: 0-494-00125-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Mohammad Rabiul Islam 2004

© All Rights Reserved

Efficient Algorithms for the Optical Multi-Trees (OMULT) Architecture

**by
Mohammad Rabiul Islam**

APPROVED BY:

H. Wu, External Examiner,
Department of Electrical and Computer Engineering

A. Aggarwal, Internal Examiner,
School of Computer Science

S. Bandyopadhyay, Supervisor,
School of Computer Science

D. Wu, Supervisor, Chair
School of Computer Science

Date: June 14, 2004

Abstract

In this thesis, we have reported our investigations on efficiently implementing algorithms on the recently proposed Optical Multi-Trees (OMULT) multi-processors interconnection architecture that uses both electronic and optical links among processors. We have investigated algorithms for matrix multiplication of two matrices of size $n^2 \times n^2$ and two matrices of arbitrary size, the prefix-sum of a series and some fundamental computational geometry problems. We show that some common algorithms for computational geometry – finding the convex hull, the smallest enclosing box, the empirical cumulative distribution function and the all-nearest neighbor problems of n data points can be computed on the OMULT network in $O(\log n)$ time, compared to $O(\sqrt{n})$ algorithms on the Optical Transpose Interconnection System (OTIS) mesh for each of these problems.

Finally we have implemented our algorithm for matrix multiplication using the SimJava simulation tool and feel that this is a convenient environment for testing such parallel algorithms.

Keywords: Optical Interconnect systems, OTIS-Mesh, Optical Multi-Trees (OMULT), Matrix Multiplication, Prefix-Sum, Convex Hull, Smallest Enclosing Box, Empirical Cumulative Distribution Function, All-Nearest Neighbor.

Acknowledgements

One of the great pleasures of conducting this thesis work is acknowledging the efforts of many people whose names may not appear on the cover, but whose hard work, cooperation, advice, friendship and understanding were crucial to the production of the thesis. I am fortunate to have been able to work with the talented and dedicated team- Dr. Bhabani P. Sinha and Dr. Subir Bandyopadhyay. I first wish to thank both Dr. Bhabani P. Sinha and Dr. Subir Bandyopadhyay for their notable and original contributions to my research work.

My special thank to my respected supervisor Dr. Subir Bandyopadhyay for his loving support, encouragement, useful comments, direction and patience with my long hour and funny schedule. I would like to take this opportunity to thank Dr. Huapeng Wu and Dr. Akshai Aggarwal for their helpful suggestions, time and cooperation.

Thanks to all of my friends for sharing their experience and suggestions. I am also grateful to Mrs. Bharati Bandyopadhyay for her inspiration and loving support.

Finally, I want to express my gratitude to my mother, brothers and sisters for their constant encouragement and support. The sparkling smile of my child Sindeed Islam is the source of my main inspiration. Special thank to my wife Nahid Afroz who participated in producing of this thesis in more ways. Their support and understanding was a fantastic help when I disappeared to my computer for many long hours. Without their backing this task would never have been completed.

Table of Contents

Abstract.....	iv
Acknowledgements.....	v
List of Figures	ix
List of Tables	x
Chapter 1: Introduction	1
1.1 Preamble	1
1.2 Work reported in this thesis	2
1.2.1 Matrix multiplication	3
1.2.2 Prefix-sum computation	4
1.2.3 Computational geometry problem.....	5
1.3 Implementation of matrix multiplication algorithm on OMULT.....	5
1.4 Organization of the thesis.....	6
Chapter 2: Review of Literature.....	7
2.1 Introduction	7
2.2 Parallel computers	7
2.3 Interconnection Architecture.....	9
2.4 Interconnection technologies.....	9
2.5 Simple Interconnection Architecture	11
2.5.1 Linear Array	11
2.5.2 Tree Interconnections.....	11
2.5.3 2D-Mesh Interconnection.....	12
2.6 Interconnection architecture based on opto-electronic technology.....	13
2.6.1 OTIS Mesh architecture	13
2.6.2 OMULT Architecture.....	16
2.7 Some Important Properties of the OMULT Network	18
2.8 Algorithms for the OMULT Architecture.....	18
2.8.1 Row/column group broadcast	19

2.8.2 Data Broadcast	20
2.8.3 Complete Group-Broadcast.....	21
2.8.4 Sorting.....	22
2.8.5 Matrix Multiplication.....	23
2.8.6 Prefix-sum.....	25
2.8.7 Computational Geometry Problems.....	26
2.8.7.1 Convex Hull	26
2.8.7.2 Smallest Enclosing Box (SEB)	29
2.8.7.3 Empirical Cumulative Distribution Function (ECDF).....	31
2.8.7.4 All-Nearest Neighbor.....	33
Chapter 3: Algorithms in Matrix Multiplication and Prefix Sum on OMULT	
Network.....	35
3.1 Introduction	35
3.2 Matrix Multiplication: Two square Matrices of size $n^2 \times n^2$	35
3.3 Matrix Multiplication where the two operands may have any size.....	42
3.4 Prefix Sum for n^2 data elements.....	45
3.5 Prefix Sum for n^3 data elements.....	53
Chapter 4: Algorithms in Computational Geometry on OMULT	55
4.1 Introduction	55
4.2 Convex Hull	55
4.3 Smallest Enclosing Box	64
4.4 Empirical Cumulative Distribution Function (ECDF).....	67
4.5 All-Nearest Neighbor	69
Chapter 5: Network Simulation.....	71
5.1 Purpose of the Simulation	71
5.2 SimJav Simulation.....	72
5.3 Problem Simulated	72
5.4 Modeling of the System	73
5.4.1 The nodes of the trees	73
5.4.2 Connecting nodes	76
5.4.3 Communication between the nodes	76
5.5 Simulation Result	81
5.6 Critical review of the Simulation	81

Chapter 6: Conclusions and Future Work	83
6.1 Conclusions	83
6.2 Future Works.....	85
Appendix A: List of symbols	86
Appendix B: Glossary of important terms.....	87
Appendix C: Simulation	89
Appendix D: References	112
VITA AUCTORIS.....	127

List of Figures

FIGURE 2.1: 2D MESH INTERCONNECTION NETWORK.....	8
FIGURE 2.2: LINEAR ARRAY INTERCONNECTION.....	11
FIGURE 2.3: TREE INTERCONNECTION.....	12
FIGURE 2.4: 2D-MESH INTERCONNECTION	13
FIGURE 2.5: OTIS-MESH NETWORK.....	15
FIGURE 2.6: OMULT NETWORK.....	17
FIGURE 2.7(A): SET OF POINTS	27
FIGURE 2.7(B): CONVEX HULL OF S	27
FIGURE 2.8: POLAR ANGLE	27
FIGURE 2.9(A): SET OF POINTS.....	27
FIGURE 2.9(B): E IS AN EXTREME POINT	28
FIGURE 2.9(C): C IS AN EXTREME POINT.....	28
FIGURE 2.10: SMALLEST ENCLOSING BOX	31
FIGURE 2.11: SET OF POINTS	32
FIGURE 3.1: MATRIX X WITH SIZE $N^2 \times N^2$, WITH $N = 4$	35
FIGURE 3.2: MATRIX Y WITH SIZE $N^2 \times N^2$, WITH $N = 4$.....	36
FIGURE 3.3 BLOCK OF SUBMATRICES OF MATRIX X	37
FIGURE 3.4: DIVISION OF MATRICES INTO SUBMATRICES.....	43
FIGURE 3.5: DATA ARE STORED AT THE LEAF NODES OF THE TREES	49
FIGURE 3.6: THE DISTRIBUTION OF DATA AFTER STEP 2	50
FIGURE 3.7: AFTER STEP 3.....	50
FIGURE 3.8: AFTER STEP 4	51
FIGURE 3.9: AFTER STEP 5.....	51
FIGURE 3.10: AFTER STEP 6	52
FIGURE 3.11: AFTER STEP 7.....	52
FIGURE 4.1: SET OF POINTS	58
FIGURE 4.2: POINTS ARE STROED AT THE LEAF NODE.....	59
FIGURE 4.3 AFTER STEP 2.3	59
FIGURE 4.4: AFTER STEP 3.3.....	60
FIGURE 4.5: AFTER STEP 4.....	60
FIGURE 4.6: AFTER STEP 5.....	61
FIGURE 4.7: AFTER STEP 6.....	61
FIGURE 4.8: AFTER STEP 8.....	62
FIGURE 5.1: A SIMULATION LAYOUT	62

List of Tables

TABLE 5.1: SUBCLASSES OF SIM_ENTITY	74
TABLE 6.1: PERFORMANCE OF ALGORITHMS ON THE OMULT	84

Chapter 1: Introduction

1.1 Preamble

In late 1940 Dr John von Neumann formulated the classical computer architecture, based on a single central processing unit, to execute machine instructions [1]. The control unit of such machines fetches the instruction to be executed and its operands from memory, sends this instruction and its operands to the central processing unit where the instruction is executed. The design objective from the very beginning was to build faster and more efficient processors to build a faster computer. The modern computers are many orders of magnitude faster and more powerful compared to earlier machines.

A different approach to designing faster computers is to use a number of processors working together to achieve a better performance. A *parallel computer* (also called a multi-processor machine) is a machine that consists of a collection of processing units, or processors that cooperate, to solve a problem, by working simultaneously on different parts of that problem [1]. The idea is that, if several operations are done in parallel, significant improvement of computer performance can be achieved through exploitation of parallelism [22]. One crucial issue in the design of multi-processor machines is to decide how the processors should communicate with each other. An *interconnection network* is used to provide connection among processors so that data can be transferred quickly between processors that need to share data. In order to achieve better execution performance of computer systems through parallelization, there have been considerable

efforts in designing interconnection network for parallel computers [12], [21] over the last few decades.

When fabricating multi-processor systems, the standard approach is to use *copper* lines to realize links between processors. One major problem in this approach is the speed and other technological limitations of copper based connections. Optical technology has been proposed as a solution to this problem. One of the recent architectures partially based on the optical technology is the Optical Transpose Interconnect System (OTIS) [25], [12], [40] in which the processors are partitioned into groups so that processors within each group are interconnected by *electronic links* and processors in different groups are interconnected by *optical links*. The OTIS-Mesh optoelectronic computer is a class of OTIS computers on which various fundamental algorithms have been conveniently mapped [27], [29], [35] – [39]. Very recently, Sinha and Bandyopadhyay [31] have introduced another opto-electronic computer system, called the Optical Multi-Trees (OMULT). The OMULT architecture uses, n^2 complete binary trees of processors, each having n leaf nodes and $n - 1$ internal nodes.

1.2 Work reported in this thesis

In this thesis we are reporting our work in developing some efficient algorithms for the OMULT architecture. The algorithms reported in this thesis include-

- 1) matrix multiplication of two matrices of size $n^2 \times n^2$,
- 2) matrix multiplication of two matrices of any arbitrary size,

- 3) computing the prefix-sum of n^2 data elements,
- 4) computing the prefix-sum of any arbitrary number of data elements,
- 5) computational geometry algorithms to find the
 - a. convex hull,
 - b. the smallest enclosing box,
 - c. the empirical cumulative distribution function and
 - d. all-nearest neighbor.

1.2.1 Matrix multiplication

Matrix multiplication is commonly used in the areas of graph theory, numerical algorithms, digital control, and signal processing [41]. Multiplication of large matrices requires a lot of computation time as its complexity is $O(n^3)$, where the size of the matrix is $n \times n$. Because most current applications require higher computational throughput, many researchers have tried to improve the performance of matrix multiplication [41]. Since there is little scope to improve sequential matrix multiplication algorithm, parallel algorithms for matrix multiplication have been proposed [41]. These algorithms use matrix decomposition based on the number of processors available and include the systolic algorithm [24], Cannon's algorithm [2], Fox and Otto's algorithm [8], PUMMA (Parallel Universal Matrix Multiplication Algorithm) [3], SUMMA (Scalable Universal Matrix Multiplication Algorithm) [11], and DIMMA (Distribution Independent Matrix Multiplication Algorithm) [3]. In these algorithms, a processor calculates a partial result using the sub-matrices of the supplied matrices and successively performs the same

calculation on new sub-matrices, adding the new results to the previous results. When all multiplication is complete, the root processor assembles the partial results and generates the product matrix. We have shown that the OMULT architecture is well-suited for matrix multiplication.

1.2.2 Prefix-sum computation

Prefix sum is very useful in scheduling and constraint satisfaction problems [1]. The problem descriptions are given below:

- We are given a series a_1, a_2, \dots, a_N
- The prefix sum problem is to compute the following sums
 - a_1
 - $a_1 + a_2$
 - $a_1 + a_2 + a_3$
 - $\dots \dots \dots$
 - $a_1 + a_2 + \dots + a_N$

In summary, the prefix sum problem is to compute $\sum_i a_i$ for all $i, 1 \leq i \leq n$. We have developed an efficient algorithm to compute these sums using the OMULT architecture.

1.2.3 Computational geometry problem

Computational geometry problem deals with algorithms that create, modify or describe geometric objects using computers [1]. It is very useful in designing 3D-graphics. In the past it was not possible to generate fully rendered movies due to lack of availability of sufficiently powerful computers. However, over the last two decades, it has become possible to develop parallel algorithms for computational geometry problems [1]. This thesis investigates parallel algorithms for some basic computational geometry problems using the OMULT architecture. The details of the relevant computational geometry (CG) problems are described in chapter 2. In chapter 4 we will show that the CG problems we studied can be solved using the OMULT network [31] more quickly than the recently introduced OTIS-Mesh network [37].

1.3 Implementation of matrix multiplication algorithm on OMULT

We have simulated the algorithm for matrix multiplication of two $n \times n$ matrices on an OMULT architecture using the SIMJAVA package. The SIMJAVA package is a process based simulation tool based on Java. A SIMJAVA simulation is a collection of entities each running on its own thread. These entities are connected together by ports and can communicate with each other by sending and receiving event objects. A central system class controls all the threads, advances the simulation time, and delivers the events [42]. The SIMJAVA package is useful for a distributed memory systems, since it provides a widely used standard of message passing program and provides a portable, efficient, and flexible standard for message passing. Our simulation result shows that the SIMJAVA package is suitable for implementing the parallel algorithms for OMULT architecture.

1.4 Organization of the thesis

We organize the rest of the thesis as follows: In chapter 2, we give an overview of multiprocessors, interconnection architecture, the OTIS-mesh, the OMULT network and algorithms for the matrix multiplication, prefix-sum and computational geometry problems. In chapter 3 we provide several new matrix multiplication and prefix-sum algorithms and time analysis on the OMULT system. We present and analyze the algorithms for computational geometry problems on the OMULT network in Chapter 4. Chapter 5 contains the detailed discussions on the simulation of the OMULT network and a critical summary of this work. Chapter 6 describes the future trends and finally the conclusions of the thesis. Appendix A through D contains the list of symbols used, glossary of the terms used, SimJava program for Simulation and the list of references.

Chapter 2: Review of Literature

2.1 Introduction

In this section we will review briefly the topics that are immediately relevant to this thesis. These include the following topics on parallel computers, interconnection architectures and the algorithms we have studied in this investigation:

- Parallel Computers
- Interconnection Architecture
 - Interconnection Technologies
 - Simple Interconnection Architecture
 - Complex Opto-electronic Architecture
- Implementation of some algorithms on the OMULT Architecture
- Fundamental algorithms relevant to the work
 - Matrix multiplication
 - Prefix sum
 - Computational Geometry

2.2 Parallel computers

A parallel computer is one that consists of a collection of processing units, or processors, that cooperate to solve a problem by working simultaneously on different parts of that problem [1]. A stream of instructions indicates, to the computer, what to do at each step

and a stream of data is affected by these instructions. With the recent advances in VLSI technology, it is possible to have a computer that contains many processors that communicate each other using either shared memory or an interconnection network. Figure 2.1 shows a static interconnection network using the single instruction multiple data (SIMD) model. Each rectangle in the figure 2.1 represents a processor and every edge connecting two rectangles represents a bi-directional link connecting the pair of processors corresponding to the two rectangles.

In order to use such a parallel computer to solve a problem, the problem is decomposed into a number of sub-problems, each of which may be solved somewhat independently of one another. The sub-problems are assigned to different processors of the parallel machine. These processors work simultaneously to solve the sub-problem assigned to it and interact with other processors as needed to exchange data/ results. Finally the results of individual processors are combined to produce an answer to the original problem.

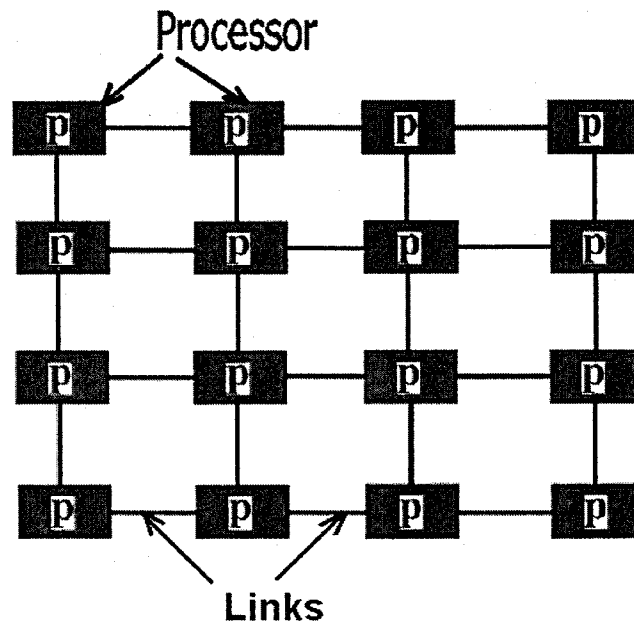


FIGURE 2.1: 2D MESH INTERCONNECTION NETWORK

2.3 Interconnection Architecture

The rules used to connect two processors are critically important. *Interconnection architecture* defines the way processors are connected to each other in a parallel computer. The interconnection architecture defines how any processor may transfer data from/to any other processor(s) in the network as needed to implement an algorithm. In recent years there has been a lot of investigation in designing interconnection architecture for parallel computer system [10, 34, 20]. The shortest path between two processors P_i and P_j is the smallest number of links that have to be used from one processor to communicate with the other. An important aspect, used to evaluate an interconnection network is the diameter, defined as the largest possible value of the shortest path between any two processors in an interconnection network[30].

2.4 Interconnection technologies

In this section we will discuss the two basic interconnection technologies that we use - *electrical interconnections* and *optical interconnections*. Traditional interconnection technology uses copper to get an electrical connection. Using VLSI, connecting two processors using copper is very straightforward and is traditionally done by embedding all processors on the same layout where the link is in the form of a via connection between metal layers [4]. Copper based interconnection system uses electronics signals for communication and the system works well when interconnection distances less than 1 millimeter [4, 23]. It is well known that the VLSI layout of many popular communication architectures is complicated and the size of such VLSI arrays as well as the physical

length of the longest link increase exponentially with the number of processors in the network [21]. This is a major problem since a longer copper line dramatically increases the delay on the line.

An optical network is a *digital communication* system that uses light waves as the medium for transmission of data. Optical technology has made significant contributions to the state of the art for long distance communication. Advantages of optical technology include high reliability, low interference, security benefits and (most important) very high bandwidth [4].

In recent years, the idea of replacing electrical connections by optical connection in an interconnection network has been drawing much attention among researchers [22]. Besides the advantages of high bandwidth and low wire density, optical communication supports high data rate communication with lower power requirements than electric interconnects [10], high bandwidth and high reliability. A major advantage is that, except for the speed of light limitation, there is no capacitive delay for a longer physical connection realized using optical technology [4]. For this reason, researchers are investigating the use of optical links rather than electronic links when the interconnection distance is more than a few millimeters [7, 16].

2.5 Simple Interconnection Architecture

In this sub-section we will present some examples of basic interconnection networks and configurations.

2.5.1 Linear Array

The simplest way to connect nodes is in the form of a linear array. If we have N processors P_0, P_1, \dots, P_{N-1} to be interconnected in a linear array, processor P_i has a link to processor P_{i-1} and a link to processor P_{i+1} [1], for all i , $0 < i < N$. The processor P_0 and P_{N-1} have only one link to processor P_1 and P_{N-2} respectively. We show a linear array in Figure 2.2. Linear arrays have a simple architecture and have been used in some SIMD machines [5]. Linear arrays have a poor diameter.



FIGURE 2.2: LINEAR ARRAY INTERCONNECTION

2.5.2 Tree Interconnections

In this network the processors form the binary tree. The binary tree has a diameter as the $O(\log N)$ where N is the number of nodes in the network assuming that each node has the same number of incoming and outgoing links [5]. The important advantage of the topology is that it is suitable for many parallel algorithms. The main drawback of the binary tree network is poor *bisection width* and *arc connectivity* [30].

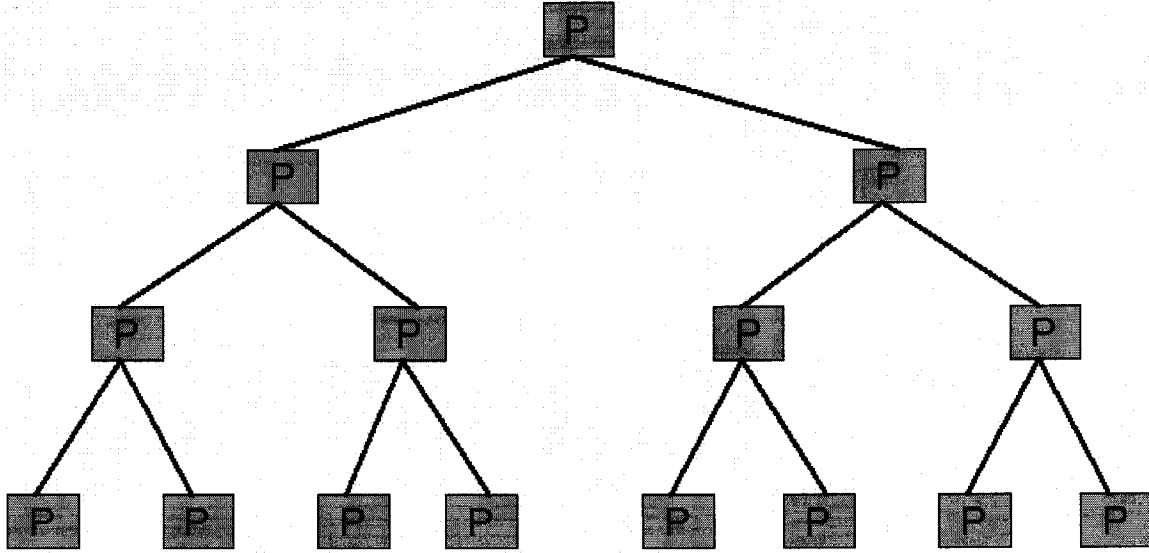


FIGURE 2.3: TREE INTERCONNECTION

In general, such a tree has d levels, numbered 0 (for leaf nodes) to $d - 1$ (for the node at the root of the tree), and $N = 2^d - 1$ nodes each of which is a processor. Figure 2.3 [15] shows a tree uses $d = 4$ and $N = 15$. The root node at level $d - 1$ has no parent and leaf nodes at level 0 have no children. Each node at level i , is connected to its root node at level $i + 1$ and to its two leaf nodes at level $i - 1$.

2.5.3 2D-Mesh Interconnection

Mesh (figure 2.4 [5]) is one of the most popular interconnection network. This is the interconnection we used in Figure 2.1. In an $n \times n$ 2D mesh, the processors are arranged in n rows and n columns in a square grid. The processor in row j and column k is denoted by $P(j, k)$, where $0 \leq j, k \leq m - 1$. There are links connecting $P(j, k)$ to processors $P(j + 1, k)$, $P(j - 1, k)$, $P(j, k + 1)$ and $P(j, k - 1)$. There are $4(n - 2)$ processors on the four outer boundaries of the grid, each of which has three processors with which it is connected by a link. There are four corner processors that have only two other processors with which it is

connected by a link. Each of the remaining $(n-2)^2$ processors have four other processors with which it is connected by a link.

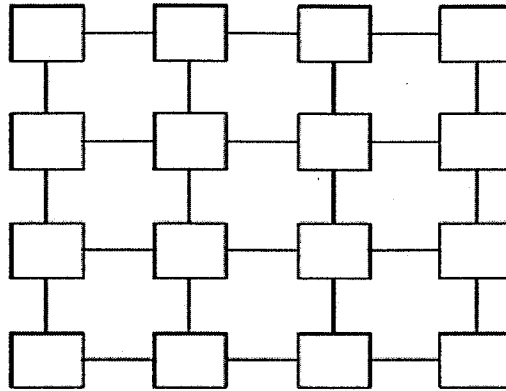


FIGURE 2.4: 2D-MESH INTERCONNECTION

There are many others proposed interconnection architectures [10] that we will not review due to lack of space.

2.6 Interconnection architecture based on opto-electronic technology

2.6.1 OTIS Mesh architecture

We have already mentioned the advantages of optical communication when the physical distance between two processors exceed a certain limit. One of the recent architectures that take advantage of the optical technology is the Optical Transpose Interconnect System (OTIS) [25], [12], [40]. The OTIS is an example of a hybrid architecture in which the processors are partitioned into groups of the same size.

Figure 2.5 shows a 16-processor OTIS-Mesh where squares with solid lines denote a processor and a rectangle with dashed lines denote a group of 4 processors. The groups are arranged in the form of a two-dimensional array and specifying the group the processor belongs to and the address of the processor within the group identifies each processor. Wang and Sahni [35]-[39] used the form (G, P) where G identifies the group and P identifies the processor within the group. Since specifying its row index and its column index may identify a group, G is specified by the pair (g_x, g_y) . Similarly a processor within a given group may be uniquely specified by its row index p_x and column index p_y . Thus the address of a processor is the 4-tuple (g_x, g_y, p_x, p_y) . The interconnections of the network are as follows:

- (1) Electronic links are used to connect processors within the group so that processor $P(g_x, g_y, p_x \pm 1, p_y)$ has an electronic link with processor $P(g_x, g_y, p_x, p_y \pm 1)$.
- (2) Optical interconnections are used in different group so that (G_i, P_j) connect to (P_i, G_j) where $i \neq j$ so that processor $P(g_x, g_y, p_x \pm 1, p_y)$ is connected to processor $P(p_x, p_y, g_x, g_y \pm 1)$ by an optical link.

Krishnamoorthy et al. [17] have shown that, when the number of processors in each group is equal to the total number of groups, then the bandwidth and power efficiency are maximized, and system area and volume are minimized. The OTIS-Mesh opto-electronic computer is a class of OTIS computers on which various fundamental algorithms have been conveniently mapped [29], [33], [35] – [39].

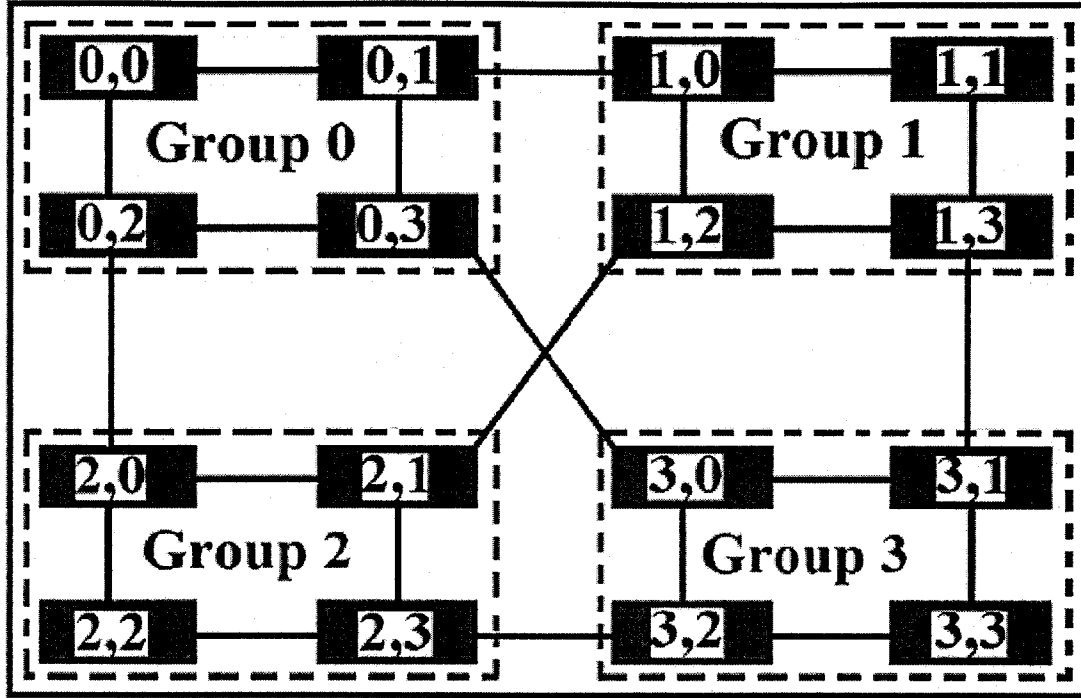


FIGURE 2.5: OTIS-MESH NETWORK

Wang and Sahni [37] have developed algorithms such as the convex hull, the smallest enclosing box, two-set dominance, maximal points, all-nearest neighbor and closest-pair on OTIS-Mesh. Rajasekaran and Sahni [29] have developed algorithms for packet routing, sorting and selection on OTIS-Mesh. The investigations of these algorithms using this network show that all those problems can be solved in $O(\sqrt{N})$ time even with N^2 inputs, whereas most of those problems takes $O(N)$ time for multi-mesh [6] and $O(N^2)$ time for mesh.

2.6.2 OMULT Architecture

Sinha and Bandyopadhyay have recently proposed a new hybrid interconnection system called the Optical Multi-Trees (OMULT) which uses binary trees as the basic building blocks[31]. The *OMULT network of order n* consists of n^2 complete binary trees T_{ij} 's, $1 \leq i, j \leq n$, arranged in the form of an $n \times n$ array, each tree having n leaf nodes and $n-1$ internal nodes. Each node in a tree is a processor. The nodes within each tree are interconnected by usual electronic links, while the leaf nodes of different trees are interconnected by optical links according to some rules discussed in this section. The nodes in each tree T_{ij} , $1 \leq i, j \leq n$, are given distinct integers from 1 to $2n-1$ in reverse level order, i.e., the leaf nodes in each tree are numbered from 1 to n , in order from left to right, and the internal nodes are also numbered from left to right in successive lower levels (the root node being at the lowest level - level 0). Thus, the root node in each tree is given the node number $2n-1$, and the node k in a tree T_{ij} will be referred to by the processor node $P(i, j, k)$, $1 \leq i, j \leq n$, $1 \leq k \leq 2n-1$. The total number of nodes in the system is $N = n^2 (2n-1) = 2n^3 - n^2$. The optical links interconnected only the leaf nodes in different trees as follows:

- 1) Processor $P(i, j, k)$, $1 \leq i, j \leq n$, $1 \leq k \leq n$, $j \neq k$, is connected to processor $P(i, k, j)$ by a bi-directional (full-duplex) optical link (*horizontal inter-tree link*),
- 2) Processor $P(i, j, k)$, $1 \leq i, j \leq n$, $1 \leq k \leq n$, $i \neq k$, is connected to processor $P(k, j, i)$ by a bi-directional optical link (*vertical inter-tree link*),

- 3) For $i = k$ and/or $j = k$, processor $P(i, j, k)$, $1 \leq i, j \leq n$, $1 \leq k \leq n$, is connected to processor $P(i, j, 2n-1)$ by a bi-directional optical link.

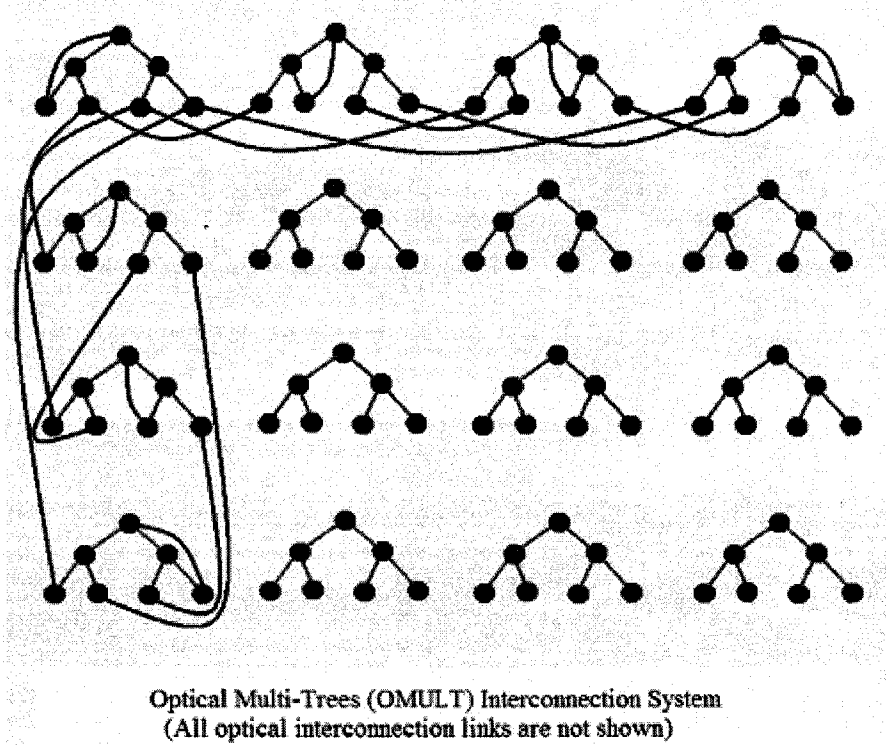


FIGURE 2.6: OMULT NETWORK

An example for the OMULT topology for $n = 4$ is shown in Fig. 2.6. These rules for interconnecting the leaf nodes in different trees have some similarities with those for interconnecting boundary/corner nodes in different meshes of the Multi-Mesh topology [6]. It follows from the above interconnection scheme that each of the leaf nodes $P(i, j, k)$, $1 \leq i, j \leq n$, $1 \leq k \leq n$, excepting those of the form $P(i, i, i)$, has 2 optical links and one electronic link connected to it, while each of the leaf nodes $P(i, i, i)$, $1 \leq i \leq n$, has only one optical link and one electronic link connected to it. All non-root internal nodes in a tree have 3 electronic links each, while each root node of the form $P(i, i, 2n-1)$,

has one optical link and two electronic links. All remaining root nodes has two optical links and two electronic links. The total number of optical links in the network is equal to $2(n^3 - n) + n + n + 2(n^2 - n) = 2(n^3 + n^2 - n)$.

In the algorithms described in [31] and in our algorithms that we will describe later, we assume that a processor $P(i, j, k)$ in an OMULT architecture has three registers $A(i, j, k)$, $B(i, j, k)$ and $C(i, j, k)$.

2.7 Some Important Properties of the OMULT Network

Property 1: The diameter of the OMULT topology is $6 \log n + 2 = O(\log N)$, with $6 \log n$ electronic links and 2 optical links.

Property 2: The node connectivity of the OMULT topology is two.

Property 3: The diameter of the OMULT topology under single node/link failure is equal to $8 \log n + 6$.

These properties are discussed in [31].

2.8 Algorithms for the OMULT Architecture

Sinha and Bandyopadhyay [31] have considered the following basic operations on the OMULT architecture: a) Data Broadcast, b) Row/Column Group-Broadcast, c) Complete Group-Broadcast, d) Summation/Average/Maximum/Minimum, e) Prefix Computation, f) Matrix Transpose, g) Matrix Multiplication and h) Sorting algorithms. These algorithms can be very efficiently solved on the OMULT topology with lesser time

complexities than those on the OTIS mesh. For example, summation/ average/ maximum/ minimum of n^3 elements and prefix computation of n^2 elements can be computed on this network in $O(\log n)$ time, two $n \times n$ matrices can be multiplied in $O(\log n)$ time, and n^2 elements can be sorted in $O(\log^2 n)$ time. These time complexities may be compared to $O(n)$ time for finding summation/maximum/minimum and prefix computation of n^4 elements, $O(n^4)$ time for multiplying two $n^4 \times n^4$ matrices, and $O(n)$ time for sorting n^4 elements on the OTIS mesh with n^4 processors. In this section, we briefly review the mapping of two fundamental algorithms on the OMULT topology as proposed in [31] since we will refer to them later on in this thesis.

2.8.1 Row/column group broadcast

Only the leaf nodes in a tree are used for performing input/output operations. Assuming that there have n data elements d_1, d_2, \dots, d_n in the n leaf nodes in a tree T_{ij} , for different applications, we may need to broadcast all these n data elements to the respective leaf nodes in all trees in the same column (row). We can be performed this operation in two phases. In phase 1, the group of data elements is distributed over all trees in the same row i (column j). In phase 2, broadcast them to all trees in column j (row i). We may perform the whole process in $O(\log n)$ time as follows.

Without loss of generality, we assume that, initially, the n data elements d_1, d_2, \dots, d_n are stored in the leaf nodes of the tree T_{11} , data d_k , $1 \leq k \leq n$, is stored in the processor node $P(1, 1, k)$, and we want to broadcast these to all trees in the first column. First, using

the horizontal optical links, we move data element d_k to $P(1, k, 1)$ for all k , $1 \leq k \leq n$. We then broadcast data d_k to all nodes in the tree T_{1k} , $1 \leq k \leq n$, which needs $2 \log n$ steps along electronic links. This completes phase 1 of distributing the n data elements to all trees in the same row (row 1).

In phase 2, for broadcasting the data elements to all trees in the same column (column 1), we take the data d_j , $1 \leq j \leq n$, now stored in the processors $P(1, j, k)$, $1 \leq k \leq n$, and send d_j to the processors $P(k, j, 1)$ using the vertical optical links. Using the horizontal optical links once again, we move data d_j to the processors $P(1, j, k)$. If the data elements were initially stored in any other tree T_{ij} , we may use the same method to broadcast them to all trees in i^{th} row and j^{th} column. It follows from above that the total number of communication steps needed for the whole process is $2 \log n$ (electronic links) + 3 (optical links).

2.8.2 Data Broadcast

To broadcast a data d residing in a processor $P(i, j, k)$, we need to broadcast d to all nodes in the tree T_{ij} in a maximum of $2 \log n$ communication steps using all electronic links. Once all the leaf nodes in T_{ij} receive the data d , we may use the optical links in the horizontal direction from each of these nodes to send the data to a leaf node in each of the remaining $(n - 1)$ trees in the i^{th} row. This requires just one communication step through optical links. We then broadcast this data reaching a leaf node in a tree T_{im} , $1 \leq m \leq n$, to all nodes in T_{im} in $2 \log n$ communication steps through the electronic links.

The leaf nodes in all the trees in i^{th} row will then send the data to one leaf node of each of the remaining $(n^2 - n)$ trees of the network using the vertical optical links. Finally, we can broadcast this data to all nodes in each of these $(n^2 - n)$ trees in another $2 \log n$ steps through electronic links. Thus, the broadcast operation needs a total of $6 \log n$ (electronic) + 2 (optical) communication steps ($O(\log N)$ time).

2.8.3 Complete Group-Broadcast

In a complete group broadcast, the group of n data elements d_1, d_2, \dots, d_n initially stored in the n leaf nodes in a tree, say T_{11} , may be communicated to the respective leaf nodes in all trees by modifying the phase 2 of the above operation of row/column group-broadcast. First, we distribute the data elements over all trees in row 1, as in phase 1 of the row/column group-broadcast. In the second phase, we send the data $d_j, 1 \leq j \leq n$, now stored in the processors $P(1, j, k), 1 \leq k \leq n$, to the processors $P(k, j, 1)$ using the vertical optical links. We now broadcast this data element $d_j, 1 \leq j \leq n$, to all the leaf nodes in the trees $T_{kj}, 1 \leq k \leq n$. This needs $2 \log n$ steps along electronic links. Using the bi-directional horizontal optical links, we move the data element d_j now stored in $P(k, j, 1), 1 \leq j, k, 1 \leq n$, to $P(k, 1, j)$. Thus, we send data d_j to the j^{th} leaf node of all the trees, and hence, we execute the broadcast operation correctly. The total number of steps needed for the whole operation is $4 \log n$ (electronic links) + 3 (optical links).

2.8.4 Sorting

Suppose there have a set of n elements $\{a_1, a_2, \dots, a_n\}$ stored in the leaf nodes of the tree T_{1l} . We assume that there is a total ordering defined on this set of elements. These elements can be sorted in the following algorithm[31] by finding the rank of each element. Thus, if the rank of an element a_i is r , then after sorting, the element a_i will be placed in the processor $P(l, l, r)$. For the sake of explaining the basic idea, we assume that the given elements are all distinct.

Algorithm SORT :

Step 1 : We broadcast the given set of n elements to all trees T_{jl} , $1 \leq j \leq n$, in the first column.

Step 2 : For all j , $1 \leq j \leq n$, we broadcast a_j to all the leaf nodes in the tree T_{jl} , compare it with all other elements and find its rank. We store the rank value in the root of the tree T_{jl} . Also, in this process of computing the rank, we eventually move the element a_j to the root of the tree T_{jl} .

Step 3 : If the rank of a_j is r , then we move the element a_j to the node $P(j, l, r)$.

Step 4 : $P(r, l, j) \leftarrow P(j, l, r);$ /* using the vertical optical link */

Step 5 : $P(r, l, l) \leftarrow P(r, l, j);$

Step 6 : $P(l, l, r) \leftarrow P(r, l, l);$ /* using the vertical optical link */

It was shown that the overall sorting can be completed in $O(\log n)$ time[31].

2.8.5 Matrix Multiplication

We give below the well-known algorithm for matrix multiplication of a matrix A of size $M \times K$ and a matrix B of size $K \times N$ giving a matrix C of size $M \times N$. We will use a_{pq} (b_{pq} and c_{pq}) to denote the element in row p and column q of A (respectively B and C) and will use the notation $A = [a_{pq}]$ (respectively $B = [b_{pq}]$ and $C = [c_{pq}]$). The following pseudo-code gives a simple sequential algorithm for matrix multiplication.

```

for all  $i, 1 \leq i \leq M$ 
  for all  $j, 1 \leq j \leq N$ 
    {  $c_{ij} = 0$ ;
      for all  $k, 1 \leq k \leq K$ 
         $c_{ij} = c_{ij} + a_{ik} * b_{kj}$ ;
    }

```

We may easily convert the above sequential algorithm to multiply two $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ to form the product matrix $C = [c_{ij}]$ to a parallel algorithm for an OMULT network of order n .

The above sequential algorithm is obviously equivalent to the following pseudo-code for a parallel algorithm:

```

for all  $i, 1 \leq i \leq n$  do in parallel
  for all  $j, 1 \leq j \leq n$  do in parallel
    {  $c_{ij} = 0$ ;
      for all  $k, 1 \leq k \leq n$  do in parallel
        {
           $c_{ij} = c_{ij} + a_{ik} * b_{kj}$ ;
        }
    }

```

Initially we store the matrix elements in the leaf nodes of the diagonal trees T_{ii} , $1 \leq i \leq n$, such that the elements $a_{i1}, a_{i2}, \dots, a_{in}$ of row i of the matrix A are stored in $A(i, i, 1)$,

$A(i, i, 2), \dots, A(i, i, n)$, respectively, and the elements $b_{1i}, b_{2i}, \dots, b_{ni}$ of column i of the matrix B are stored in $B(i, i, 1), B(i, i, 2), \dots, B(i, i, n)$, respectively. Algorithm M [31] for the OMULT topology ensures that the elements of the row i of the product matrix C remain in the leaf nodes of the diagonal tree T_{ii} , $1 \leq i \leq n$ when the algorithm terminates.

Algorithm M:

Step 1: For all i , $1 \leq i \leq n$, broadcast the elements of row i of the matrix A to the A -registers of the leaf nodes of all trees T_{ij} , $1 \leq j \leq n$ in the same row (using the above row/column group-broadcast algorithm).

Step 2: For all j , $1 \leq j \leq n$, broadcast the elements of column j of the matrix B to the B -registers of the leaf nodes of all trees T_{ij} , $1 \leq i \leq n$, in the same column.

/ The leaf nodes of the tree T_{ij} , $1 \leq i, j \leq n$, now contains the elements of row i of A and column j of B */*

Step 3: For all i, j, k , $1 \leq i, j, k \leq n$, do in parallel

Begin */* product element c_{ij} is now computed in the tree T_{ij} */*

$$A(i, j, k) \leftarrow A(i, j, k) * B(i, j, k);$$

$$A(i, j, 2n-1) \leftarrow A(i, j, 1) + A(i, j, 2) + \dots + A(i, j, n);$$

$$A(i, j, i) \leftarrow A(i, j, 2n-1);$$

end;

/ c_{ij} values are moved to the diagonal tree T_{ii} */*

Step 4: For all i, j , $1 \leq i, j \leq n$, do in parallel

$A(i, i, j) \leftarrow A(i, j, i);$ /* using the horizontal optical links */

2.8.6 Prefix-sum

Prefix sum is very useful in scheduling and constraint satisfaction problems [1]. The problem description is given below:

- We are considering a series a_1, a_2, \dots, a_N

- We need to compute

➤ a_1

➤ $a_1 + a_2$

➤ $a_1 + a_2 + a_3$

➤

➤ $a_1 + a_2 + \dots + a_N$

- we can write the general form as follows:

- $a_1 + a_2 + \dots + a_i$, $1 \leq i \leq N$ where $N = n^3$.

2.8.7 Computational Geometry Problems

Computational Geometry (CG) deals with algorithms that create, modify or describe geometric objects using computers and is an important field in computer graphics, robotics etc. Important classes of computational geometry problems [1] include:

- (1) Inclusion problems such as
 - a. locating a point in a planar subdivision,
 - b. reporting which points among a given set are contained in a specified domain.
- (2) Intersection problems such as finding intersections of
 - a. line segments,
 - b. polygons,
 - c. circles,
 - d. rectangles,
 - e. half spaces
- (3) Proximity problems such as
 - a. determining the closest pair among a set of given points or among the vertices of a polygon,
 - b. computing the smallest distance from one set of points to another
- (4) Construction problems such as
 - a. identifying the convex hull of a polygon,
 - b. obtaining the smallest box that includes a set of points.

We now briefly describe some CG algorithms that we will use later on in this thesis.

2.8.7.1 Convex Hull

The convex hull problem [28] is to find a hull that surrounds and encloses a given set of points. To find the convex hull for a given set of points S on a plane ($|S| = n$), we need to identify the extreme points. We assume that no three points in S are collinear. Fig. 2.7(A) shows an example for the set of points S and Fig. 2.7(B) shows the corresponding convex hull of S . Corresponding to a point $p_i \in S$, let $p_{i0}, p_{i1}, \dots, p_{i,n-2}$ be the points in $S - \{p_i\}$, (i.e., $p_{ik} \neq p_i$ for $0 \leq k \leq n-2$), sorted by the polar angle made by the vector, $0 \leq k \leq n-2$. Then, by the results shown in [37], p_i is an extreme point of S if the

counterclockwise angle between any pair of consecutive vectors $\overrightarrow{P_i P_{ik}}$ and $\overrightarrow{P_i P_{i,(k+1) \bmod n - 1}}$ is more than π . For example, for $S = \{a, b, c\}$, if the counter-clockwise angle (polar angle) between the vectors \overrightarrow{ab} and \overrightarrow{ac} , as shown in Fig. 2.8, is greater than π , then point a is an extreme point.

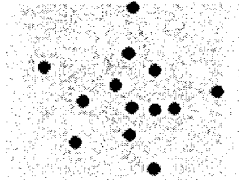


FIGURE 2.7(A): SET OF POINTS

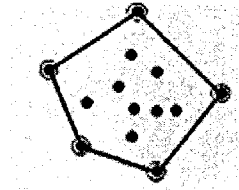


FIGURE 2.7(B): CONVEX HULL OF S

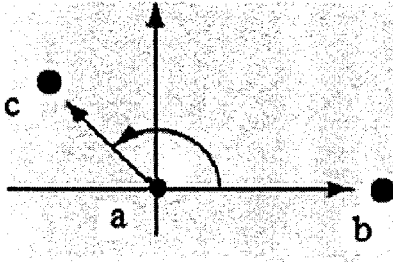


FIGURE 2.8: POLAR ANGLE

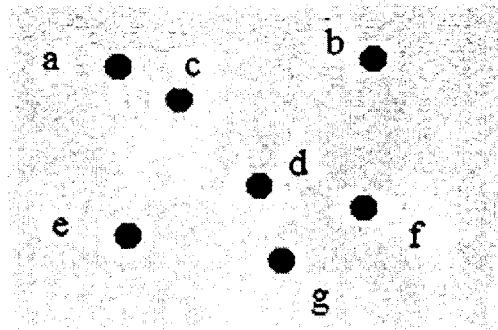
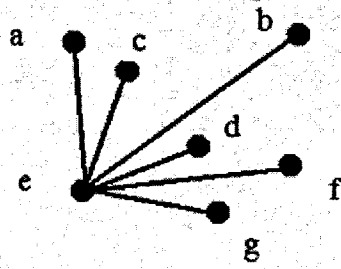


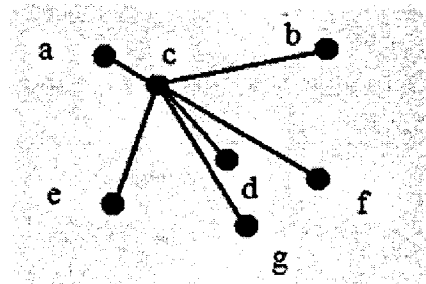
FIGURE 2.9(A): SET OF POINTS

Figure 2.9(A) shows a set of points $S = \{a, b, c, d, e, f, g\}$, for which an extreme point is illustrated in Figure 2.9(B). The counterclockwise angle between the vectors \overrightarrow{ea} and \overrightarrow{eg} is more than π , so e is an extreme point, whereas c is not an extreme point shown in

figure 2.9(c) because the counterclockwise angle between no two consecutive vectors (sorted by their polar angles) originating at the point c is more than π . In chapter 4, we have used the above property to find the convex hull for a set S of points on OMULT network and the algorithm proposed by [37] for solving the Convex Hull problem on the OTIS mesh is given below:



e is an extreme point



c is not an extreme point

FIGURE 2.9(B): E IS AN EXTREME POINT

FIGURE 2.9(C): C IS AN EXTREME POINT

Wang and Sahni [37] have proposed the following algorithm to find the Convex Hull using the OTIS mesh.

Algorithm Convex Hull:

Step 1: Perform an OTIS move of the points in group 0 .

Step 2: Processor 0 of group i , $0 \leq i \leq N$ broadcasts the point it received in step 1 to all processors in its group. All processors in a group now have the same point in their A registers.

Step 3: Perform an OTIS move on the points in the A registers. The data is received into B registers. Now, each group i processor has the point p_i in its A register and a point of $S - \{p_i\}$ in its B register.

Step 4: Each processor computes the polar angle of the vector defined by the points in its A and B registers.

Step 5: Each group sorts, into snake-like order, the angles computed by its processors.

Step 6: Each processor in a group checks the condition of computing the angle between the vectors defined by p_i , the point in its B register, and the point in the B register of the next processor in the snake like order.

Step 7: Processor 0 of each group is notified by group processors that conclude a point p_i is an extreme point.

Step 8: The points that pass the test of Theorem 1 [37] are transmitted to group 0 via an OTIS move.

Step 9: The extreme points accumulated by group 0 are sorted by polar angle order.

2.8.7.2 Smallest Enclosing Box (SEB)

Given a set S of coplanar points, the smallest enclosing box (SEB) problem [37] is to find the rectangle with the minimum area that encloses all the points in S . Freeman and Shapira showed [9] that the SEB of S has one side that is collinear with an edge of the convex hull of S and that the remaining three sides of the SEB pass through at least one convex hull vertex each. Figure 2.10 shows a rectangle of SEB that encloses all the

points. The algorithm proposed by Wang and Sahni [37] for solving the SEB problem on the OTIS mesh is given below:

Algorithm SEB:

Step 1: Compute the convex hull as in Section 2.8.7.1.

Step 2: Broadcast the hull vertices from group 0 to all remaining groups.

Step 3: Group i determines the i th hull edge (p_l, p_r) and broadcasts this to all processors within the group, for all i , $1 \leq i \leq n$.

Step 4: Each group i processor determines the distance h between its hull vertex q (if any) and the i th hull edge (p_l, p_r) as well as the distance w to the perpendicular bisector L of the i th hull edge. If q and p_l are on the same side of L , set $l = -w$ and $r = 0$; otherwise, set $l = 0$ and $r = w$.

Step 5: Processor 0 of each group i compute the maximum of the h 's and r 's in its group and the minimum of the l 's in its group. The area of the SEB that has one side collinear with (p_l, p_r) is $A_i = h_{\max} * (r_{\max} - l_{\min})$.

Step 6: Perform an OTIS move on the A_i 's. Now all A_i 's are in the group 0 processors.

Step 7: Processor 0 of group 0 determines the minimum A_i .

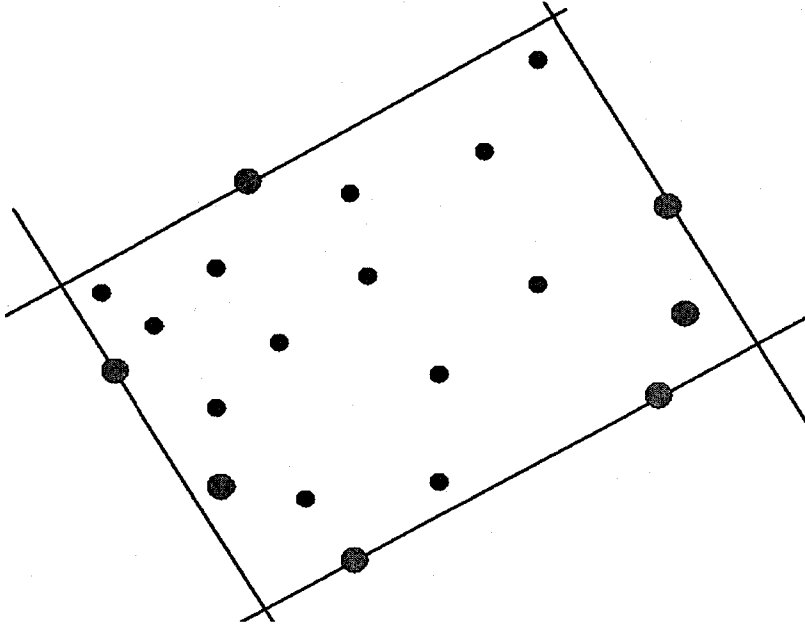


FIGURE 2.10: SMALLEST ENCLOSING BOX

2.8.7.3 Empirical Cumulative Distribution Function (ECDF)

The Empirical Cumulative Distribution Function (ECDF) problem [37] is to find the number of points dominated by each point $p_i \in S$. Any point *dominates* another point if and only if both coordinate values of the first point is greater than those of the second point. In other words, a point $p_i = (x_i, y_i)$, $p_i \in S$ dominates another point $p_j = (x_j, y_j)$, $p_j \in S$ if and only if $x_i \geq x_j$ and $y_i \geq y_j$ [37]. For example, Figure 2.11 shows four points p_1 , p_2 , p_3 and p_4 . Point p_4 dominates all other points whereas p_2 dominates only p_1 , but not p_3 or p_4 . An algorithm proposed by [37] for solving the ECDF problem on the OTIS mesh is given below:

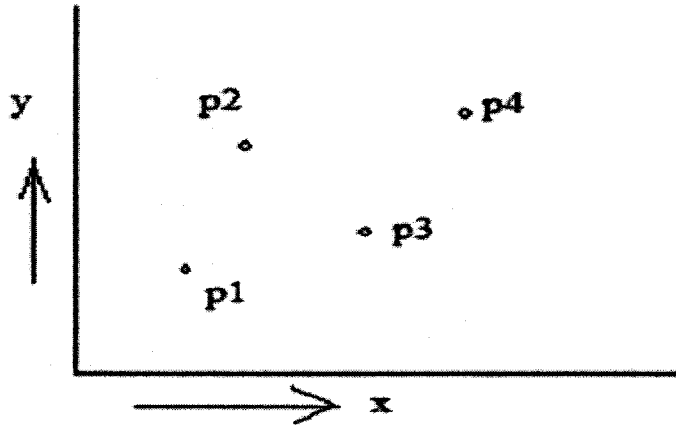


FIGURE 2.11: SET OF POINTS

Algorithm ECDF:

Step 1: Perform an OTIS move on the points initially in group 0. Now processor $(i, 0)$ has point p_i .

Step 2: Processor $(i, 0)$ broadcasts point p_i to the remaining processors in its group. Each processor saves its point in register A as well as register B .

Step 3: Perform an OTIS move on the register B data. Now, processor (i, j) has point p_i in register A and point p_j in register B .

Step 4: Each processor sets its C register to 1 if its A register point dominates its B register point; the C register is set to 0 otherwise.

Step 5: Processor $(i, 0)$ computes the sum of the C values in its group.

Step 6: Perform an OTIS move on the sums computed in Step 5.

2.8.7.4 All-Nearest Neighbor

All nearest neighbor problem is to find the minimum distance between any two points. We are considering a set of points (figure 2.11) $S = \{p_1, p_2, \dots, p_n\}$, the All-Nearest Neighbor problem deals with finding a point $q \in S$ corresponding to every point $p \in S$, such that $q \neq p$ and q is nearest to p among all other points in S [14]. Without loss of generality, we assume that all points of S are distinct. The algorithm proposed by [37] for solving the ANN problem on the OTIS mesh is given below:

Algorithm ANN:

Step 1: Perform an OTIS move on the points initially in group 0. Now processor $(i, 0)$ has point p_i .

Step 2: Processor $(i, 0)$ broadcasts point p_i to the remaining processors in its group. Each processor saves its point in register A as well as register B .

Step 3: Perform an OTIS move on the register B data. Now, processor (i, j) has point p_i in register A and point p_j in register B .

Step 4: Each processor sets its C register to the distance between the points in its A and B registers (if the points are the same, the C register is set to).

Step 5: Processor $(i, 0)$ computes the minimum of the C values in its group and thereby identifies the nearest neighbor of the point in the group's A registers.

Step 6: Perform an OTIS move on the nearest neighbors computed in the $(i, 0)$ processors.

Wang and Sahni [37] have efficiently mapped the above computational geometry problems for finding

- i) the convex hull,
- ii) the smallest enclosing box (SEB),
- iii) the empirical cumulative distribution function (ECDF) and
- iv) all-nearest neighbor(ANN) problem on an OTIS-Mesh in $O(\sqrt{n})$ time for n data inputs.

Chapter 3: Algorithms in Matrix Multiplication and Prefix Sum on OMULT Network

3.1 Introduction

In chapter 2 we already explained the basic algorithms for matrix multiplication, prefix-sum on the OMULT architecture [1]. In this chapter we will present an algorithm for:

- 1) matrix multiplication of two matrices of size $n^2 \times n^2$,
- 2) matrix multiplication of two matrices of any arbitrary size,
- 3) computing the prefix-sum of n^2 data elements,
- 4) computing the prefix-sum of n^3 data elements.

After describing each algorithm we will analyze it and, whenever possible, compare our results with those for other topologies.

3.2 Matrix Multiplication: Two square Matrices of size $n^2 \times n^2$

In this sub-section we're considering the multiplication of two square matrices X and Y each of size $n^2 \times n^2$. Figures 3.1 and 3.2 shows two such matrices with $n = 4$.

$x_{1,2}$	$x_{1,2}$	$x_{1,3}$	$x_{1,16}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,16}$
.....
.....
.....
$x_{15,1}$	$x_{15,2}$	$x_{15,3}$	$x_{15,16}$
$x_{16,1}$	$x_{16,2}$	$x_{16,3}$	$x_{16,16}$

FIGURE 3.1: MATRIX X WITH SIZE $n^2 \times n^2$, WITH $n = 4$

$y_{1,2}$	$y_{1,2}$	$y_{1,3}$	$y_{1,16}$
$y_{2,1}$	$y_{2,2}$	$y_{2,3}$	$y_{2,16}$
.....
.....
.....
$y_{15,1}$	$y_{15,2}$	$y_{15,3}$	$y_{15,16}$
$y_{16,1}$	$y_{16,2}$	$y_{16,3}$	$y_{16,16}$

FIGURE 3.2: MATRIX Y WITH SIZE $N^2 \times N^2$, WITH $N = 4$

To describe our algorithm, it is convenient to visualize X as consisting of rows and columns of sub-matrices with n sub-matrices in each row and each column. For this reason we divide the X and Y matrices into blocks of sub-matrices, each of size $n \times n$, so that each of the matrices X and Y are divided into n^2 blocks of sub-matrices. In Figure 3.3 we show how we divide a 16×16 matrix X . This division helps us to ensure that each sub-matrix of X and Y have exactly n rows and n columns so the multiplication of such sub-matrices may be carried out using the algorithm for matrix multiplication given in chapter 2.

In general, when we consider X (Y) as consisting of n rows and n columns of sub-matrices, we will denote the sub-matrix in row r and column s by X^{rs} (Y^{rs}). Each sub-

matrix X^{rs} (Y^{rs}) contains n rows and n columns of data elements. We will use X^{rs}_{ij} (Y^{rs}_{ij}) to denote the element in row i and column j of X^{rs} (Y^{rs}). As in chapter 2, we assume that the OMULT network contains n^2 trees with n leaf nodes in each tree and that the leaf nodes of each tree can handle input/output operations. It may be readily verified that $X^{rs}_{ij} = x_{pq}$ where $p = (r-1) \times n + i$ and $q = (s-1) \times n + j$.

$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$	$x_{1,13}$ $x_{1,14}$ $x_{1,15}$ $x_{1,16}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$	$x_{2,13}$ $x_{2,14}$ $x_{2,15}$ $x_{2,16}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$	$x_{3,13}$ $x_{3,14}$ $x_{3,15}$ $x_{3,16}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$	$x_{4,13}$ $x_{4,14}$ $x_{4,15}$ $x_{4,16}$
...
...
...
...
$x_{13,1}$ $x_{13,2}$ $x_{13,3}$ $x_{13,4}$	$x_{13,13}$ $x_{13,14}$ $x_{13,15}$ $x_{13,16}$
$x_{14,1}$ $x_{14,2}$ $x_{14,3}$ $x_{14,4}$	$x_{14,13}$ $x_{14,14}$ $x_{14,15}$ $x_{14,16}$
$x_{15,1}$ $x_{15,2}$ $x_{15,3}$ $x_{15,4}$	$x_{15,13}$ $x_{15,14}$ $x_{15,15}$ $x_{15,16}$
$x_{16,1}$ $x_{16,2}$ $x_{16,3}$ $x_{16,4}$	$x_{16,13}$ $x_{16,14}$ $x_{16,15}$ $x_{16,16}$

FIGURE 3.3: BLOCKS OF SUB-MATRICES OF MATRIX X

To describe our algorithm, we start with the pseudo-code for the well-known sequential algorithm, we presented in chapter 2 for multiplying two matrices of size $n^2 \times n^2$.

for all i , $1 \leq i \leq n^2$

```

for all  $j, 1 \leq j \leq n^2$ 
{  $z_{ij} = 0$ ;
  for all  $k, 1 \leq k \leq n^2$ 
     $z_{ij} = z_{ij} + x_{ik} * y_{kj}$ ;
}

```

This is obviously equivalent to the following pseudo-code:

```

for all  $i_1, 1 \leq i_1 \leq n$ 
for all  $i_2, 1 \leq i_2 \leq n$ 
{  $i = (i_1 - 1) n + i_2$ ;
  for all  $j_1, 1 \leq j_1 \leq n$ 
  for all  $j_2, 1 \leq j_2 \leq n$ 
  {  $j = (j_1 - 1) n + j_2$ ;
    {  $z_{ij} = 0$ ;
      for all  $k_1, 1 \leq k_1 \leq n$ 
      for all  $k_2, 1 \leq k_2 \leq n$ 
      {
         $k = (k_1 - 1) n + k_2$ ;
         $z_{ij} = z_{ij} + x_{ik} * y_{kj}$ ;
      }
    }
  }
}

```

This is equivalent to the following pseudocode:

```

for all  $r, 1 \leq r \leq n$ 
for all  $s, 1 \leq s \leq n$ 
for all  $t, 1 \leq t \leq n$ 
{
  for all  $i, 1 \leq i \leq n$ 
  for all  $j, 1 \leq j \leq n$ 
  {  $z_{ij} = 0$ ;
    for all  $k, 1 \leq k \leq n$ 
    {
       $z_{ij} = z_{ij} + x_{ik} * y_{kj}$ ;
    }
  }
}

```

To multiply the sub-matrix X^{ik} by Y^{kj} , we can implement the following sequential algorithm by the OMULT network contains n^2 trees with n leaf nodes using the OMULT algorithm for matrix multiplication described in chapter 2.

```

for all  $i, 1 \leq i \leq n$  do in parallel
  for all  $j, 1 \leq j \leq n$  do in parallel
    {  $z_{ij} = 0;$ 
      for all  $k, 1 \leq k \leq n$  do in parallel
        {
           $z_{ij} = z_{ij} + x_{ik} * y_{kj};$ 
        }
    }

```

In other words, the three innermost for loops may be replaced as follows:

```

for all  $r, 1 \leq r \leq n$ 
for all  $s, 1 \leq s \leq n$ 
for all  $t, 1 \leq t \leq n$ 
{
  multiply the sub-matrix  $X^{rk}$  by  $Y^{st}$  using the OMULT algorithm for matrix multiplication
}

```

We recall from chapter 2 that each processor $P(i, j, k)$ in the OMULT system has three registers A, B and C which we denoted by $A(i, j, k), B(i, j, k)$ and $C(i, j, k)$. We will use the A -register and the B -register for data movement. In our algorithm, we carry out all input/output operations only using the trees of processors $T_{i,p}, 1 \leq i \leq n$.

We slightly modify the algorithm for multiplying two matrices A and B on the OMULT network described in chapter 2 as follows:

Modified Algorithm M:

Step 1 : For all $i, 1 \leq i \leq n$, broadcast the elements of row i of the matrix A to the A -registers of the leaf nodes of all trees $T_{ij}, 1 \leq j \leq n$ in the same row (using the row/column group-broadcast algorithm described in chapter 2).

Step 2 : For all $j, 1 \leq j \leq n$, broadcast the elements of column j of the matrix B to the B -registers of the leaf nodes of all trees $T_{ij}, 1 \leq i \leq n$, in the same column.

/ The leaf nodes of the tree $T_{ij}, 1 \leq i, j \leq n$, now contains the elements of row i of A and column j of B */*

Step 3 : For all $i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

Begin */* product element c_{ij} is now computed in the tree T_{ij} */*

$$A(i, j, k) \leftarrow A(i, j, k) * B(i, j, k);$$

$$A(i, j, 2n-1) \leftarrow A(i, j, 1) + A(i, j, 2) + \dots + A(i, j, n);$$

$$C(i, j, 2n-1) \leftarrow C(i, j, 2n-1) + A(i, j, 2n-1);$$

End

Time Complexity of Modified Algorithm M:

Steps 1 and 2 require $2 \log n + 3$ time units each. Step 3 needs $\log n + 2$ time units. Hence, the overall time required $5 \log n + 8$ time units.

We describe below the algorithm A for multiplying X and Y matrices on the OMULT topology.

Algorithm A for Matrix Multiplication:

Step 1: Repeat steps 2 – 9 for all $r, 1 \leq r \leq n$

Step 2: Repeat steps 3 – 9 for all $s, 1 \leq s \leq n$

Step 3: Initialize all registers in the OMULT network to 0

Step 4: Repeat steps 5 – 6 for all $t, 1 \leq t \leq n$

Step 5: Load the OMULT array with X^{rt} and Y^{ts}

Step 6: Compute the product $X^{rt} \times Y^{ts}$ using the modified algorithm M described above, generating a partial product in the C -registers in the root processors of all the trees of the OMULT network.

Step 7: For all $i, j, 1 \leq i, j \leq n$ do in parallel

$$A(i, j, 2n - 1) \leftarrow C(i, j, 2n - 1)$$

$$A(i, j, i) \leftarrow A(i, j, 2n - 1)$$

Step 8: For all $i, j, 1 \leq i, j \leq n$ do in parallel

$$A(i, i, j) \leftarrow A(i, j, i)$$

Step 9: Output the values of Z^{rs} stored in the processors lying on the diagonal of the OMULT tree.

Time complexity for Algorithm A:

We are ignoring the input and output time in this analysis so that we ignore the time needed in steps 5 and 9. As discussed earlier, step 6 requires $5 \log n + 8$ time units. We repeat steps 5 – 6 n times so that the steps 4 – 6 takes $n(5 \log n + 8)$ time. Step 3 needs 1 time unit and step 7 requires $1 + \log n$ time units. Step 8 needs one step of data movement – one unit time. So steps 3 – 8 requires $n(5 \log n + 8) + \log n + 2$ time units.

Steps 1 - 9 is repeated n times and steps 2 - 9 are repeated n times. So we require $n^2\{n(5\log n + 8) + \log n + 2\}$ time units for multiplying two matrices of size $n^2 \times n^2$. This may be compared to $O(n^6)$ time needed in a sequential implementation.

Theorem 1: Algorithm A takes $O(n^3 \log n)$ time for multiplying two matrices of size $n^2 \times n^2$.

3.3 Matrix Multiplication where the two operands may have any size

We will now discuss how to multiply two square matrices X and Y of any sizes by using the OMULT network. We're considering two matrices X and Y of size $M \times P$ and $P \times Q$ where $M, P, Q \geq n$. To simplify our algorithm, we first divide the matrices X and Y into sub-matrices of size $n \times n$ each. In the previous section we did the same when $M = P = Q = n^2$ as shown in figure 3.3 for $n = 4$. Let $X(Y)$ consist of m rows and p columns (respectively p rows and q columns) of sub-matrices. We will denote the sub-matrix in row r and column s of $X(Y)$ by X^{rs} (respectively Y^{rs}), $1 \leq r \leq m$, $1 \leq s \leq p$ (respectively $1 \leq r \leq p$, $1 \leq s \leq q$). Each sub-matrix X^{rs} (Y^{rs}) $1 \leq r < m$, $1 \leq s < p$ (respectively $1 \leq r < p$, $1 \leq s < q$) contains exactly n rows and n columns of data elements. We will use X^{rs}_{ij} (Y^{rs}_{ij}) to denote the element in row i and column j of X^{rs} (Y^{rs}). We note that the last sub-matrix in each row or column has a size less than or equal to n as shown in figure 3.4. For the sub-matrices X^{rm} (respectively Y^{ms}), $1 \leq r \leq m$ (respectively $1 \leq s \leq q$), we will fill the remaining rows and columns of sub-matrices X^{rm} (respectively Y^{ms}) with 0.

$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$	$x_{1,13}$ $x_{1,14}$ $x_{1,15}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$	$x_{2,13}$ $x_{2,14}$ $x_{2,15}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$	$x_{3,13}$ $x_{3,14}$ $x_{3,15}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$	$x_{4,13}$ $x_{4,14}$ $x_{4,15}$
...
...
...
...
$x_{13,1}$ $x_{13,2}$ $x_{13,3}$ $x_{13,4}$	$x_{13,13}$ $x_{13,14}$ $x_{13,15}$
$x_{14,1}$ $x_{14,2}$ $x_{14,3}$ $x_{14,4}$	$x_{14,13}$ $x_{14,14}$ $x_{14,15}$
$x_{15,1}$ $x_{15,2}$ $x_{15,3}$ $x_{15,4}$	$x_{15,13}$ $x_{15,14}$ $x_{15,15}$

FIGURE 3.4: DIVISION OF MATRICES INTO SUB-MATRICES

We recall, from chapter 2, that each processor $P(i, j, k)$ in the OMULT system has three registers A , B and C which we denoted by $A(i, j, k)$, $B(i, j, k)$ and $C(i, j, k)$. As before, we use the A -register and the B -register for data movement and we carry out all input/output operations only using the trees of processors $T_{i,p}$ $1 \leq i \leq n$. We describe below the algorithm B for multiplying X and Y matrices on the OMULT topology.

Algorithm B for Matrix Multiplication:

Step 1: Repeat steps 2 – 9 for all r , $1 \leq r \leq m$

Step 2: Repeat steps 3 – 9 for all s , $1 \leq s \leq q$

Step 3: Initialize all registers in the OMULT network to 0

Step 4: Repeat steps 5 – 6 for all t , $1 \leq t \leq p$

Step 5: Load the OMULT array with X^r and Y^s

Step 6: Compute the product $X^T \times Y^T$ using the modified algorithm **M** described above, generating a partial product in the C -registers in the root processors of all the trees of the OMULT network.

Step 7: For all $i, j, 1 \leq i, j \leq n$ do in parallel

$$A(i, j, 2n - 1) \leftarrow C(i, j, 2n - 1)$$

$$A(i, j, i) \leftarrow A(i, j, 2n - 1)$$

Step 8: For all $i, j, 1 \leq i, j \leq n$ do in parallel

$$A(i, i, j) \leftarrow A(i, j, i)$$

Step 9: Output the values of Z^s stored in the processors lying on the diagonal of the OMULT tree.

Time complexity:

In this algorithm we are ignoring the input and output time in this analysis so that we ignore the time needed in steps 5 and 9. Step 3 requires 1 time unit. Steps 5 – 6 require the same time as the modified algorithm M . Therefore steps 4 – 6 requires $p(5 \log n + 8)$ time units. Step 7 requires $1 + \log n$ time unit. Step 8 requires one step. Ignoring the input output time in step 9, the time for steps 3 - 9 is $1 + p(5 \log n + 8) + 1 + \log n$ time units. Steps 3 – 9 is repeated q times and steps 2 – 9 are repeated m times. So the required time for multiplying a matrix of size $M \times P$ by a matrix of size $P \times Q$ is $mq(p(5 \log n + 8) + \log n + 2)$ time units.

3.4 Prefix Sum for n^2 data elements

Before this computation, we assume we have n^2 data elements, a_1, a_2, \dots, a_n , stored in the n^2 leaf nodes of the T_{il} trees of the OMULT system, for all $i, 1 \leq i \leq n$. We recall, from chapter 2, that in order to compute the prefix-sum, each processor $P(i, j, k)$ has three registers $A(i, j, k)$, $B(i, j, k)$ and $C(i, j, k)$. Initially all the data elements a_1, a_2, \dots, a_n are stored in the A -registers of the leaf nodes of the trees in column 1 of the OMULT architecture. We assume that the A -registers of the leaf nodes $P(i, 1, k)$ of the tree T_{il} , $1 \leq i, k \leq n$, initially store the data values $a_{(i-1)n+k}$. When the process is over, the final prefix-sum values will also be stored in the leaf nodes of the trees T_{il} . The algorithm **PS1** for prefix-sum on the OMULT system is given below.

In order to carry out the process of computing the prefix sums, we describe the two algorithms- algorithm **I** and algorithm **II** modified from data broadcast algorithms we described in chapter 2.

Algorithm I:

The purpose of this algorithm is to send data in registers $A(i, 1, 1), A(i, 1, 2), \dots, A(i, 1, j)$ to registers $A(i, j, 1), A(i, j, 2), \dots, A(i, j, j)$, for all $i, j, 1 \leq i, j \leq n$.

Step 1: /* using horizontal optical links, move data from T_{il} to T_{ij} , $1 \leq i, j \leq n$ */

For all $i, j, k, 1 \leq i, j \leq n$, do in parallel

$A(i, j, 1) \leftarrow A(i, 1, j);$

Step 2: /* broadcast data within each tree T_{ij} */

$\forall i, k, 1 \leq i, k \leq n$, **do in parallel**

$A(i, j, k) \leftarrow A(i, j, 1);$

Step 3: /* using horizontal optical links, move data across trees */

For all $i, j, k, 1 \leq i, j \leq n, j \leq k \leq n$ do in parallel

$A(i, k, j) \leftarrow A(i, j, k);$

/*After step 3 we assume processor $P(i, j, 1)$ will have first data element, $P(i, j, 2)$ first two data element $P(i, j, 3)$ first three data element, \dots , $P(i, j, n)$ all n data element */

Time complexity: Step 1 and 3 takes 1 time unit each. Step 2 requires $2 \log n$ time unit.

Algorithm II:

This algorithm is used to broadcast data from register $A(i, n, i)$ in tree T_{in} to register $A(k, j, i)$ in all other trees T_{kj} for all $i, j, k, i < k \leq n, 1 \leq j \leq n$.

Step 1: /*broadcast content within trees T_{in} where $i \leq n$ */

$\forall i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

$A(i, n, k) \leftarrow A(i, n, i);$

Step 2: /* using horizontal optical links, move data across trees T_{ij} 's, $1 \leq i, j \leq n$ */

$\forall i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

$A(i, k, n) \leftarrow A(i, n, k);$

Step 3: /*broadcast content within trees T_{ij} */

$\forall i, j, k, 1 \leq i, j, k \leq n$, do in parallel

$A(i, j, k) \leftarrow A(i, j, n)$;

Step 4: /* using vertical optical links, move data across trees $T_{ij}, 1 \leq i, j \leq n$ */

$\forall i, j, 1 \leq i, j \leq n, i < k \leq n$ do in parallel

$A(k, j, i) \leftarrow A(i, j, k)$;

/*After step 4 the A-register $A(i, j, i)$, of tree T_{kj} will receive the data $T_{i,n}$, for all $i, 1 \leq i, \leq n, i < j \leq n$ using A-register */

Time Complexity: Step 1 and 3 each require $2 \log n$ time units. Step 2 and 4 need 1 time unit each.

Before the algorithm starts we assume that register C in all processors contain 0.

Algorithm PS1 for n^2 elements:

Step 1: Initialize registers A and B of all trees to 0.

Step2: Using the algorithm I, broadcast selected data in the A-registers of the tree T_{i1} to the A-registers of the tree T_{ij} , so that $A(i, 1, 1), A(i, 1, 2), \dots, A(i, 1, j)$ is sent to registers $A(i, j, 1), A(i, j, 2), \dots, A(i, j, j)$, for all $i, j, 1 \leq i, j \leq n$. After the broadcast, processors $P(i, 1, 1)$ will have data element a_1 in it's A-register, $P(i, 2, 1)$ will have data element a_1 , $P(i, 2, 2)$ will have data element a_2 , ..., $P(i, j, 1)$ will have data element a_1 , $P(i, j, 2)$ will have data element a_2 , ... $P(i, j, 2)$ will have data element a_j . All other processors will have a value of 0 in their A-registers.

Step 3: /*Compute the sum of each tree and store it in A -register $A(k, j, 2n-1)$. */

For all $i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

$$A(i, j, 2n-1) \leftarrow A(i, j, 1) + A(i, j, 2) + \dots + A(i, j, n);$$

Step 4: /*Move contents from $A(i, j, 2n-1)$ to $A(i, j, i)$ and copy to B -register*/

For all $i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

$$A(i, j, i) \leftarrow A(i, j, 2n-1);$$

$$B(i, j, i) \leftarrow A(i, j, i) + C(i, j, i);$$

Step 5: For all $i, j, k, i < k \leq n, 1 \leq j \leq n$ using the A -register, broadcast data from register $A(i, n, i)$ in tree T_{in} to register $A(k, j, i)$ in all other trees T_{kj} using the algorithm described above.

Step 6: For all $i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

$$A(i, j, 2n-1) \leftarrow A(i, j, 1) + A(i, j, 2) + \dots + A(i, j, n);$$

$$A(i, j, 1) \leftarrow A(i, j, 2n-1);$$

Step 7: For all $i, j, k, 1 \leq i, j, k \leq n$, **do in parallel**

$$A(i, j, 2n-1) \leftarrow A(i, j, 1) + B(i, j, i)$$

$$A(i, j, 1) \leftarrow A(i, j, 2n-1)$$

Step 8: /*Move data from $A(i, j, 1)$ to $A(i, 1, k)$ */

For all $i, j, 1 \leq i, j \leq n$, **do in parallel**

$$A(i, 1, j) \leftarrow A(i, j, 1);$$

Step 9: Output the prefix-sum stored at A -register in the processors lying on the T_{il} of the OMULT tree.

In the algorithm **PS1**, the contents of register C used in step 4 plays no role since the value was assumed to be 0. In the next algorithm, this register will play an important role.

Example: We consider algorithm **PS1** where we have to compute the prefix-sum for n^2 data elements.

1. Before the algorithm starts, the data elements are stored in the A -registers of the leaf nodes of the tree T_{II} (figure 3.5).

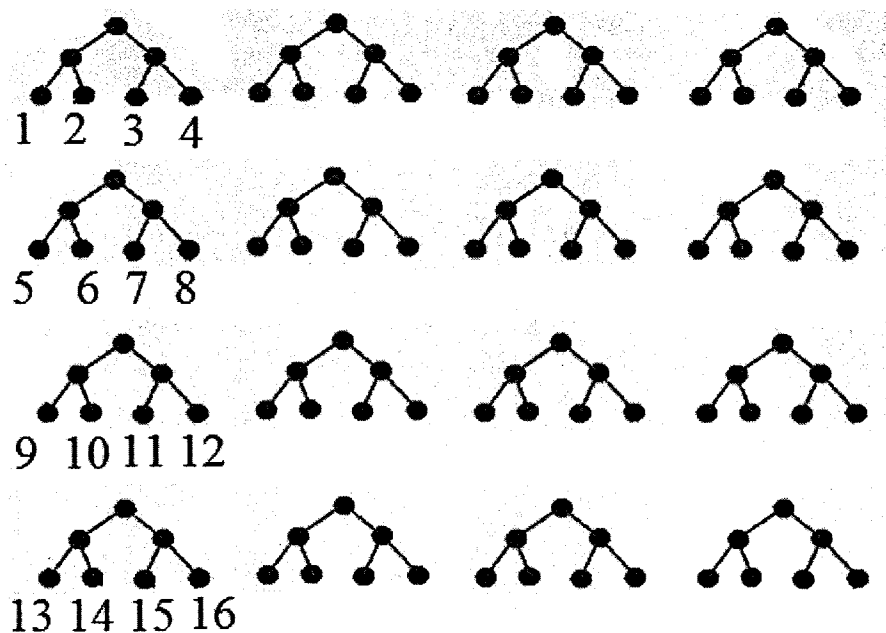


FIGURE 3.5: DATA ARE STORED AT THE LEAF NODES OF THE TREES

2. After step 2, data in the network are as shown in Figure 3.6

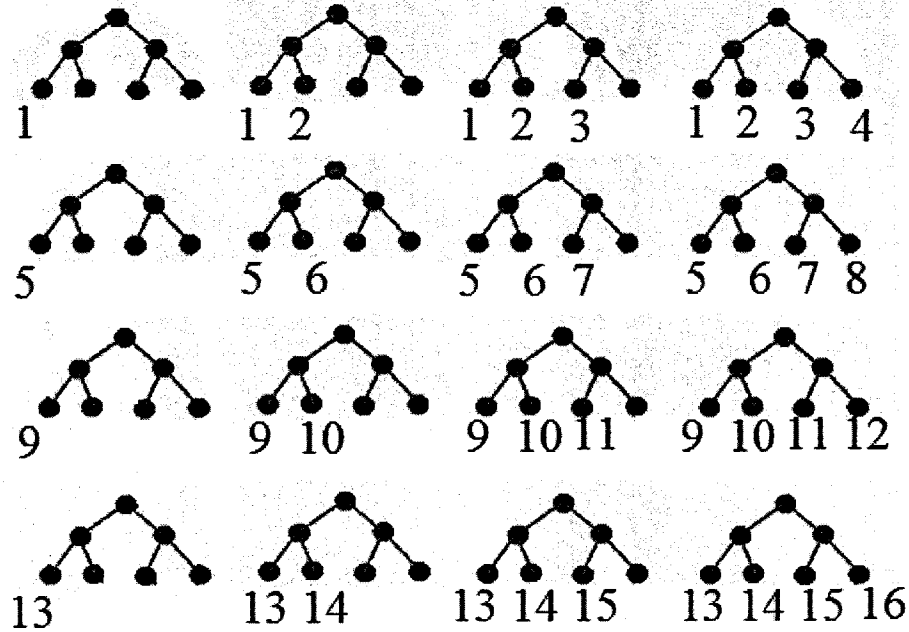


FIGURE 3.6: THE DISTRIBUTION OF DATA AFTER STEP 2

3. After step 3, content in the network are as shown in Figure 3.7

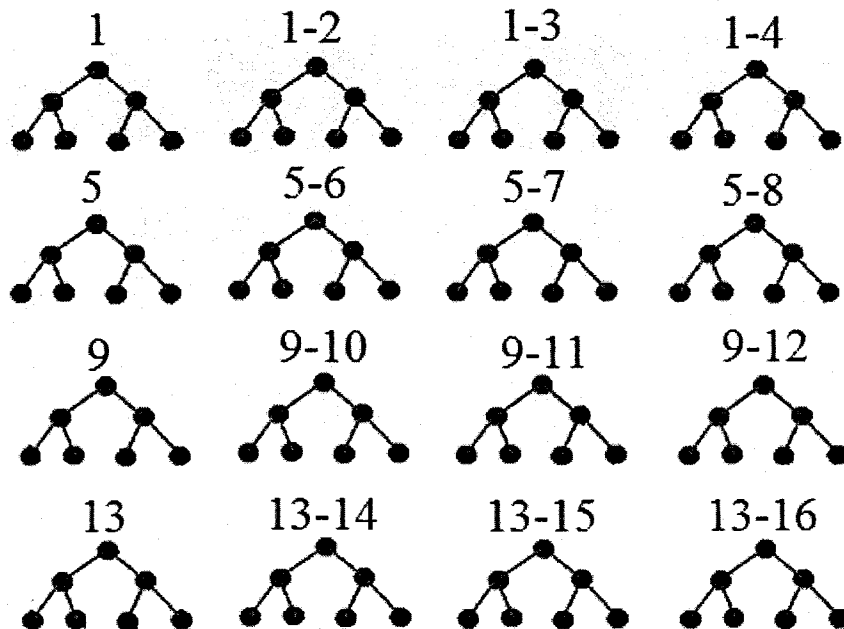


FIGURE 3.7: AFTER STEP 3.

4. After step 4, content in the network are as shown in Figure 3.8

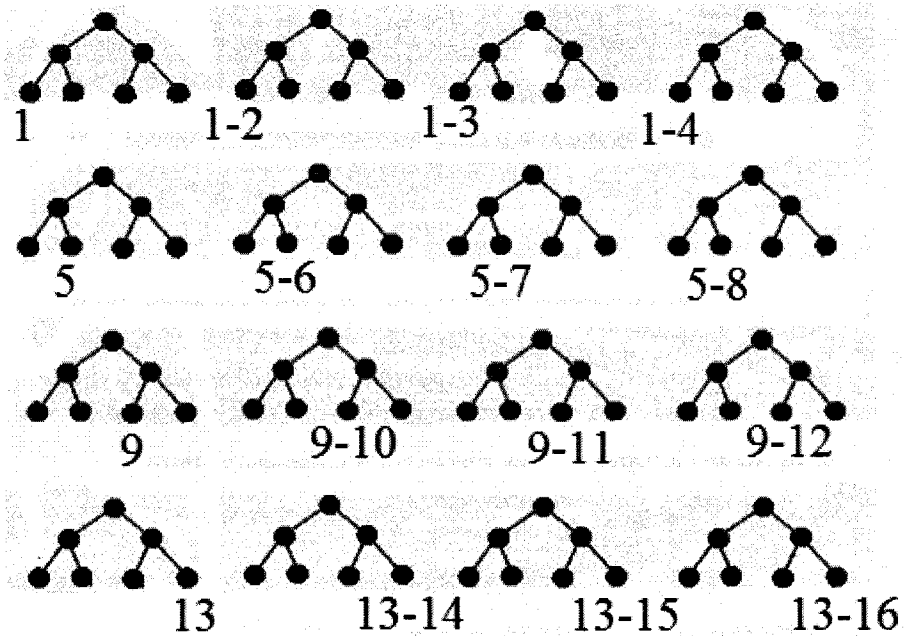
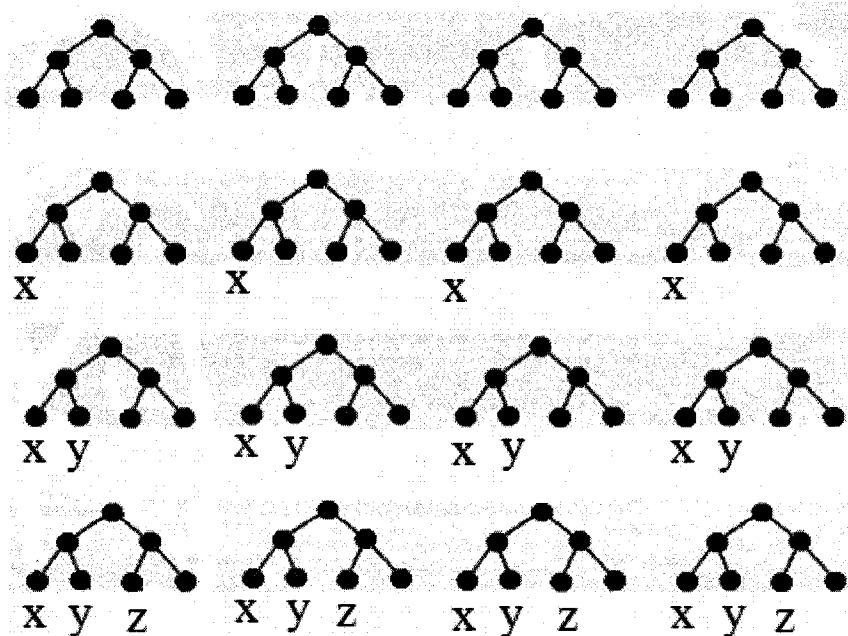


FIGURE 3.8: AFTER STEP 4

5. After step 5, content in the network are as shown in Figure 3.9



Where, $x=1-4$, $y=5-8$, $z=9-12$

FIGURE 3.9: AFTER STEP 5

6. After step 6, content in the network are as shown in Figure 3.10

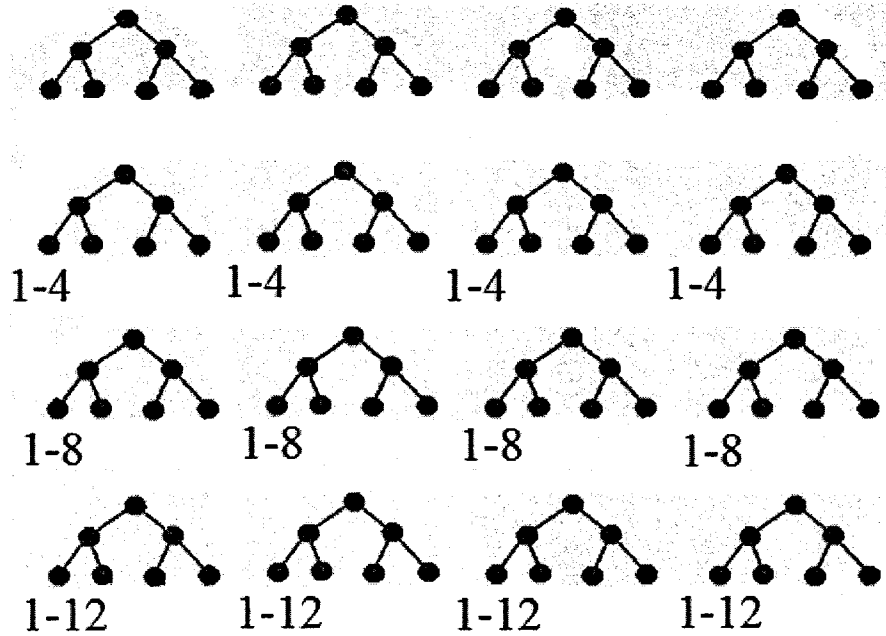


FIGURE 3.10: AFTER STEP 6

7. Now from the algorithm we will sum of the content, stored at the leaf nodes of the *A*-registers and *B*-registers. And we will get the final result as shown in figure 3.11.

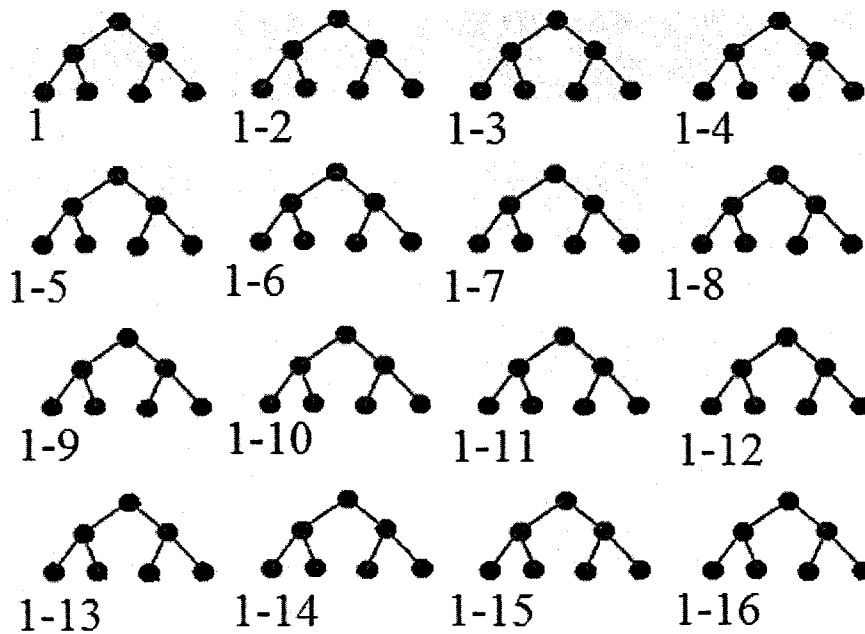


FIGURE 3.11: AFTER STEP 7

Time Complexity:

Step 1, step 8 and step 9 takes 1 unit time. Step 2 will need $2 \log n + 2$ as algorithm I need. Step 3 need $\log n$ time and step 4 takes 2 units time. Step 5 will need $4 \log n + 2$ time units as algorithm II need. Step 6 need $\log n + 1$ time unit. Step 7 required $2 \log n$ units time. So time required for computing n^2 data elements $10 \log n + 10$ time units.

Theorem 1: Algorithm *PSI* computes the prefix sum of n^2 data elements of $O(\log n)$ time.

3.5 Prefix Sum for n^3 data elements

We now describe the case where we have $X = n^3$ data elements. We first divide the data elements into blocks of n^2 elements. The prefix-sums for a block of n^2 elements may be carried out using the algorithm for prefix-sum given in section 3.4. The algorithm **PS2** for n^3 on the OMULT system is given below.

Algorithm PS2 for n^3 data elements:

Step 1: Initialize all the C registers in the OMULT network to 0.

Step 2: Repeat steps 3 – 6 for all r , $1 \leq r \leq n$.

Step 3: Using the algorithm for data broadcasting, broadcast the data in $C(n, 1, n)$ to all the C -registers in the entire network.

Step 4: Read the next block of n^2 elements.

Step 5: Compute the prefix-sum of the current block of n^2 elements using algorithm PS1, generating a partial sum in the A -registers in the root processors of all the trees of the OMULT network.

Step 6: Copy the sum $C(n, 1, n) \leftarrow A(n, 1, n)$ in tree T_{n1} .

Time Complexity:

Step 1-8 is repeated n times. Step 2, step 3 and 6 need 1 time unit each. Step 4 required $10 \log n + 6$ time units (algorithm PS1). Step 7 will need $4 \log n + 2$ time units as algorithm II need. Step 5 needs $2 \log n + 1$ time. Now step 2 to 7 needs $16 \log n + 12$. So time required for computing n^3 data elements $n(16 \log n + 12)$ time units.

Theorem 1: Algorithm PS2 computes the prefix sum of n^3 data element of $O(n(\log n))$ time.

Chapter 4: Algorithms in Computational Geometry on OMULT

4.1 Introduction

In this section, we show how some of the common algorithms in computational geometry can be mapped on the OMULT system. We have reviewed, in chapter 2, the algorithms we will study in this chapter.

4.2 Convex Hull

Based on the property explained in chapter 2, we describe below our algorithm for finding the convex hull for a set of points $S = \{p_1, p_2, \dots, p_n\}$, where we assume that no three of these points are collinear. We recall that each processor $P(i, j, k)$ in the OMULT system has three registers $A(i, j, k)$, $B(i, j, k)$, $C(i, j, k)$. In implementing this algorithm we use the A -register and the B -register for data movement operations. When the algorithm starts, the coordinates of all data points are stores in the A -registers of the leaf nodes of tree T_{II} .

Algorithm CH_OMULT:

Step 1: copy all the points from A -register to B -register of T_{II} tree

Step 2: /* using A -register move data from T_{II} to T_{ij} */

Step 2.1: /* using vertical optical links, move data from T_{II} to T_{il} , $1 \leq i \leq n$ */

$\forall i, 1 \leq i \leq n$, **do in parallel**

$A(i, 1, 1) \leftarrow A(1, 1, i);$

Step 2.2: /* broadcast data in the A-register of $P(i, 1, 1)$ within each tree T_{il} */

$\forall i, k, 1 \leq i, k \leq n$, **do in parallel**

$A(i, 1, k) \leftarrow A(i, 1, 1);$

Step 2.3: /* using horizontal optical links, move data from T_{il} to T_{ij} , $1 \leq i, j \leq n$ */

$\forall i, j, 1 \leq i, j \leq n$, **do in parallel**

$A(i, j, 1) \leftarrow A(i, 1, j);$

/* After step 2.3, all p_i values, $1 \leq i, j \leq n$, are stored at A-registers of the processors $P(i, j, 1)$ in the leaf nodes of each tree */

Step 3: /* using B-register move data from T_{il} to T_{ij} */

Step 3.1: /* using horizontal optical links, move data from T_{il} to T_{li} , $1 \leq i \leq n$ */

$\forall i, 1 \leq i \leq n$, **do in parallel**

$B(1, i, 1) \leftarrow B(1, 1, i);$

Step 3.2: /* broadcast data within each tree T_{li} */

$\forall i, k, 1 \leq i, k \leq n$, **do in parallel**

$B(1, i, k) \leftarrow B(1, i, 1);$

Step 3.3: /* using vertical optical links, move data from T_{li} to T_{ij} , $1 \leq i, j \leq n$ */

$\forall i, j, 1 \leq i, j \leq n$, **do in parallel**

$B(j, i, 1) \leftarrow B(1, i, j);$

/* After step 3.3, all p_j values, $1 \leq i, j \leq n$, are stored at B-registers of the processors $P(i, j, 1)$ in the leaf nodes of each tree */

Step 4 : $\forall i, j, 1 \leq i, j \leq n$, compute the vector $\overrightarrow{p_i p_j}$ in the tree T_{ij} , and store it in the register $A(i, j, 1)$ of the respective root node. (Note that for $i = j$, a 0 value will be stored for the vector).

Step 5 : Sort, in ascending order, the n vectors $\overrightarrow{p_i p_j}$ (including the zero vector) stored in the leaf nodes of the trees $T_{il}, 1 \leq i \leq n$ in the order of their polar angles by rank computation, in a manner similar to that described in the algorithm SORT in [31]. We will use $\overrightarrow{p_i q_j}$ to denote the vector in processor $P(i, 1, j)$ after the sorting is over. The A -register in each processor will still be used for data movements across different processors needed for this rank computation.

Step 6 : /* using A-register move points within tree in $T_{il}, 1 \leq i, */$

$\forall i, i \leq n$, **do in parallel**

$A(i, 1, 1) \leftarrow A(i, 1, n);$

Step 7 : /* copy the sorted list from A-register to B-register of T_{il} tree where $1 \leq i \leq n. */$

$\forall i, j, i \leq n, j \leq n$, **do in parallel**

$B(i, 1, j) \leftarrow A(i, 1, j);$

Step 8 : Again sort, in ascending sequence, the new list of n vectors $\overrightarrow{p_i q_n}, \overrightarrow{p_i q_1}, \overrightarrow{p_i q_2} \dots$

$\overrightarrow{p_i q_n}$ stored in the leaf nodes $P(i, 1, 1), P(i, 1, 2), P(i, 1, 3), \dots, P(i, 1, n)$ of the trees $T_{il}, 1 \leq i \leq n$ in the order of their polar angles by rank computation, in a manner similar to that described in the algorithm SORT in [31]. The A -register in

each processor will still be used for data movements across different processors needed for this rank computation.

Step 9: Compute the polar angle between vectors $\overrightarrow{p_i q_j}$ and $\overrightarrow{p_i q_{(j+1)}}$ in the tree leaf node of the T_{il} and store it in the respective leaf node.

Step 10 : If the polar angle computed in any processor $P(i, l, j)$ in T_{il} is more than π , then point p_i is an extreme point (since the polar angle between the sides $\overrightarrow{p_i q_j}$ and $\overrightarrow{p_i q_{(j+1)}}$ is more than π). Convey this information to the processor $P(i, l, 2n-1)$ in T_{il} by setting an appropriate tag bit (tag = 1, if p_i is an extreme point, and 0 otherwise).

Step 11 : Move the information regarding all such convex hull points to the leaf nodes of the processors $P(l, l, k)$, $1 \leq k \leq n$, in the tree T_{ll} using the vertical optical links.

Example: We consider a situation where we have 4 data points as shown in figure 4.1

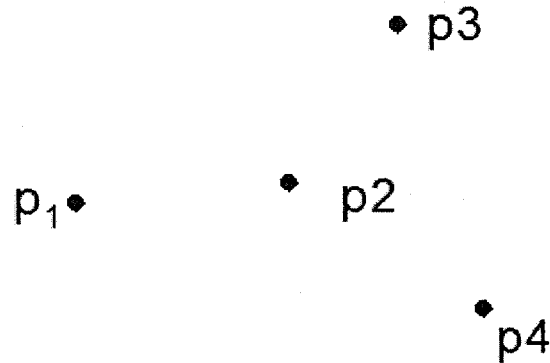


FIGURE 4.1: SET OF POINTS

1. Before the algorithm starts, the coordinates of these four points are stored in the A -registers of the leaf nodes of the tree T_{11} (figure 4.2).

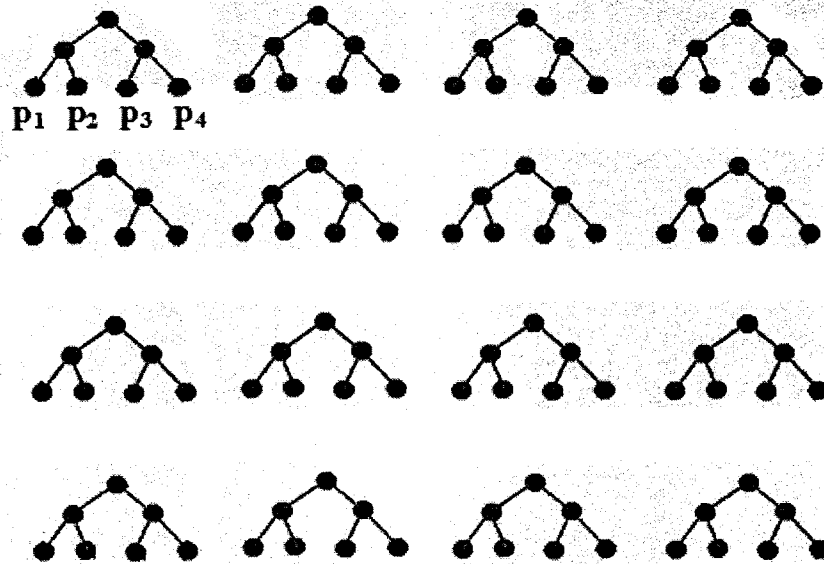


FIGURE 4.2: ALL THE POINTS ARE STORED AT THE LEAF NODE OF THE TREE T_{11}

2. After step 2.3 the points in the network are as shown in Figure 4.3

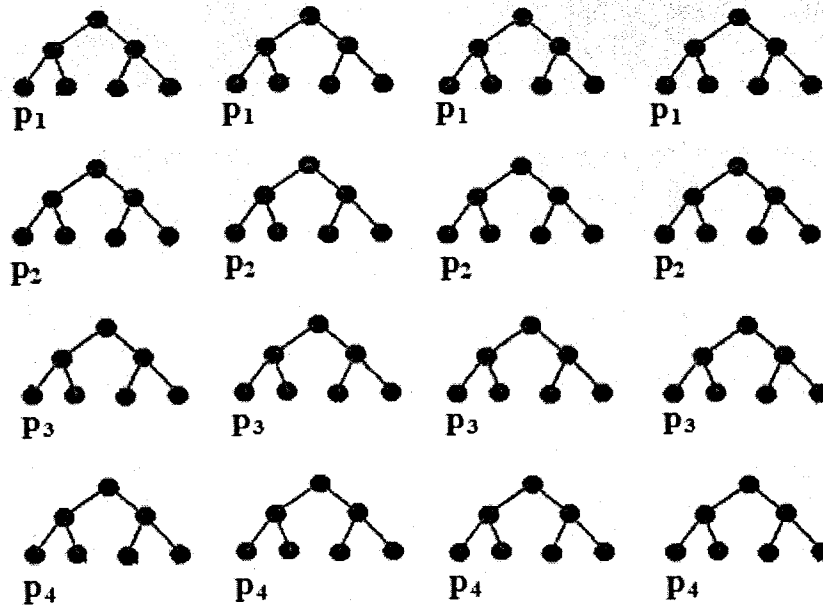


FIGURE 4.3: AFTER STEP 2.3

3. After step 3.3 the points in the network are as shown in Figure 4.4

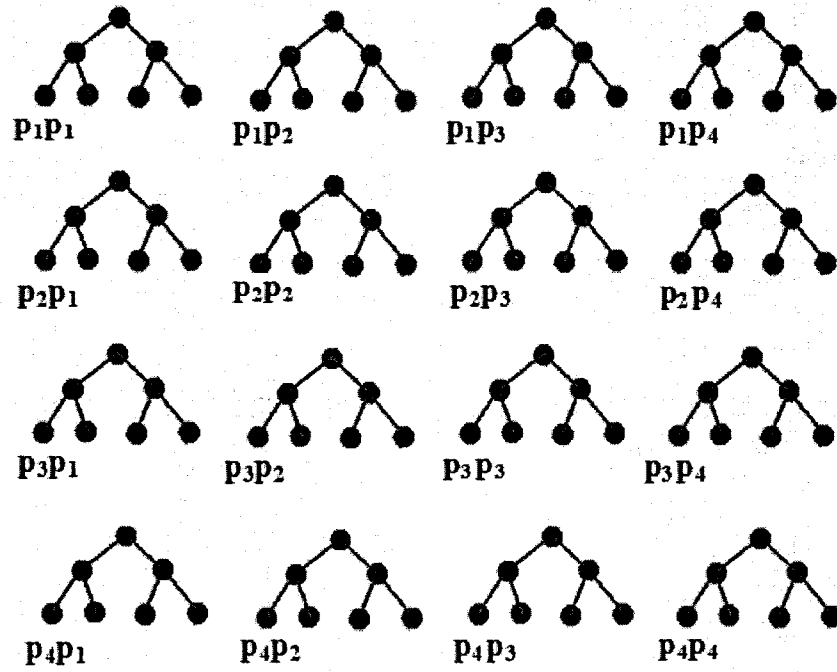


FIGURE 4.4: AFTER STEP 3.3

4. After step 4 the data in the network are as shown in Figure 4.5

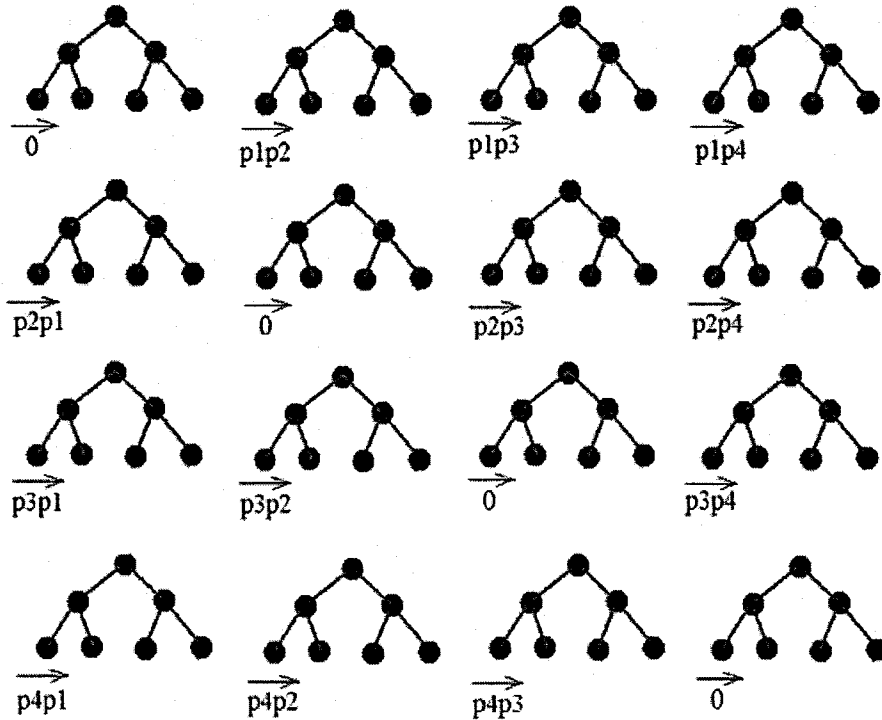


FIGURE 4.5: AFTER STEP 4

5. After step 5 the data in the network are as shown in Figure 4.6

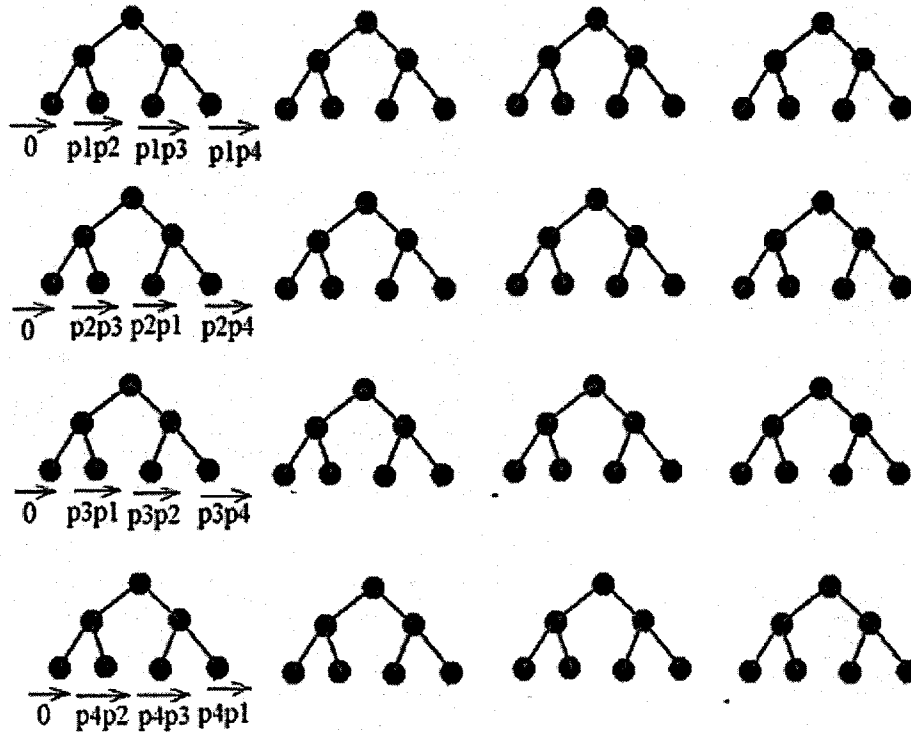


FIGURE 4.6: AFTER STEP 5

6. After step 6 the data in the network are as shown in Figure 5.7

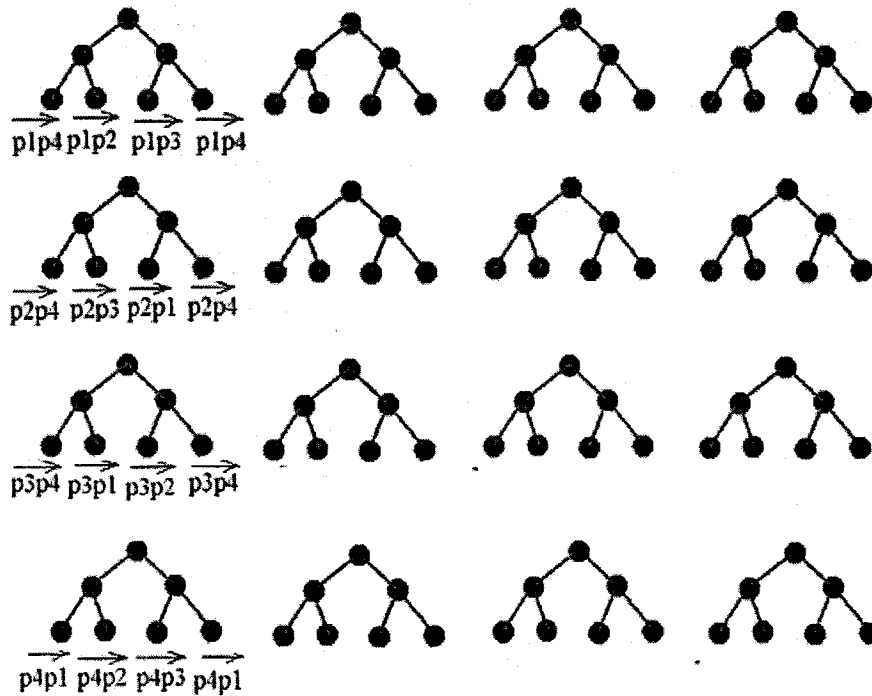


FIGURE 4.7: AFTER STEP 6

7. After step 8, the data points are as shown in Figure 4.8. We note that the values of the vectors are now in the A -registers and in step 7 we have saved one copy in B -register those are not shown in figure for the next steps.

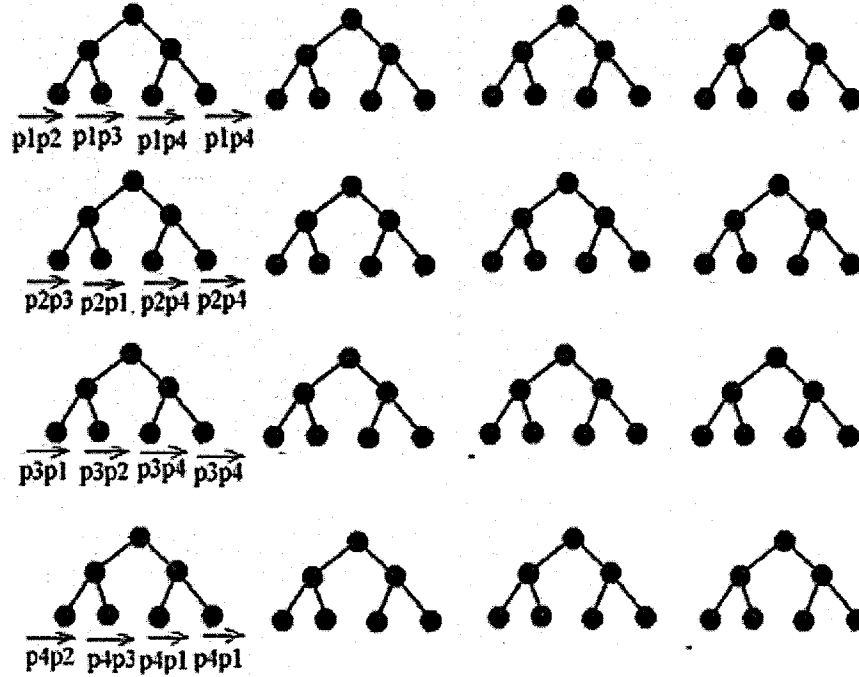


FIGURE 4.8: AFTER STEP 8

8. Now from the algorithm and with the above example, in step 7 we have saved one copy of vectors in B -register and after step 8 we have a sorted list of vectors in A -registers. So we can easily compute the extreme points using vectors $\overline{p_i q_j}$ and $\overline{p_i q_{(j+1)}}$ stored in the leaf nodes of the tree T_{il} by setting an appropriate tag bit (tag = 1, if p_i is an extreme point, and 0 otherwise).

Time Complexity:

Step 2 and 3 need $2 \log n + 2$ data transfer steps each. Step 1, step 7 and 9 need 1 time unit to copy data and step 4 require 1 time unit to compute vectors. Step 5 and 8 will require $3 \log n + 2$ time units [31]. Step 6 needs $2 \log n$ time units. Step 11 will need $\log n + 1$ time units each. Step 10 requires $\log n$ time units (assuming that setting the tag bit and related information about an extreme point requires one time unit). Hence, we have the following result.

Theorem 1: Algorithm CH_OMULT computes the convex hull of n points in $O(\log n)$ time.

4.3 Smallest Enclosing Box

In the smallest enclosing box (SEB) problem, first of all we need to find the convex hull vertices that we can get by invoking the algorithm in section 4.2. The algorithm to solve the SEB problem on the OMULT system is given below.

Algorithm SEB_OMULT:

Step 1 : /* compute the convex hull vertices and store the corresponding information in the leaf nodes of the tree T_{II} . We assume there are m convex hull vertices. We will refer to the successive vertices, which define the convex hull, as v_1, v_2, \dots, v_m . */

$\forall i, 1 \leq i \leq n$, **do in parallel**

if (point p_i is a convex hull vertex) then $P(1, 1, i) \leftarrow v_i$

else $P(1, 1, i) \leftarrow 0$;

Step 2: Sort, in descending order, the m convex hull vertices (including the zero vertices) stored in the leaf nodes of the tree T_{II} , in the order of their polar angles by rank computation, in a manner similar to that described in the algorithm SORT in [31]. The A -register in each processor will still be used for data movements across different processors needed for this rank computation.

Step 3 : Broadcast the information about the convex hull vertices v_1, v_2, \dots, v_m from T_{II} to all trees T_{il} , $1 \leq i \leq m$.

Step 4 : /* copy the sorted list from A -register to B -register of T_{il} tree where $1 \leq i \leq n$. */

$\forall i, j, 1 \leq i \leq n, 1 \leq j \leq n$, **do in parallel**

$B(i, 1, j) \leftarrow A(i, 1, j)$;

Step 5 : $\forall i, 1 \leq i \leq m$, compute the i^{th} hull edge $(v_i, v_{(i+1) \bmod m})$ in tree T_{il} and broadcast this edge to the leaf nodes $1, 2, \dots, n$ of the same tree T_{il} .

Step 6 : $\forall i, j, 1 \leq i, j \leq m$, using A -register and B -register, compute in the leaf node $P(i, 1, j)$, the height $d1$ between the hull vertex v_j and the hull edge $(v_i, v_{(i+1) \bmod m})$.

Step 7 : $\forall i, j, 1 \leq i, j \leq m$, compute, in the same leaf node $P(i, 1, j)$, the perpendicular bisector L of the hull edge $(v_i, v_{(i+1) \bmod m})$.

Step 8 : $\forall i, j, 1 \leq i, j \leq m$, compute, in the node $P(i, 1, j)$ the distance $d2$ from the vertex p_j to the perpendicular bisector L of the hull edge $(v_i, v_{(i+1) \bmod m})$.

Step 9 : $\forall i, j, 1 \leq i, j \leq m$, in the node $P(i, 1, j)$ check if v_i and v_j are on the same side of L . If so, set in the node $P(i, 1, j)$, $left \leftarrow d2$ and $right \leftarrow 0$; otherwise $P(i, 1, j)$ sets $left \leftarrow 0$ and $right \leftarrow d2$.

Step 10 : $\forall i, 1 \leq i \leq m$, determine the following:

- h_{max} , the maximum of all height values stored in processor $P(i, 1, j)$, $\forall j, 1 \leq j \leq m$,
- r_{max} , the maximum of all the right values stored in processor $P(i, 1, j)$, $\forall j, 1 \leq j \leq m$,
- l_{min} , the minimum of all left values stored in processor $P(i, 1, j)$, $\forall j, 1 \leq j \leq m$.

/* h_{max} , r_{max} and l_{min} are respectively the farthest, rightmost and leftmost points, from the vertex v_i and are saved in $P(i, 1, 2n - 1)$.*/

Step 11 : $\forall i, 1 \leq i \leq m$, compute in processor $P(i, 1, 2n-1)$ the area $A_i = h_{max} (r_{max} - l_{min})$.

Step 12 : $\forall i, 1 \leq i \leq n$, move the value of the area A_i from $P(i, 1, 2n-1)$ to the leaf node $P(1, 1, i)$. (This is done by first moving A_i to $P(i, 1, 1)$ in $\log n$ steps and then to $P(1, 1, i)$ in one step).

Step 13 : Find the minimum A_i of all area values in the leaf nodes of the tree T_{11} along with the relevant information regarding the bounding edges.

After step 9 is over, $P(1, 1, 2n-1)$ has the smallest enclosing box.

Time Complexity

Each of steps 1, 4 and 6-10 needs constant time. Step 2 will require $3 \log n + 2$ time units [31] and step 3 needs $2 \log n + 2$ time unit. Step 5 takes $2 \log n + 1$ time units and each of the remaining steps needs $O(\log n)$ time. Hence, we have the following result.

Theorem 2 : Algorithm SEB computes the smallest enclosing box of a given set of n points in $O(\log n)$ time.

4.4 Empirical Cumulative Distribution Function (ECDF)

We describe below the algorithm for finding the ECDF for a given set of points $S = \{p_1, p_2, \dots, p_n\}$. We assume that the coordinates of all the n points are initially stored in the leaf nodes $A(1, 1, 1), A(1, 1, 2), \dots, A(1, 1, n)$ of the tree T_{11} .

Algorithm ECDF_OMULT

Step 1 : Broadcast the coordinates of all the points from the leaf nodes of T_{11} to the leaf nodes of all trees T_{il} , $\forall l, 1 \leq i \leq n$, by using the algorithm for row/column group-broadcast in [31].

Step 2 : $\forall i, 1 \leq i \leq n$, do in parallel

$$B(i, 1, i) \leftarrow A(i, 1, i)$$

Step 3 : $\forall i, j, 1 \leq i, j \leq n$, do in parallel

$$A(i, j, 1) \leftarrow A(i, 1, j)$$

/* At this point, the A -registers in $P(i, j, 1)$ contain p_j */

Step 4 : /* broadcast data using B -register within each tree T_{il} */

$\forall i, k, 1 \leq i, k \leq n$, do in parallel

$$B(i, 1, k) \leftarrow B(i, 1, i);$$

Step 5 : $\forall i, k, 1 \leq i, k \leq n$, do in parallel

$$B(i, k, 1) \leftarrow B(i, 1, k);$$

/* At this point, the B -registers in $P(i, j, 1)$ contain p_i */

Step 6 : $\forall i, j, 1 \leq i, j \leq n$, **do in parallel**

Processor $P(i, j, 1)$ of T_{ij} tests if p_i dominates p_j ;

if (p_i dominates p_j) then $C(i, j, 1) \leftarrow 1$

else $C(i, j, 1) \leftarrow 0$;

Step 7 : $\forall i, j, 1 \leq i, j \leq n$, **do in parallel**

$C(i, 1, j) \leftarrow C(i, j, 1)$

Step 8 : $\forall i, 1 \leq i \leq n$, **do in parallel** /* in the tree T_{i1} */

compute the sum $C(i, 1, 2n-1) = C(i, 1, 1) + C(i, 1, 2) + \dots + C(i, 1, n)$;

Step 9 : move $C(i, 1, 2n-1)$ from $P(i, 1, 2n-1)$ to the node $P(1, 1, i)$ in T_{11} ;

/* leaf node $P(1, 1, i)$ in T_{11} stores the number of points dominated by p_i */

Time Complexity

Each of steps 2, 3 and 5-7 needs constant time. Step 1 require $2 \log n + 2$ time units and step 4 needs $2 \log n$ time units. Step 8 takes $\log n$ time units and the remaining step 9 needs $\log n + 1$ time. Hence, we have the following result.

Theorem 3 : Algorithm ECDF computes the empirical cumulative distribution function of a given set of n points in $O(\log n)$ time.

4.5 All-Nearest Neighbor

Assuming that the coordinates of all the n points are initially stored in the leaf nodes of the tree T_{11} , we describe below the algorithm for finding the all-nearest neighbor for the given set of points S .

Algorithm ANN_OMULT

Step 1 : Broadcast the coordinates of all the points from the leaf nodes of T_{11} to the leaf nodes of all trees T_{i1} , $\forall i$, $1 \leq i \leq n$, by using the algorithm for column group-broadcast in [31].

Step 2 : $\forall i$, $1 \leq i \leq n$, do in parallel

$$B(i, 1, i) \leftarrow A(i, 1, i)$$

Step 3 : $\forall i, j$, $1 \leq i, j \leq n$, do in parallel

$$A(i, j, 1) \leftarrow A(i, 1, j)$$

/* At this point, the A -registers in $P(i, j, 1)$ contain p_j */

Step 4 : /* broadcast data within each tree T_{i1} */

$\forall i, k$, $1 \leq i, k \leq n$, do in parallel

$$B(i, 1, k) \leftarrow B(i, 1, i);$$

Step 5 : $\forall i, k$, $1 \leq i, k \leq n$, do in parallel

$$B(i, k, 1) \leftarrow B(i, 1, k);$$

/* At this point, the B -registers in $P(i, j, 1)$ contain p_i */

Step 6 : /*Compute the distance between p_i and p_j of processor $P(i, j, 1)$ in T_{ij} and store in $C(i, j, 1)$ */

$\forall i, j, 1 \leq i, j \leq n$, **do in parallel**

$C(i, j, 1) \leftarrow$ distance between p_i and p_j

Step 7 : $\forall i, j, 1 \leq i, j \leq n$, **do in parallel**

$C(i, 1, j) \leftarrow C(i, j, 1)$

Step 8 : $\forall i, j, 1 \leq i, j \leq n$, find the minimum $D(i)$ of all $C(i, 1, j)$ and store it in $C(i, 1, 2n - 1)$

Step 9 : move $C(i, 1, 2n - 1)$ from $P(i, 1, 2n-1)$ to the node $P(1, 1, i)$ in T_{11} ;

/*leaf node $P(1, 1, i)$ in T_{11} stores the closest points and the corresponding distance from the point p_i */

Time Complexity

Each of steps 2, 3 and 5-7 needs constant time. Step 1 require $2 \log n + 2$ time units and step 4 needs $2 \log n$ time units. Step 8 takes $\log n$ time units and the remaining step 9 needs $\log n + 1$ time. Hence, we have the following result.

Theorem 4: Algorithm ANN computes the all-nearest neighbor of a given set of n points in $O(\log n)$ time.

Chapter 5: Network Simulation

5.1 Purpose of the Simulation

The purpose of our simulation was to see how difficult it is to use a simulator to

- define the OMULT architecture
- understand the behavior of a large complex OMULT network, and
- analyze network performance

The simulation for multiprocessor systems is complicated because of the difficulty of mapping the hardware with its high degree of parallelism within the frame work of existing simulation software. There are an increasing number of tools available to simulate the parallel and distribution systems and it was quite difficult to select the correct tools for simulating the application. We chose the SimJava simulation tool because it has an extremely powerful technique for evaluating performance of parallel and distribution systems. SimJava is a process based discrete event simulation package based on the Java programming language. By using the SimJava package we were able to represent the OMULT architecture in a realistic manner.

5.2 SimJava

A SimJava simulation is a collection of entities (`Sim_entity` class) each of which runs in its own thread [43]. These entities are connected together by ports (`Sim_port` class) and can communicate with each other by sending and receiving event objects (`Sim_event` class) through these ports. A static `Sim_system` class controls all the threads, advances the simulation time, and maintains the event queues [43]. The progress of the simulation is recorded through trace messages produced by the entities, and saved in a file.

5.3 Problem simulated

We considered the problem of simulating the algorithm **M** for matrix multiplication described in chapter 2 on the OMULT network. We recall that this algorithm multiplies a matrix A of size $n \times n$ and a matrix B of size $n \times n$ giving a matrix C of size $n \times n$. In chapter 2, we have described how we initially store the matrix elements in the leaf nodes of the diagonal trees T_{ii} , $1 \leq i \leq n$, such that the elements $a_{i1}, a_{i2}, \dots, a_{in}$ of row i of the matrix A are stored in $A(i, i, 1), A(i, i, 2), \dots, A(i, i, n)$, respectively, and the elements $b_{1i}, b_{2i}, \dots, b_{ni}$ of column i of the matrix B are stored in $B(i, i, 1), B(i, i, 2), \dots, B(i, i, n)$, respectively. As in chapter 2, we assume that the OMULT network contains n^2 trees with n leaf nodes in each tree and that the leaf nodes of each tree can handle input/output operations. Our simulation considered the case where $n = 4$.

5.4 Modeling of the system

In order to model the OMULT architecture we need to represent the followings-

- the nodes of the trees
- interconnection between nodes within a tree
- interconnection leaf nodes of different trees (Horizontal and Vertical links)
- communication between the nodes

We have modeled the OMULT system by using the SimJava in a following way-

5.4.1 The nodes of the trees

The nodes of the trees are represented by using the entities of a SimJava where each node is an individual potential entity. In order to do this we need to extend the standard Sim_entity class and override the body () method. The entities are added by using the Sim_system.add(entity) method. In this model the main entities are Nodes and the ports connect them to each other. They communicate with each other by sending and receiving information to corresponding Node.

Entities of a Tree:

As an example a simulation layout in a tree T11 are given in Figure 5.1:

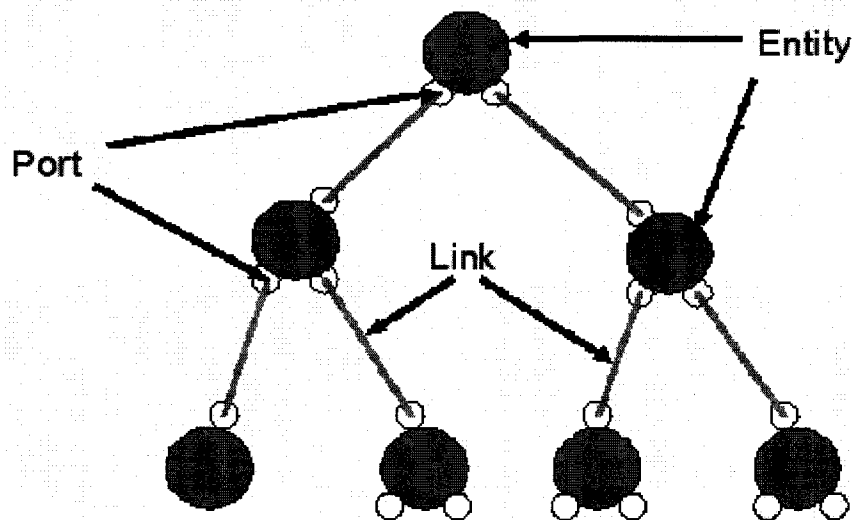


Figure 5.1: A simulation layout

We used the following subclasses of Sim_entity:

Table 5.1: subclasses of Sim_entity

Subclass name	Purpose
RootNodeT11A	to represent the Root node of each tree
IntNodeT11B	to represent the left intermediate node of each tree
IntNodeT11C	to represent the right intermediate node of each tree
LeafNodeT111	to represent the leftmost leaf node of each tree
LeafNodeT112	to represent the second leftmost leaf node of each tree
LeafNodeT113	to represent the second rightmost node of each tree
LeafNodeT114	to represent the rightmost node of each tree

After deciding which objects of each of the above classes are to be present for the simulation, we have to specify their behavior [43] by defining appropriate methods within each class. These objects interact with each other by sending them messages – each corresponding to an event. This means that some objects generate events which trigger methods in the object receiving the message. We need to override the body() method of the class Sim_entity to provide the subclass objects the needed behavior. In the OMULT architecture the following tasks are done by each individual nodes-

Root node:

The role of a root node is to send/receive data to/from its intermediate nodes. For instance, in our case, it sums the partial results during the multiplication of matrix elements, when it will receive the partial content of the matrix elements from the intermediate nodes.

Intermediate nodes:

The role of an intermediate node, is to send/ receive data to/from it's root node as well as its leaf nodes. Another most important task is to compute the sum of the results of the matrix elements from the partial results from the leaf nodes and send this partial result to the root node.

Leaf nodes:

The role of a leaf node is to send/receive data to the intermediate nodes in the same tree as well as to the leaf nodes of the different trees to which it is connected by an inter block link. After the broadcast of all the matrix elements to all the leaf nodes, then each leaf node carries out the requisite multiplication on the appropriate elements of the A and B matrices. After multiplying the elements, the results are sent to the intermediate nodes.

5.4.2 Connecting nodes

Connecting nodes within a tree and as well as the leaf nodes of different trees (Horizontal and Vertical links) are represented by linking the ports by using the method- `Sim_system.link_ports()` available from the Sim-Java package, where the ports are used for linking the nodes[43].

5.4.3 Communication between the nodes

The nodes that are connected by ports can communicate by using the built-in methods `sim_schedule (Sim_port port, int tag, Integer data)` available from the Sim-Java package.

The following examples of a simulation show how SimJava works.

Example 1:

```
1. import eduni.simjava.*;
2.   public class Simulation
3.   {   public static void main(String args[])
4.       {
5.           Sim_system.initialise();
6.   System.out.println("Start time" + Sim_system.clock());
```



```

7. /* nodes are added by adding entities to the Sim_system */
8. Sim_system.add (new RootNode ("T11(A)"));
9. Sim_system.add(new IntNodeT11B("T11(B)"));
10. Sim_system.add(new IntNodeT11C("T11(C)"));
11. Sim_system.add(new LeafNodeT111("T11(1)"));
12. Sim_system.add(new LeafNodeT112("T11(2)"));
13. Sim_system.add(new LeafNodeT113("T11(3)"));
14. Sim_system.add(new LeafNodeT114("T11(4)"));
15. // nodes are connected by linking entities using port to the Sim_system
16. Sim_system.link_ports("T11(A)", "lT11APort", "T11(B)", "iT11BPort");
17. Sim_system.link_ports("T11(A)", "rT11APort", "T11(C)", "iT11CPort");
18. Sim_system.link_ports("T11(B)", "lT11BPort", "T11(1)", "iT111Port");
19. Sim_system.link_ports("T11(B)", "rT11BPort", "T11(2)", "iT112Port");
20. Sim_system.link_ports("T11(C)", "lT11CPort", "T11(3)", "iT113Port");
21. Sim_system.link_ports("T11(C)", "rT11CPort", "T11(4)", "iT114Port");
22. Sim_system.link_ports("T11(2)", "hT112Port", "T12(1)", "hT121Port");
23. Sim_system.link_ports("T11(2)", "vT112Port", "T21(1)", "vT211Port");
24. Sim_system.link_ports("T11(3)", "hT113Port", "T13(1)", "hT131Port");
25. Sim_system.link_ports("T11(3)", "hT113Port", "T13(1)", "h1T131Port");
26. Sim_system.link_ports("T11(3)", "vT113Port", "T31(1)", "vT311Port");
27. Sim_system.link_ports("T11(3)", "vT113Port", "T31(1)", "v1T311Port");
28. Sim_system.link_ports("T11(4)", "hT114Port", "T14(1)", "hT141Port");
29. Sim_system.link_ports("T11(4)", "vT114Port", "T41(1)", "vT411Port");
30. Sim_system.run ();
31. System.out.println("End time" + Sim_system.clock());
32.     } // main close
33.     } // class close

```

In this simple simulation four steps are required [42] -

- Initialise sim_system.
- Make an instance for each entity
- Link the entities ports
- Run the simulation

The first line imports all the requisite classes in the SimJava package. The Sim_system object is initialized done, at the start of simulation, in line 5 by invoking the Sim_system.initialise() method. The objects are then created in line 8-14 by using the Sim_system.add() method. These entities are linked together by invoking the

Sim_system.link_port() method in lines 16-29. Finally the simulation is run by calling the Sim_system.run() method in line 30.

The classes for the entities are derived from the Sim_entity class. The code for the node T112 is given below with example 2. Rest of the classes are similar and given in appendix D.

Example 2:

```
1. class LeafNodeT112 extends Sim_entity
2. {
3.     private Sim_port iT112Port, hT112Port, vT112Port;
4.     int a12=2, b21=21;
5.     int x=0, y=0, z=0;
6.     public LeafNodeT112(String name)
7.     {
8.         super(name);
9.         iT112Port=new Sim_port("iT112Port");
10.        add_port(iT112Port);
11.        hT112Port=new Sim_port("hT112Port");
12.        add_port(hT112Port);
13.        vT112Port=new Sim_port("vT112Port");
14.        add_port(vT112Port)
15.    }
16.    public void body()
17.    {
18.        Integer i1 = new Integer(0);
19.        Integer i2 = new Integer(0);
20.        Integer i3 = new Integer(0);
21.        Integer i4 = new Integer(0);
```

```

22. sim_schedule("vT112Port", 0.01, 2, new Integer(a12));
23. sim_trace(1, "leafNodeT221");
24. sim_hold(0.01);
25. sim_schedule("hT112Port", 0.04, 21, new Integer(b21));
26. sim_trace(1, "leafNodeT112");
27. sim_hold(0.01);
28. Sim_event ev= new Sim_event();
29. {
30.     z= a12 * b21;
31.     if (z>0)
32.     {
33.         System.out.println("T112      z -> " + z);
34.         sim_schedule("iT112Port", 0.16, 112, new Integer(z));
35.         sim_trace(1, "leafNodeT112");
36.         sim_hold(0.01);
37.     }
38. }
39. while(true)
40. {
41.     sim_wait(ev);
42.     if(ev.get_tag() == 18)
43.     {
44.         i4=(Integer)ev.get_data();
45.         System.out.println("T11(2)b12 -> " + i4);
46.         sim_schedule("hT112Port", 0.1, 18, i4);
47.         sim_trace(1, "leafNodeT112");
48.         sim_hold(0.01);
49.     }
50.     if(ev.get_tag() == 17)
51.     {
52.         i2=(Integer)ev.get_data();
53.         System.out.println("T11(2)b11-> " + i2);
54.         sim_schedule("vT112Port", 0.09, 17, i2);
55.         sim_trace(1, "leafNodeT112");
56.         sim_hold(0.01);
57.     }
58.     if(ev.get_tag() == 5)
59.     {
60.         i3=(Integer)ev.get_data();
61.         System.out.println("T11(2)a21-> " + i3);
62.         sim_schedule("vT112Port", 0.06, 5, i3); //0.07
63.         sim_trace(1, "leafNodeT112");
64.         sim_hold(0.01);
65.     }

```

```

66.         if(ev.get_tag() == 1)
67.         {
68.             i1=(Integer)ev.get_data();
69.             sim_schedule("hT112Port", 0.06, 1, i1); //0.06
70.             sim_trace(1, "leafNodeT112");
71.             sim_hold(0.01);
72.             System.out.println("T11(2)a11-> " + i1);
73.         }
74.     } //while close
75. } // body close

76. } //class close

```

The constructor method in line 6 first calls the Sim_entity's constructor, super(name) in order to invoke the superclass constructor. Then it creates ports (line 9-14) iT112Port, hT112Port, vT112Port and adds the ports to its list of ports. These ports are linked to the appropriate objects through their requisite ports in the main() function as shown in example 1.

The body() of the T112 entity includes the following the most important methods of SimJava [43]:

sim_schedule(Sim_port portName, int tag, Integer data) - send data to the entity connected to the port with the given tag.

sim_hold(double d) –hold the data for d simulation time units.

sim_trace(int level, String msg) - adds the message msg to the trace file

5.5 Simulation Result

We have successfully simulated OMULT network and broadcast data elements for matrix A of size 4×4 and a matrix B of size 4×4 over the leaf nodes of the OMULT trees. Multiplying matrix A and matrix B we have got the product matrix C of size 4×4 .

The simulation time required for getting the product matrix C of size 4×4 is - 0.88 time units.

5.6 Critical Review of the Simulation

In this thesis, we have theoretically investigated the following algorithms for efficient implementation on the OMULT architecture:

- matrix multiplication of two matrices having fixed size,
- matrix multiplication of two matrices having arbitrary sizes,
- computing the prefix-sum of a series containing n^2 integers,
- computing the prefix-sum of a series containing n^3 integers
- number of fundamental computational geometry problems.

We have indicated, in chapter 3 and 4, that these algorithms are efficient with respect to the implementations of some of these algorithms on the OTIS mesh – a comparable architecture based on the same idea of opto-electronic technology.

We have successfully used the SimJava simulation tool to support the model and to understand the behaviors of an OMULT network.

The simulation for multiprocessor systems was complicated because of so many interactions between parallel software and hardware. There are an increasing number of tools available to simulate the parallel and distribution systems and it is very difficult to select the correct tools for simulating the application. By using the SimJava package we were able to represent the OMULT architecture in a realistic manner. One crucial issue in the designing of multi-processor simulation was to decide how the processors should communicate with each other. SimJava simulation tool provides efficient methods to produce ports to provide connections among processors so that data can be transferred quickly between processors that need to share data.

The SimJava simulator we wrote is included as an appendix(Appendix C).

Chapter 6: Conclusions and Future Work

6.1 Conclusions

In this thesis, we have investigated the following algorithms for efficient implementation on the OMULT architecture:

- matrix multiplication of two matrices having fixed size,
- matrix multiplication of two matrices having as arbitrary sizes,
- computing the prefix-sum of a series containing n^2 integers,
- computing the prefix-sum of a series containing n^3 integers
- a number of fundamental computational geometry problems.

These problems have not been studied in the literature. It is interesting to note from Table 6.1 given below that our algorithms are efficient with respect to recent implementations of some of these algorithms on the OTIS mesh – a comparable architecture based on the same idea of opto-electronic technology.

Finally we have implemented our algorithm for matrix multiplication using the SimJava simulation tool. In our experience SimJava is a convenient environment for testing such parallel algorithms.

Table 6.1: Performance of algorithms on the OMULT

A summary of the performances of some algorithms, along with the topological properties of the OTIS mesh and the OMULT network has been given in table 6.1. In the table Electronic links are identified by E and optical links by O .

Table 6.1: Comparisons between OTIS-Mesh and OMULT Network

	OTIS-Mesh	OMULT
Number of Nodes (N)	$N = n^4$	$N = 2n^3 - n^2$
Diameter	$4n - 3$	$6 \log n + 2$
Broadcast time	$4n - 1 (E) + 1 (O)$	$6 \log n (E) + 2 (O)$
Prefix Sum time	$7n - 1 (E) + 2 (O)$	$10 \log n(E) + 10(O)$
(# of elements)	(for n^4 elements)	(for n^2 elements)
Matrix multiplication	$O(n^4)$	$O(n^3 \log n)$
Time (size of matix)	($n^4 \times n^4$ matrices)	($n^2 \times n^2$ matrices)
Convex hull	$18\sqrt{n} (E) + 3(O)$	$14 \log n(E) + 10(O)$
Smallest enclosing box	$26\sqrt{n} (E) + 3(O)$	$18 \log n(E) + 16(O)$
ECDF	$4(\sqrt{n} - 1) (E) + 3(O)$	$6 \log n(E) + 3(O)$
All-nearest Neighbor	$4(\sqrt{n} - 1) (E) + 3(O)$	$6 \log n(E) + 3(O)$

6.2 Future Works

We are studying the implementation of a number of other algorithms on the OMULT architecture. Algorithm mapping for the common basic operations involved in real-life applications for numerical and scientific processing, image and signal processing can be more efficiently done using the OMULT network than on the OTIS-Mesh, with comparable investments on establishing optical links among the processor nodes, and lesser cost for the electronic links. Also, the topology of the proposed OMULT network is very simple, making it particularly attractive for parallel computing.

Appendix A: List of symbols

A, B, C - registers

a_1, a_2, \dots, a_N - values in a series for computing the prefix sum

a, b, c, d, e, f, g – set of points

$a_{i1}, a_{i2}, \dots, a_{in}$ – elements of row i of the matrix A

a_{ik} - elements of matrix B

b_{kj} - elements of matrix A

$b_{1i}, b_{2i}, \dots, b_{ni}$ – elements of column i of the matrix B

c_{ij} - elements of matrix C

d_1, d_2, \dots, d_n - data elements

(G, P) - G identifies the group and P identifies the processor within the group

$K \times N$ - size of matrix B

$M \times K$ - size of matrix A

$M \times N$ - size of matrix C

$n \times n$ - array of tree of the OMULT system

N - total number of nodes of the OMULT network

n - number of leaf nodes in a tree within the network

$n-1$ – internal nodes

P_0, P, \dots, P_{N-1} - series of processors

$p_{i0}, p_{i1}, \dots, p_{i,n-2}$ – set of points(S)

p_1, p_2, \dots, p_n – set of points

(x_i, y_i) - coordinate of points p_i

(p_l, p_r) - hull edge

Appendix B: Glossary of important terms

Arc connectivity: the minimum number of arcs that have to be removed from the network to cut it into two disconnected networks. Higher connectivity is better since it reduces the contention for links.

All nearest neighbor problem: is to find the minimum distance between any two points.

Bisection width: the minimum number of links that need to be removed to break the network into two equal halves.

Cost: the number communication links required by the network.

Convex hull: problem is to find a hull that surrounds and encloses a given set of points.

Diameter: largest possible value of the shortest path between any two processors in an interconnection network.

Extreme point: if the counterclockwise angle between any pair of consecutive vectors is more than then the 180° then the point is called extreme point.

Empirical Cumulative Distribution Function: problem is to find the number of points dominated by each point

Interconnection network is used to provide connections among processors so that data can be transferred between processors.

Optical network: is a digital communication system that uses light waves as the medium for transmits data.

Parallel computer: (also called a multi-processor machine) is a machine that consists of a collection of processors or processing units, that cooperate, to solve a problem, by working simultaneously on different parts of that problem .

Polar angle: is the counterclockwise angle between two vectors.

Shortest path: is the smallest number of links needed to communicate between two processors.

SIMJAVA package: is a process based simulation tool based on Java. A SIMJAVA simulation is a collection of entities each running on its own thread.

SIMD (Single instruction multiple data): is a parallel computer consists of a number of processors that operate under the control of a single instruction issued by a central control unit.

Smallest enclosing box: problem is to find the rectangle with the minimum area that encloses all the points

Appendix C: Simulation

```
/*
 * Simulation of Matrix Multiplication on OMULT architecture
 *Using SimJava simulation package
 */

import java.awt.*;
import eduni.simjava.*;
import javax.swing.*;
import java.awt.event.*;

/*****Tree T11*****/

/*****Root Node*****/

class RootNodeT11A extends Sim_entity
{
    private Sim_port lT11APort, rT11APort;
    int x=0, y=0, z=0;
    public RootNodeT11A(String name)
    {
        super(name);
        lT11APort=new Sim_port("lT11APort");
        add_port(lT11APort);
        rT11APort=new Sim_port("rT11APort");
        add_port(rT11APort);
    }

    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer x1 = new Integer(0);
        Integer x2 = new Integer(0);
        Sim_event ev= new Sim_event();

        while (true)
        {
            sim_wait(ev);
            if(ev.get_tag() == 1)
            {
                i1=(Integer)ev.get_data();
                sim_schedule("rT11APort", 0.04, 1, i1);
                sim_trace(1, "leafNodeT11A");
                sim_hold(0.01);
            }
        }
    }
}
```

```

    }
    if(ev.get_tag() == 17)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("rT11APort", 0.07, 17, i2); //0.7
        sim_trace(1, "leafNodeT11A");
        sim_hold(0.01);
    }

    if(ev.get_tag() == 100) x1=(Integer)ev.get_data();
    if(ev.get_tag() == 200) x2=(Integer)ev.get_data();
    x= x1.intValue();
    if (x>0)
        y= x2.intValue();
    if (y>0)
        z= x + y;
    if (z>0)
        System.out.println("T11A          z ->  " + z);
    } //while close
} // body close
} //class close

/***** Intermediate Node*****/

class IntNodeT11B extends Sim_entity
{
    private Sim_port iT11BPort, lT11BPort,rT11BPort, r1T11BPort;
    int x=0, y=0, z=0;

    public IntNodeT11B (String name)
    {
        super(name);
        iT11BPort=new Sim_port("iT11BPort");
        add_port(iT11BPort);
        lT11BPort=new Sim_port("lT11BPort");
        add_port(lT11BPort);
        rT11BPort=new Sim_port("rT11BPort");
        add_port(rT11BPort);
        r1T11BPort=new Sim_port("r1T11BPort");
        add_port(r1T11BPort);
    }
    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer x1 = new Integer(0);
        Integer x2 = new Integer(0);

        Sim_event ev= new Sim_event();
        while(true)
        {

```

```

sim_wait(ev);
if(ev.get_tag() == 1)
{
    i1=(Integer)ev.get_data();
    sim_schedule("iT11BPort", 0.03, 1, i1);
    sim_trace(1, "leafNodeT11B");
    sim_hold(0.01);
    sim_schedule("rT11BPort", 0.05, 1, i1); //0.5
    sim_trace(1, "leafNodeT11B");
    sim_hold(0.01);
}
if(ev.get_tag() == 17)
{
    i2=(Integer)ev.get_data();
    sim_schedule("iT11BPort", 0.06, 17, i2);
    sim_trace(1, "leafNodeT11B");
    sim_hold(0.01); //
    sim_schedule("rT11BPort", 0.06, 17, i2); //0.8
    sim_trace(1, "leafNodeT11B");
    sim_hold(0.01);
}
if(ev.get_tag() == 111) x1=(Integer)ev.get_data();
if(ev.get_tag() == 112) x2=(Integer)ev.get_data();

x= x1.intValue();
if (x>0)
    y= x2.intValue();
if (y>0)
    z= x + y;
if (z>0)
{
    System.out.println("T11B          z -> " + z);
    sim_schedule("iT11BPort", 0.17, 100, new Integer(z));
    sim_trace(1, "leafNodeT11B");
    sim_hold(0.01);
}
} //while close
} //body close
} //class close

class IntNodeT11C extends Sim_entity
{
    private Sim_port iT11CPort, lT11CPort, rT11CPort;
    int x=0, y=0, z=0;

    public IntNodeT11C (String name)
    {
        super(name);
        iT11CPort=new Sim_port("iT11CPort");
        add_port(iT11CPort);
        lT11CPort=new Sim_port("lT11CPort");

```

```

add_port(lT11CPort);
rT11CPort=new Sim_port("rT11CPort");
add_port(rT11CPort);
}

public void body()
{
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer x1 = new Integer(0);
    Integer x2 = new Integer(0);
    Sim_event ev= new Sim_event();

while(true)
{
    sim_wait(ev);
    if(ev.get_tag() == 1)
    {
        i1=(Integer)ev.get_data();
        sim_schedule("lT11CPort", 0.05, 1, i1);
        sim_trace(1, "leafNodeT11C");
        sim_hold(0.01);
        sim_schedule("rT11CPort", 0.05, 1, i1);
        sim_trace(1, "leafNodeT11C");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 17)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("lT11CPort", 0.08, 17, i2);
        sim_trace(1, "leafNodeT11C");
        sim_hold(0.01);
        sim_schedule("rT11CPort", 0.08, 17, i2);
        sim_trace(1, "leafNodeT11C");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 113) x1=(Integer)ev.get_data();
    if(ev.get_tag() == 114) x2=(Integer)ev.get_data();
    x= x1.intValue();
    if (x>0)
        y= x2.intValue();
    if (y>0)
        z= x + y;
    if (z>0)
    {
        System.out.println("T11C          z -> " + z);
        sim_schedule("iT11CPort", 0.17, 200, new Integer(z));
        sim_trace(1, "leafNodeT11C");
        sim_hold(0.01);
    }
}
} //while close

```



```

    }//body close
} //class close

class LeafNodeT111 extends Sim_entity
{
    private Sim_port iT111Port, hT111Port, vT111Port;
    int a11=1, b11=17 ;
    int x=0, y=0, z=0;
    public LeafNodeT111(String name)
    {
        super(name);
        iT111Port=new Sim_port("iT111Port");
        add_port(iT111Port);
        hT111Port=new Sim_port("hT111Port");
        add_port(hT111Port);
        vT111Port=new Sim_port("vT111Port");
        add_port(vT111Port);
    }
    public void body()
    {
        sim_schedule("iT111Port", 0.02, 1, new Integer(a11));
        sim_trace(1, "leafNodeT111");
        sim_hold(0.01);
        sim_schedule("iT111Port", 0.05, 17, new Integer(b11));
        sim_trace(1, "leafNodeT111");
        sim_hold(0.01);
        {
            z= a11 * b11;
            if (z>0)
            {
                System.out.println("T111          z ->  " + z);
                sim_schedule("iT111Port", 0.15, 111, new Integer(z));
                sim_trace(1, "leafNodeT111");
                sim_hold(0.01);
            }
        }
    }
}

class LeafNodeT112 extends Sim_entity
{
    private Sim_port iT112Port, hT112Port, vT112Port;
    int a12=2, b21=21;
    int x=0, y=0, z=0;
    public LeafNodeT112(String name)
    {
        super(name);
        iT112Port=new Sim_port("iT112Port");
        add_port(iT112Port);
        hT112Port=new Sim_port("hT112Port");
        add_port(hT112Port);
        vT112Port=new Sim_port("vT112Port");
    }
}

```

```

    add_port(vT112Port);
}
public void body()
{
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer i3 = new Integer(0);
    Integer i4 = new Integer(0);
    sim_schedule("vT112Port", 0.01, 2, new Integer(a12));
    sim_trace(1, "leafNodeT221");
    sim_hold(0.01);
    sim_schedule("hT112Port", 0.04, 21, new Integer(b21));
    sim_trace(1, "leafNodeT112");
    sim_hold(0.01);

    Sim_event ev= new Sim_event();
    {
        z= a12 * b21;
        if (z>0)
        {
            System.out.println("T112          z -> " + z);
            sim_schedule("iT112Port", 0.16, 112, new Integer(z));
            sim_trace(1, "leafNodeT112");
            sim_hold(0.01);
        }
    }

    while(true)
    {
        sim_wait(ev);
        if(ev.get_tag() == 18)
        {
            i4=(Integer)ev.get_data();
            sim_schedule("hT112Port", 0.1, 18, i4);
            sim_trace(1, "leafNodeT112");
            sim_hold(0.01);
        }
        if(ev.get_tag() == 17)
        {
            i2=(Integer)ev.get_data();
            sim_schedule("vT112Port", 0.09, 17, i2); //check print out
            sim_trace(1, "leafNodeT112");
            sim_hold(0.01);
        }
        if(ev.get_tag() == 5)
        {
            i3=(Integer)ev.get_data();
            sim_schedule("vT112Port", 0.06, 5, i3); //0.07
            sim_trace(1, "leafNodeT112");
            sim_hold(0.01);
        }
    }
}

```

```

if(ev.get_tag() == 1)
{
    i1=(Integer)ev.get_data();
    sim_schedule("hT112Port", 0.06, 1, i1); //0.06
    sim_trace(1, "leafNodeT112");
    sim_hold(0.01);
}
} //while close
}

class LeafNodeT113 extends Sim_entity
{
    private Sim_port iT113Port, hT113Port, vT113Port, i1T113Port, h1T113Port,
    v1T113Port, v2T113Port;
    int a13 = 3, b31=25;
    int x=0, y=0, z=0;
    public LeafNodeT113(String name)
    {
        super(name);
        iT113Port=new Sim_port("iT113Port");
        add_port(iT113Port);
        hT113Port=new Sim_port("hT113Port");
        add_port(hT113Port);
        vT113Port=new Sim_port("vT113Port");
        add_port(vT113Port);
        i1T113Port=new Sim_port("i1T113Port");
        add_port(i1T113Port);
        h1T113Port=new Sim_port("h1T113Port");
        add_port(h1T113Port);
        v1T113Port=new Sim_port("v1T113Port");
        add_port(v1T113Port);
        v2T113Port=new Sim_port("v2T113Port");
        add_port(v2T113Port);
    }

    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);
        Integer i4 = new Integer(0);
        sim_schedule("vT113Port", 0.01, 3, new Integer(a13));
        sim_trace(1, "leafNodeT113");
        sim_hold(0.01);
        sim_schedule("hT113Port", 0.04, 25, new Integer(b31));
        sim_trace(1, "leafNodeT113");
        sim_hold(0.01);
        Sim_event ev= new Sim_event();
        while(true)
        {

```

```

sim_wait(ev);
if(ev.get_tag() == 1)
{
    i1=(Integer)ev.get_data();
    sim_schedule("hT113Port", 0.06, 1, i1); //06
    sim_trace(1, "leafNodeT113");
    sim_hold(0.01);
}
if(ev.get_tag() == 19)
{
    i2=(Integer)ev.get_data();
    sim_schedule("hT113Port", 0.1, 19, i2); //06
    sim_trace(1, "leafNodeT113");
    sim_hold(0.01);
}
if(ev.get_tag() == 9)
{
    i3=(Integer)ev.get_data();
    sim_schedule("vT113Port", 0.06, 9, i3); //07
    sim_trace(1, "leafNodeT113");
    sim_hold(0.01);
}
if(ev.get_tag() == 17)
{
    i4=(Integer)ev.get_data();
    sim_schedule("vT113Port", 0.09, 17, i4);
    sim_trace(1, "leafNodeT112");
    sim_hold(0.01);
}
    z= a13 * b31;
    if (z>0)
    {
        System.out.println("T113          z -> " + z);
        sim_schedule("iT113Port", 0.15, 113, new Integer(z));
        sim_trace(1, "leafNodeT113");
        sim_hold(0.01);
    }
} //while close
}

class LeafNodeT114 extends Sim_entity
{
    private Sim_port iT114Port, hT114Port, vT114Port, v1T114Port;
    int a14= 4, b41=29;
    int x=0, y=0, z=0;
    public LeafNodeT114(String name)
    {
        super(name);
        iT114Port=new Sim_port("iT114Port");
        add_port(iT114Port);
    }
}

```

```

hT114Port=new Sim_port("hT114Port");
add_port(hT114Port);
vT114Port=new Sim_port("vT114Port");
add_port(vT114Port);
v1T114Port=new Sim_port("v1T114Port");
add_port(v1T114Port);
}

public void body()
{
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer i3 = new Integer(0);
    Integer i4 = new Integer(0);
    sim_schedule("vT114Port", 0.01, 4, new Integer(a14));
    sim_trace(1, "leafNodeT114");
    sim_hold(0.01);
    sim_schedule("hT114Port", 0.02, 29, new Integer(b41)); //04
    sim_trace(1, "leafNodeT114");
    sim_hold(0.01);
    Sim_event ev= new Sim_event();

while(true)
{
    sim_wait(ev);
    if(ev.get_tag() == 1)
    {
        i1=(Integer)ev.get_data();
        sim_schedule("hT114Port", 0.06, 1, i1); //06
        sim_trace(1, "leafNodeT114");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 20)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("hT114Port", 0.1, 20, i2);
        sim_trace(1, "leafNodeT114");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 13)
    {
        i3=(Integer)ev.get_data();
        sim_schedule("v1T114Port", 0.07, 13, i3);
        sim_trace(1, "leafNodeT114");
        sim_hold(0.01);
    }

    if(ev.get_tag() == 17)
    {
        i4=(Integer)ev.get_data();
        sim_schedule("vT114Port", 0.09, 17, i4); //09

```

```

sim_trace(1, "leafNodeT112");
sim_hold(0.01);
}
z= a14 * b41;
if (z>0)
{
    System.out.println("T114      z -> " + z);
    sim_schedule("iT114Port", 0.15, 114, new Integer(z));
    sim_trace(1, "leafNodeT114");
    sim_hold(0.01);
}
} //while close
}
}

/*****Tree T12*****/

/*****Root Node*****/

class RootNodeT12A extends Sim_entity
{
    private Sim_port lT12APort, rT12APort;
    int x=0, y=0, z=0;

    public RootNodeT12A(String name)
    {
        super(name);
        lT12APort=new Sim_port("lT12APort");
        add_port(lT12APort);
        rT12APort=new Sim_port("rT12APort");
        add_port(rT12APort);
    }

    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer x1 = new Integer(0);
        Integer x2 = new Integer(0);
        Sim_event ev= new Sim_event();

        while(true)
        {
            sim_wait(ev);
            if(ev.get_tag() == 21)
            {
                i2=(Integer)ev.get_data();
                sim_schedule("rT12APort", 0.07, 21, i2);
                sim_trace(1, "leafNodeT12A");
                sim_hold(0.01);
            }
            if(ev.get_tag() == 5)

```

```

{
    i1=(Integer)ev.get_data();
    sim_schedule("rT12APort", 0.04, 5, i1);
    sim_trace(1, "leafNodeT12A");
    sim_hold(0.01);
}
if(ev.get_tag() == 100) x1=(Integer)ev.get_data();
if(ev.get_tag() == 200) x2=(Integer)ev.get_data();
x= x1.intValue();
if (x>0)
    y= x2.intValue();
    if (y>0)
        z= x + y;
        if (z>0)
            System.out.println("T12A      z -> " + z);
    }
} //while close
} // body close
} //class close

```

```

class IntNodeT12B extends Sim_entity
{
    private Sim_port iT12BPort, lT12BPort, rT12BPort;
    int x=0, y=0, z=0;
    public IntNodeT12B (String name)
    {
        super(name);
        iT12BPort=new Sim_port("iT12BPort");
        add_port(iT12BPort);
        lT12BPort=new Sim_port("lT12BPort");
        add_port(lT12BPort);
        rT12BPort=new Sim_port("rT12BPort");
        add_port(rT12BPort);
    }
    public void body()
    {
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);
        Integer x1 = new Integer(0);
        Integer x2 = new Integer(0);
        Sim_event ev= new Sim_event();
        while (true)
        {
            sim_wait(ev);
            if(ev.get_tag() == 5)
            {
                i3=(Integer)ev.get_data();
                sim_schedule("iT12BPort", 0.03, 5, i3);
                sim_trace(1, "leafNodeT12B");
                sim_hold(0.01);
                sim_schedule("lT12BPort", 0.05, 5, i3); //0.05
            }
        }
    }
}

```

```

sim_trace(1, "leafNodeT12B");
sim_hold(0.01);
}
if(ev.get_tag() == 21)
{
    i2=(Integer)ev.get_data();
    sim_schedule("iT12BPort", 0.06, 21, i2);
    sim_trace(1, "leafNodeT12B");
    sim_hold(0.01);
    sim_schedule("rT12BPort", 0.08, 21, i2);
    sim_trace(1, "leafNodeT12B");
    sim_hold(0.01);
}
if(ev.get_tag() == 121) x1=(Integer)ev.get_data();
if(ev.get_tag() == 122) x2=(Integer)ev.get_data();
x= x1.intValue();
if (x>0)
    y= x2.intValue();
    if (y>0)
        z= x + y;
        if (z>0)
        {
            System.out.println("T12B          z -> " + z);
            sim_schedule("iT12BPort", 0.17, 100, new Integer(z));
            sim_trace(1, "leafNodeT12B");
            sim_hold(0.01);
        }
    }//while close
} //body close
} //class close

class IntNodeT12C extends Sim_entity
{
    private Sim_port iT12CPort, lT12CPort, rT12CPort;
    int x=0, y=0, z=0;

    public IntNodeT12C (String name)
    {
        super(name);
        iT12CPort=new Sim_port("iT12CPort");
        add_port(iT12CPort);
        lT12CPort=new Sim_port("lT12CPort");
        add_port(lT12CPort);
        rT12CPort=new Sim_port("rT12CPort");
        add_port(rT12CPort);
    }

    public void body()
    {
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);

```



```

Integer y1 = new Integer(0);
Integer y2 = new Integer(0);
Sim_event ev= new Sim_event();
while (true)
{
    sim_wait(ev);
    if(ev.get_tag() == 5)
    {
        i3=(Integer)ev.get_data();
        sim_schedule("iT12CPort", 0.05, 5, i3);
        sim_trace(1, "leafNodeT12C");
        sim_hold(0.01);
        sim_schedule("rT12CPort", 0.06, 5, i3); //0.05
        sim_trace(1, "leafNodeT12C");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 21)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("iT12CPort", 0.08, 21, i2);
        sim_trace(1, "leafNodeT12C");
        sim_hold(0.01);
        sim_schedule("rT12CPort", 0.08, 21, i2);
        sim_trace(1, "leafNodeT12C");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 123) y1=(Integer)ev.get_data();
    if(ev.get_tag() == 124) y2=(Integer)ev.get_data();
    x= y1.intValue();
    if (x>0) y= y2.intValue();
    if (y>0) z= x + y;
    if (z>0)
    {
        sim_schedule("iT12CPort", 0.17, 200, new Integer(z));
        sim_trace(1, "leafNodeT12C");
        sim_hold(0.01);
    }
} //while close
} //body close
} //class close

class LeafNodeT121 extends Sim_entity
{
    private Sim_port iT121Port, h1T121Port, hT121Port, vT121Port;
    int x=0, y=0, z=0;

    public LeafNodeT121(String name)
    {
        super(name);
        iT121Port=new Sim_port("iT121Port");
        add_port(iT121Port);
    }
}

```

```

hT121Port=new Sim_port("hT121Port");
add_port(hT121Port);
h1T121Port=new Sim_port("h1T121Port");
add_port(h1T121Port);
vT121Port=new Sim_port("vT121Port");
add_port(vT121Port);
}
public void body()
{
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer i3 = new Integer(0);
    Integer i4 = new Integer(0);
    Sim_event ev= new Sim_event();
while(true)
{
    sim_wait(ev);
    if(ev.get_tag() == 5)
    {
        i3=(Integer)ev.get_data();
        sim_schedule("h1T121Port", 0.06, 5, i3);
        sim_trace(1, "leafNodeT121");
    }
    if(ev.get_tag() == 1) i1=(Integer)ev.get_data();
    if(ev.get_tag() == 18) i4=(Integer)ev.get_data();
    if(ev.get_tag() == 21)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("iT121Port", 0.05, 21, i2);
        sim_trace(1, "leafNodeT121");
        sim_hold(0.01);
    }
    x= i4.intValue();
    if (x>0) y= i1.intValue();
    if (y>0) z= x * y;
    if (z>0)
    {
        sim_schedule("iT121Port", 0.15, 121, new Integer(z));
        sim_trace(1, "leafNodeT121");
        sim_hold(0.01);
    }
} //while close
}
}

class LeafNodeT122 extends Sim_entity
{
    private Sim_port iT122Port, vT122Port;
    int x=0, y=0, z=0;

    public LeafNodeT122(String name)

```

```

{
    super(name);
    iT122Port=new Sim_port("iT122Port");
    add_port(iT122Port);
    vT122Port=new Sim_port("vT122Port");
    add_port(vT122Port);
}

public void body()
{
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer i3 = new Integer(0);
    Integer i4 = new Integer(0);
    Sim_event ev= new Sim_event();
    while(true)
    {
        sim_wait(ev);
        if(ev.get_tag() == 2) i2=(Integer)ev.get_data();
        if(ev.get_tag() == 21) i1=(Integer)ev.get_data();
        sim_schedule("vT122Port", 0.09, 21, i1);
        sim_trace(1, "leafNodeT122");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 22) i4=(Integer)ev.get_data();
    if(ev.get_tag() == 5) i3=(Integer)ev.get_data();
    sim_schedule("iT122Port", 0.02, 5, i3);
    sim_trace(1, "leafNodeT122");
    sim_hold(0.01);
    x= i2.intValue();
    if (x>0) y= i4.intValue();
    if (y>0) z= x * y;
    if (z>0)
    {
        sim_schedule("iT122Port", 0.15, 122, new Integer(z));
        sim_trace(1, "leafNodeT122");
        sim_hold(0.01);
    }
    }//while close
}

class LeafNodeT123 extends Sim_entity
{
    private Sim_port iT123Port, hT123Port, vT123Port;
    int x=0, y=0, z=0;
    public LeafNodeT123(String name)
    {
        super(name);
        iT123Port=new Sim_port("iT123Port");
        add_port(iT123Port);
    }
}

```

```

hT123Port=new Sim_port("hT123Port");
add_port(hT123Port);
vT123Port=new Sim_port("vT123Port");
add_port(vT123Port);
}
public void body()
{
Integer i1 = new Integer(0);
Integer i2 = new Integer(0);
Integer i3 = new Integer(0);
Integer i4 = new Integer(0);
Integer i5 = new Integer(0);
Integer i6 = new Integer(0);
Sim_event ev= new Sim_event();
while(true)
{
sim_wait(ev);
if(ev.get_tag() == 3) i1=(Integer)ev.get_data();
if(ev.get_tag() == 23) i2=(Integer)ev.get_data();
sim_schedule("hT123Port", 0.1, 23, i2);
sim_trace(1, "leafNodeT123");
sim_hold(0.01);
}
if(ev.get_tag() == 5)
{
i3=(Integer)ev.get_data();
sim_schedule("hT123Port", 0.06, 5, i3);
sim_trace(1, "leafNodeT123");
sim_hold(0.01);
}
if(ev.get_tag() == 26) i4=(Integer)ev.get_data();
if(ev.get_tag() == 9) i5=(Integer)ev.get_data();
sim_schedule("vT123Port", 0.07, 9, i5);
sim_trace(1, "leafNodeT123");
sim_hold(0.01);
if(ev.get_tag() == 21)
{
i6=(Integer)ev.get_data();
sim_schedule("vT123Port", 0.09, 21, i6);
sim_trace(1, "leafNodeT122");
sim_hold(0.01);
}
x= i1.intValue();
if (x>0) y= i4.intValue();
if (y>0) z= x * y;
if (z>0)
{
sim_schedule("iT123Port", 0.15, 123, new Integer(z));
sim_trace(1, "leafNodeT123");
sim_hold(0.01);
}
}

```

```

    } //while close
}
}

class LeafNodeT124 extends Sim_entity
{
    private Sim_port iT124Port, hT124Port, vT124Port;
    int x=0, y=0, z=0;

    public LeafNodeT124(String name)
    {
        super(name);
        iT124Port=new Sim_port("iT124Port");
        add_port(iT124Port);
        hT124Port=new Sim_port("hT124Port");
        add_port(hT124Port);
        vT124Port=new Sim_port("vT124Port");
        add_port(vT124Port);
    }
    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);
        Integer i4 = new Integer(0);
        Integer i5 = new Integer(0);
        Integer i6= new Integer(0);
        Sim_event ev= new Sim_event();
        while(true)
        {
            sim_wait(ev);
            if(ev.get_tag() == 5)
            {
                i1=(Integer)ev.get_data();
                sim_schedule("hT124Port", 0.06, 5, i1);
                sim_trace(1, "leafNodeT124");
                sim_hold(0.01);
            }
            if(ev.get_tag() == 4) i2=(Integer)ev.get_data();
            if(ev.get_tag() == 24) i3=(Integer)ev.get_data();
            sim_schedule("hT124Port", 0.1, 24, i3);
            sim_trace(1, "leafNodeT124");
            sim_hold(0.01);

            if(ev.get_tag() == 30) i4=(Integer)ev.get_data();
            if(ev.get_tag() == 13)
            {
                i5=(Integer)ev.get_data();
                sim_schedule("vT124Port", 0.05, 13, i5); //0.07
                sim_trace(1, "leafNodeT124");
                sim_hold(0.01);
            }
        }
    }
}

```

```

    }
    if(ev.get_tag() == 21)
    {
        i6=(Integer)ev.get_data();
        sim_schedule("vT124Port", 0.09, 21, i6);
        sim_trace(1, "leafNodeT122");
        sim_hold(0.01);
    }
    x= i2.intValue();
    if (x>0) y= i4.intValue();
    if (y>0) z= x * y;
    if (z>0)
    {
        sim_schedule("iT124Port", 0.15, 124, new Integer(z));
        sim_trace(1, "leafNodeT124");
        sim_hold(0.01);
    }
    }//while close
}

/*****Coding of Tree T13 to Tree T43 are similar (not shown here)*****/

/*****Tree T44*****/

/*****Root Node*****/

class RootNodeT44A extends Sim_entity
{
    private Sim_port lT44APort, rT44APort;
    int x=0, y=0, z=0;

    public RootNodeT44A(String name)
    {
        super(name);
        lT44APort=new Sim_port("lT44APort");
        add_port(lT44APort);
        rT44APort=new Sim_port("rT44APort");
        add_port(rT44APort);
    }
    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);
        Integer i4 = new Integer(0);
        Integer x1 = new Integer(0);
        Integer x2 = new Integer(0);
        Sim_event ev= new Sim_event();
        while (true)

```

```

{
sim_wait(ev);
if(ev.get_tag() == 16)
{
i1=(Integer)ev.get_data();
sim_schedule("IT44APort", 0.04, 16, i1);
sim_trace(1, "leafNodeT44A");
sim_hold(0.01);
}
if(ev.get_tag() == 32)
{
i2=(Integer)ev.get_data();
sim_schedule("IT44APort", 0.07, 32, i2);
sim_trace(1, "leafNodeT44A");
sim_hold(0.01);
}
if(ev.get_tag() == 100) x1=(Integer)ev.get_data();
if(ev.get_tag() == 200) x2=(Integer)ev.get_data();
x= x1.intValue();
if (x>0) y= x2.intValue();
if (y>0) z= x + y;
if (z>0) System.out.println("T44A      z ->  " + z);
} //while close
} // body close
} //class

```

```

class IntNodeT44B extends Sim_entity
{
private Sim_port iT44BPort, lT44BPort, rT44BPort;
int x=0, y=0, z=0;
public IntNodeT44B (String name)
{
super(name);
iT44BPort=new Sim_port("iT44BPort");
add_port(iT44BPort);
lT44BPort=new Sim_port("lT44BPort");
add_port(lT44BPort);
rT44BPort=new Sim_port("rT44BPort");
add_port(rT44BPort);
}
public void body()
{
Integer i1 = new Integer(0);
Integer i2 = new Integer(0);
Integer i3 = new Integer(0);
Integer i4 = new Integer(0);
Integer x1 = new Integer(0);
Integer x2 = new Integer(0);
Sim_event ev= new Sim_event();
while (true)
{

```

```

sim_wait(ev);
if(ev.get_tag() == 16)
{
    i1=(Integer)ev.get_data();
    sim_schedule("iT44BPort", 0.05, 16, i1);
    sim_trace(1, "leafNodeT44B");
    sim_hold(0.01);
    sim_schedule("rT44BPort", 0.07, 16, i1);
    sim_trace(1, "leafNodeT44B");
    sim_hold(0.01);
}
if(ev.get_tag() == 32)
{
    i2=(Integer)ev.get_data();
    sim_schedule("iT44BPort", 0.08, 32, i2);
    sim_trace(1, "leafNodeT44B");
    sim_hold(0.01);
    sim_schedule("rT44BPort", 0.08, 32, i2);
    sim_trace(1, "leafNodeT44B");
    sim_hold(0.01);
}
if(ev.get_tag() == 441)
{
    x1=(Integer)ev.get_data();
    System.out.println("T44B          x1-> " + x1);
}
if(ev.get_tag() == 442) x2=(Integer)ev.get_data();
x= x1.intValue();
if (x>0) y= x2.intValue();
if (y>0) z= x + y;
if (z>0)
{
    System.out.println("T44B          z -> " + z);
    sim_schedule("iT44BPort", 0.17, 100, new Integer(z));
    sim_trace(1, "leafNodeT44B");
    sim_hold(0.01);
}
} //while close
} //body close
} //class close

class IntNodeT44C extends Sim_entity
{
    private Sim_port iT44CPort, lT44CPort, rT44CPort;
    int x=0, y=0, z=0;
    public IntNodeT44C (String name)
    {
        super(name);
        iT44CPort=new Sim_port("iT44CPort");
        add_port(iT44CPort);
        lT44CPort=new Sim_port("lT44CPort");

```



```

add_port(lT44CPort);
rT44CPort=new Sim_port("rT44CPort");
add_port(rT44CPort);
}
public void body()
{
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer i3 = new Integer(0);
    Integer i4 = new Integer(0);
    Integer x1 = new Integer(0);
    Integer x2 = new Integer(0);
    Sim_event ev= new Sim_event();
while(true)
{
    sim_wait(ev);
    if(ev.get_tag() == 16)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("iT44CPort", 0.03, 16, i2);
        sim_trace(1, "leafNodeT44C");
        sim_hold(0.01);
        sim_schedule("iT44CPort", 0.05, 16, i2);
        sim_trace(1, "leafNodeT44C");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 32)
    {
        i1=(Integer)ev.get_data();
        sim_schedule("iT44CPort", 0.06, 32, i1);
        sim_trace(1, "leafNodeT44C");
        sim_hold(0.01);
        sim_schedule("iT44CPort", 0.06, 32, i1); //0.8
        sim_trace(1, "leafNodeT44C");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 443) x1=(Integer)ev.get_data();
    if(ev.get_tag() == 444) x2=(Integer)ev.get_data();
    {
        x= x1.intValue();
        if (x>0)
            y= x2.intValue();
            if (y>0)
                z= x + y;
                if (z>0)
                {
                    System.out.println("T44C          z -> " + z);
                    sim_schedule("iT44CPort", 0.17, 200, new Integer(z));
                    sim_trace(1, "leafNodeT44C");
                    sim_hold(0.01);
                }
    }
}

```

```

    }
  } //while
} //body close
} //class close

class LeafNodeT441 extends Sim_entity
{
  private Sim_port iT441Port, hT441Port, vT441Port;
  int a41=13, b14=20;
  int x=0, y=0, z=0;

  public LeafNodeT441(String name)
  {
    super(name);
    iT441Port=new Sim_port("iT441Port");
    add_port(iT441Port);
    hT441Port=new Sim_port("hT441Port");
    add_port(hT441Port);
    vT441Port=new Sim_port("vT441Port");
    add_port(vT441Port);
  }

  public void body()
  {
    Integer i1 = new Integer(0);
    Integer i2 = new Integer(0);
    Integer i3 = new Integer(0);
    Integer i4 = new Integer(0);
    Integer i5 = new Integer(0);
    Sim_event ev= new Sim_event();
    sim_schedule("vT441Port", 0.01, 13, new Integer(a41)); //0.1
    sim_trace(1, "leafNodeT441");
    sim_hold(0.01);
    sim_schedule("hT441Port", 0.02, 20, new Integer(b14)); //0.4
    sim_trace(1, "leafNodeT441");
    sim_hold(0.01);
    {
      z= a41 * b14;
      if (z>0)
      {
        System.out.println("T441      z -> " + z);
        sim_schedule("iT441Port", 0.15, 441, new Integer(z));
        sim_trace(1, "leafNodeT441");
        sim_hold(0.01);
      }
    }
    while(true)
    {
      sim_wait(ev);
      if(ev.get_tag() == 16)
      {

```

```

        i1=(Integer)ev.get_data();
        sim_schedule("hT441Port", 0.06, 16, i1);
        sim_trace(1, "leafNodeT441");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 4)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("vT441Port", 0.07, 4, i2);
        sim_trace(1, "leafNodeT441");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 32)
    {
        i3=(Integer)ev.get_data();
        sim_schedule("vT441Port", 0.09, 32, i3);
        sim_trace(1, "leafNodeT441");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 29)
    {
        i4=(Integer)ev.get_data();
        sim_schedule("hT441Port", 0.10, 29, i4);
        sim_trace(1, "leafNodeT441");
        sim_hold(0.01);
    }
} //while close
}
}

class LeafNodeT442 extends Sim_entity
{
    private Sim_port iT442Port, hT442Port, vT442Port;
    int x=0, y=0, z=0;
    int a42=14, b24=24;

    public LeafNodeT442(String name)
    {
        super(name);
        iT442Port=new Sim_port("iT442Port");
        add_port(iT442Port);
        hT442Port=new Sim_port("hT442Port");
        add_port(hT442Port);
        vT442Port=new Sim_port("vT442Port");
        add_port(vT442Port);
    }
    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);

```

```

Integer i4 = new Integer(0);
Sim_event ev= new Sim_event();
sim_schedule("vT442Port", 0.01, 14, new Integer(a42));
sim_trace(1, "leafNodeT442");
sim_hold(0.01);
sim_schedule("hT442Port", 0.04, 24, new Integer(b24));
sim_trace(1, "leafNodeT442");
sim_hold(0.01);
{
    z= a42 * b24;
    if (z>0)
    {
        System.out.println("T442          z -> " + z);
        sim_schedule("iT442Port", 0.14, 442, new Integer(z));
        sim_trace(1, "leafNodeT442");
        sim_hold(0.01);
    }
}
while(true)
{
    sim_wait(ev);
    if(ev.get_tag() == 16)
    {
        i1=(Integer)ev.get_data();
        sim_schedule("hT442Port", 0.06, 16, i1);
        sim_trace(1, "leafNodeT442");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 8)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("vT442Port", 0.07, 8, i2);
        sim_trace(1, "leafNodeT442");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 30)
    {
        i3=(Integer)ev.get_data();
        sim_schedule("hT442Port", 0.10, 30, i3);
        sim_trace(1, "leafNodeT442");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 32)
    {
        i4=(Integer)ev.get_data();
        sim_schedule("vT442Port", 0.09, 32, i4);
        sim_trace(1, "leafNodeT442");
        sim_hold(0.01);
    }
} //while close
}

```

```

}
class LeafNodeT443 extends Sim_entity
{
    private Sim_port iT443Port, hT443Port, vT443Port, v1T443Port;
    int x=0, y=0, z=0;
    int a43=15, b34=28;

    public LeafNodeT443(String name)
    {
        super(name);
        iT443Port=new Sim_port("iT443Port");
        add_port(iT443Port);
        hT443Port=new Sim_port("hT443Port");
        add_port(hT443Port);
        vT443Port=new Sim_port("vT443Port");
        add_port(vT443Port);
        v1T443Port=new Sim_port("v1T443Port");
        add_port(v1T443Port);
    }

    public void body()
    {
        Integer i1 = new Integer(0);
        Integer i2 = new Integer(0);
        Integer i3 = new Integer(0);
        Integer i4 = new Integer(0);
        Sim_event ev= new Sim_event();
        sim_schedule("vT443Port", 0.01, 15, new Integer(a43));
        sim_trace(1, "leafNodeT443");
        sim_hold(0.01);
        sim_schedule("hT443Port", 0.04, 28, new Integer(b34));
        sim_trace(1, "leafNodeT443");
        sim_hold(0.01);
        {
            z= a43 * b34;
            if (z>0)
            {
                System.out.println("T443          z -> " + z);
                sim_schedule("iT443Port", 0.15, 443, new Integer(z));
                sim_trace(1, "leafNodeT443");
                sim_hold(0.01);
            }
        }
        while(true)
        {
            sim_wait(ev);
            if(ev.get_tag() == 16)
            {
                i1=(Integer)ev.get_data();
                sim_schedule("hT443Port", 0.06, 16, i1);
                sim_trace(1, "leafNodeT443");
            }
        }
    }
}

```

```

        sim_hold(0.01);
    }
    if(ev.get_tag() == 12)
    {
        i2=(Integer)ev.get_data();
        sim_schedule("v1T443Port", 0.08, 12, i2); //0.07
        sim_trace(1, "leafNodeT443");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 31)
    {
        i3=(Integer)ev.get_data();
        sim_schedule("hT443Port", 0.1, 31, i3);
        sim_trace(1, "leafNodeT443");
        sim_hold(0.01);
    }
    if(ev.get_tag() == 32)
    {
        i4=(Integer)ev.get_data();
        sim_schedule("vT443Port", 0.09, 32, i4);
        sim_trace(1, "leafNodeT443");
        sim_hold(0.01);
    }
} //while close
}
}

class LeafNodeT444 extends Sim_entity
{
    private Sim_port iT444Port, hT444Port, vT444Port;
    int a44=16, b44=32;
    int x=0, y=0, z=0;
    public LeafNodeT444(String name)
    {
        super(name);
        iT444Port=new Sim_port("iT444Port");
        add_port(iT444Port);
        hT444Port=new Sim_port("hT444Port");
        add_port(hT444Port);
        vT444Port=new Sim_port("vT444Port");
        add_port(vT444Port);
    }
    public void body()
    {
        sim_schedule("iT444Port", 0.02, 16, new Integer(a44));
        sim_trace(1, "leafNodeT444");
        sim_hold(0.01);
        sim_schedule("iT444Port", 0.05, 32, new Integer(b44));
        sim_trace(1, "leafNodeT444");
        sim_hold(0.01);
    }
}

```

```

    z= a44 * b44;
    if (z>0)
    {
        System.out.println("T444      z -> " + z);
        sim_schedule("iT444Port", 0.17, 444, new Integer(z));
        sim_trace(1, "leafNodeT444");
        sim_hold(0.01);
    }
}
}
}

/*****Main Class*****/
public class Simulation
{
    public static void main(String args[])
    {
        Sim_system.initialise();

        //T11
        Sim_system.add(new RootNodeT11A("T11(A)"));
        Sim_system.add(new IntNodeT11B("T11(B)"));
        Sim_system.add(new IntNodeT11C("T11(C)"));
        Sim_system.add(new LeafNodeT111("T11(1)"));
        Sim_system.add(new LeafNodeT112("T11(2)"));
        Sim_system.add(new LeafNodeT113("T11(3)"));
        Sim_system.add(new LeafNodeT114("T11(4)"));

        //T12
        Sim_system.add(new RootNodeT12A("T12(A)"));
        Sim_system.add(new IntNodeT12B("T12(B)"));
        Sim_system.add(new IntNodeT12C("T12(C)"));
        Sim_system.add(new LeafNodeT121("T12(1)"));
        Sim_system.add(new LeafNodeT122("T12(2)"));
        Sim_system.add(new LeafNodeT123("T12(3)"));
        Sim_system.add(new LeafNodeT124("T12(4)"));

        //T13
        Sim_system.add(new RootNodeT13A("T13(A)"));
        Sim_system.add(new IntNodeT13B("T13(B)"));
        Sim_system.add(new IntNodeT13C("T13(C)"));
        Sim_system.add(new LeafNodeT131("T13(1)"));
        Sim_system.add(new LeafNodeT132("T13(2)"));
        Sim_system.add(new LeafNodeT133("T13(3)"));
        Sim_system.add(new LeafNodeT134("T13(4)"));

        //T14
        Sim_system.add(new RootNodeT14A("T14(A)"));
        Sim_system.add(new IntNodeT14B("T14(B)"));
        Sim_system.add(new IntNodeT14C("T14(C)"));
        Sim_system.add(new LeafNodeT141("T14(1)"));

```

```

Sim_system.add(new LeafNodeT142("T14(2)"));
Sim_system.add(new LeafNodeT143("T14(3)"));
Sim_system.add(new LeafNodeT144("T14(4)"));

//T21
Sim_system.add(new RootNodeT21A("T21(A)"));
Sim_system.add(new IntNodeT21B("T21(B)"));
Sim_system.add(new IntNodeT21C("T21(C)"));
Sim_system.add(new LeafNodeT211("T21(1)"));
Sim_system.add(new LeafNodeT212("T21(2)"));
Sim_system.add(new LeafNodeT213("T21(3)"));
Sim_system.add(new LeafNodeT214("T21(4)"));

//T22
Sim_system.add(new RootNodeT22A("T22(A)"));
Sim_system.add(new IntNodeT22B("T22(B)"));
Sim_system.add(new IntNodeT22C("T22(C)"));
Sim_system.add(new LeafNodeT221("T22(1)"));
Sim_system.add(new LeafNodeT222("T22(2)"));
Sim_system.add(new LeafNodeT223("T22(3)"));
Sim_system.add(new LeafNodeT224("T22(4)"));

//T23
Sim_system.add(new RootNodeT23A("T23(A)"));
Sim_system.add(new IntNodeT23B("T23(B)"));
Sim_system.add(new IntNodeT23C("T23(C)"));
Sim_system.add(new LeafNodeT231("T23(1)"));
Sim_system.add(new LeafNodeT232("T23(2)"));
Sim_system.add(new LeafNodeT233("T23(3)"));
Sim_system.add(new LeafNodeT234("T23(4)"));

//T24
Sim_system.add(new RootNodeT24A("T24(A)"));
Sim_system.add(new IntNodeT24B("T24(B)"));
Sim_system.add(new IntNodeT24C("T24(C)"));
Sim_system.add(new LeafNodeT241("T24(1)"));
Sim_system.add(new LeafNodeT242("T24(2)"));
Sim_system.add(new LeafNodeT243("T24(3)"));
Sim_system.add(new LeafNodeT244("T24(4)"));

//T31
Sim_system.add(new RootNodeT31A("T31(A)"));
Sim_system.add(new IntNodeT31B("T31(B)"));
Sim_system.add(new IntNodeT31C("T31(C)"));
Sim_system.add(new LeafNodeT311("T31(1)"));
Sim_system.add(new LeafNodeT312("T31(2)"));
Sim_system.add(new LeafNodeT313("T31(3)"));
Sim_system.add(new LeafNodeT314("T31(4)"));

//T32
Sim_system.add(new RootNodeT32A("T32(A)"));

```



```

Sim_system.add(new IntNodeT32B("T32(B)"));
Sim_system.add(new IntNodeT32C("T32(C)"));
Sim_system.add(new LeafNodeT321("T32(1)"));
Sim_system.add(new LeafNodeT322("T32(2)"));
Sim_system.add(new LeafNodeT323("T32(3)"));
Sim_system.add(new LeafNodeT324("T32(4)"));

//T33
Sim_system.add(new RootNodeT33A("T33(A)"));
Sim_system.add(new IntNodeT33B("T33(B)"));
Sim_system.add(new IntNodeT33C("T33(C)"));
Sim_system.add(new LeafNodeT331("T33(1)"));
Sim_system.add(new LeafNodeT332("T33(2)"));
Sim_system.add(new LeafNodeT333("T33(3)"));
Sim_system.add(new LeafNodeT334("T33(4)"));

//T34
Sim_system.add(new RootNodeT34A("T34(A)"));
Sim_system.add(new IntNodeT34B("T34(B)"));
Sim_system.add(new IntNodeT34C("T34(C)"));
Sim_system.add(new LeafNodeT341("T34(1)"));
Sim_system.add(new LeafNodeT342("T34(2)"));
Sim_system.add(new LeafNodeT343("T34(3)"));
Sim_system.add(new LeafNodeT344("T34(4)"));

//T41
Sim_system.add(new RootNodeT41A("T41(A)"));
Sim_system.add(new IntNodeT41B("T41(B)"));
Sim_system.add(new IntNodeT41C("T41(C)"));
Sim_system.add(new LeafNodeT411("T41(1)"));
Sim_system.add(new LeafNodeT412("T41(2)"));
Sim_system.add(new LeafNodeT413("T41(3)"));
Sim_system.add(new LeafNodeT414("T41(4)"));

//T42
Sim_system.add(new RootNodeT42A("T42(A)"));
Sim_system.add(new IntNodeT42B("T42(B)"));
Sim_system.add(new IntNodeT42C("T42(C)"));
Sim_system.add(new LeafNodeT421("T42(1)"));
Sim_system.add(new LeafNodeT422("T42(2)"));
Sim_system.add(new LeafNodeT423("T42(3)"));
Sim_system.add(new LeafNodeT424("T42(4)"));

//T43
Sim_system.add(new RootNodeT43A("T43(A)"));
Sim_system.add(new IntNodeT43B("T43(B)"));
Sim_system.add(new IntNodeT43C("T43(C)"));
Sim_system.add(new LeafNodeT431("T43(1)"));
Sim_system.add(new LeafNodeT432("T43(2)"));
Sim_system.add(new LeafNodeT433("T43(3)"));
Sim_system.add(new LeafNodeT434("T43(4)"));

```

```

//T44
Sim_system.add(new RootNodeT44A("T44(A)"));
Sim_system.add(new IntNodeT44B("T44(B)"));
Sim_system.add(new IntNodeT44C("T44(C)"));
Sim_system.add(new LeafNodeT441("T44(1)"));
Sim_system.add(new LeafNodeT442("T44(2)"));
Sim_system.add(new LeafNodeT443("T44(3)"));
Sim_system.add(new LeafNodeT444("T44(4)"));

//Link entities using port

//T11
Sim_system.link_ports("T11(A)", "IT11APort", "T11(B)", "iT11BPort");
Sim_system.link_ports("T11(A)", "rT11APort", "T11(C)", "iT11CPort");
Sim_system.link_ports("T11(B)", "IT11BPort", "T11(1)", "iT111Port");
Sim_system.link_ports("T11(B)", "rT11BPort", "T11(2)", "iT112Port");
Sim_system.link_ports("T11(C)", "IT11CPort", "T11(3)", "iT113Port");
Sim_system.link_ports("T11(C)", "rT11CPort", "T11(4)", "iT114Port");
Sim_system.link_ports("T11(2)", "hT112Port", "T12(1)", "hT121Port");
Sim_system.link_ports("T11(2)", "vT112Port", "T21(1)", "vT211Port");
Sim_system.link_ports("T11(3)", "hT113Port", "T13(1)", "hT131Port");
Sim_system.link_ports("T11(3)", "hT113Port", "T13(1)", "h1T131Port");
Sim_system.link_ports("T11(3)", "vT113Port", "T31(1)", "vT311Port");
Sim_system.link_ports("T11(3)", "vT113Port", "T31(1)", "v1T311Port");
Sim_system.link_ports("T11(4)", "hT114Port", "T14(1)", "hT141Port");
Sim_system.link_ports("T11(4)", "vT114Port", "T41(1)", "vT411Port");
//T12
Sim_system.link_ports("T12(A)", "IT12APort", "T12(B)", "iT12BPort");
Sim_system.link_ports("T12(A)", "rT12APort", "T12(C)", "iT12CPort");
Sim_system.link_ports("T12(B)", "IT12BPort", "T12(1)", "iT121Port");
Sim_system.link_ports("T12(B)", "rT12BPort", "T12(2)", "iT122Port");
Sim_system.link_ports("T12(C)", "IT12CPort", "T12(3)", "iT123Port");
Sim_system.link_ports("T12(C)", "rT12CPort", "T12(4)", "iT124Port");
Sim_system.link_ports("T12(1)", "h1T121Port", "T11(2)", "h1T112Port");
Sim_system.link_ports("T12(2)", "vT122Port", "T22(1)", "vT221Port");
Sim_system.link_ports("T12(3)", "hT123Port", "T13(2)", "hT132Port");
Sim_system.link_ports("T12(3)", "vT123Port", "T32(1)", "vT321Port");
Sim_system.link_ports("T12(4)", "hT124Port", "T14(2)", "hT142Port");
Sim_system.link_ports("T12(4)", "vT124Port", "T42(1)", "vT421Port");

//T13
Sim_system.link_ports("T13(A)", "IT13APort", "T13(B)", "iT13BPort");
Sim_system.link_ports("T13(A)", "rT13APort", "T13(C)", "iT13CPort");
Sim_system.link_ports("T13(B)", "IT13BPort", "T13(1)", "iT131Port");
Sim_system.link_ports("T13(B)", "rT13BPort", "T13(2)", "iT132Port");
Sim_system.link_ports("T13(C)", "IT13CPort", "T13(3)", "iT133Port");
Sim_system.link_ports("T13(C)", "rT13CPort", "T13(4)", "iT134Port");
Sim_system.link_ports("T13(2)", "vT132Port", "T23(1)", "vT231Port");
Sim_system.link_ports("T13(2)", "v1T132Port", "T23(1)", "vT231Port");
Sim_system.link_ports("T13(3)", "vT133Port", "T33(1)", "vT331Port");

```

```

Sim_system.link_ports("T13(4)", "hT134Port", "T14(3)", "hT143Port");
Sim_system.link_ports("T13(4)", "hT134Port", "T14(3)", "h1T143Port");
Sim_system.link_ports("T13(4)", "vT134Port", "T43(1)", "vT431Port");

```

```

//T14
Sim_system.link_ports("T14(A)", "iT14APort", "T14(B)", "iT14BPort");
Sim_system.link_ports("T14(A)", "rT14APort", "T14(C)", "iT14CPort");
Sim_system.link_ports("T14(B)", "iT14BPort", "T14(1)", "iT141Port");
Sim_system.link_ports("T14(B)", "rT14BPort", "T14(2)", "iT142Port");
Sim_system.link_ports("T14(C)", "iT14CPort", "T14(3)", "iT143Port");
Sim_system.link_ports("T14(C)", "rT14CPort", "T14(4)", "iT144Port");
Sim_system.link_ports("T14(2)", "vT142Port", "T24(1)", "vT241Port");
Sim_system.link_ports("T14(2)", "v1T142Port", "T24(1)", "v1T241Port");
Sim_system.link_ports("T14(3)", "vT143Port", "T34(1)", "vT341Port");
Sim_system.link_ports("T14(3)", "vT143Port", "T34(1)", "v1T341Port");
Sim_system.link_ports("T14(4)", "vT144Port", "T44(1)", "vT441Port");

```

```

//T21
Sim_system.link_ports("T21(A)", "iT21APort", "T21(B)", "iT21BPort");
Sim_system.link_ports("T21(A)", "rT21APort", "T21(C)", "iT21CPort");
Sim_system.link_ports("T21(B)", "iT21BPort", "T21(1)", "iT211Port");
Sim_system.link_ports("T21(B)", "rT21BPort", "T21(2)", "iT212Port");
Sim_system.link_ports("T21(C)", "iT21CPort", "T21(3)", "iT213Port");
Sim_system.link_ports("T21(C)", "rT21CPort", "T21(4)", "iT214Port");
Sim_system.link_ports("T21(2)", "hT212Port", "T22(1)", "hT221Port");
Sim_system.link_ports("T21(3)", "hT213Port", "T23(1)", "hT231Port");
Sim_system.link_ports("T21(3)", "vT213Port", "T31(2)", "vT312Port");
Sim_system.link_ports("T21(3)", "v1T213Port", "T31(2)", "vT312Port");
Sim_system.link_ports("T21(4)", "hT214Port", "T24(1)", "hT241Port");
Sim_system.link_ports("T21(4)", "vT214Port", "T41(2)", "vT412Port");
Sim_system.link_ports("T21(4)", "v1T214Port", "T41(2)", "v1T412Port");

```

```

//T22
Sim_system.link_ports("T22(A)", "iT22APort", "T22(B)", "iT22BPort");
Sim_system.link_ports("T22(A)", "rT22APort", "T22(C)", "iT22CPort");
Sim_system.link_ports("T22(B)", "iT22BPort", "T22(1)", "iT221Port");
Sim_system.link_ports("T22(B)", "rT22BPort", "T22(2)", "iT222Port");
Sim_system.link_ports("T22(C)", "iT22CPort", "T22(3)", "iT223Port");
Sim_system.link_ports("T22(C)", "rT22CPort", "T22(4)", "iT224Port");
Sim_system.link_ports("T22(3)", "hT223Port", "T23(2)", "hT232Port");
Sim_system.link_ports("T22(3)", "vT223Port", "T32(2)", "vT322Port");
Sim_system.link_ports("T22(4)", "hT224Port", "T24(2)", "hT242Port");
Sim_system.link_ports("T22(4)", "vT224Port", "T42(2)", "vT422Port");

```

```

//T23
Sim_system.link_ports("T23(A)", "iT23APort", "T23(B)", "iT23BPort");
Sim_system.link_ports("T23(A)", "rT23APort", "T23(C)", "iT23CPort");
Sim_system.link_ports("T23(B)", "iT23BPort", "T23(1)", "iT231Port");
Sim_system.link_ports("T23(B)", "rT23BPort", "T23(2)", "iT232Port");

```

```

Sim_system.link_ports("T23(C)", "lT23CPort", "T23(3)", "iT233Port");
Sim_system.link_ports("T23(C)", "rT23CPort", "T23(4)", "iT234Port");
Sim_system.link_ports("T23(3)", "vT233Port", "T33(2)", "vT332Port");
Sim_system.link_ports("T23(4)", "hT234Port", "T24(3)", "hT243Port");
Sim_system.link_ports("T23(4)", "vT234Port", "T43(2)", "vT432Port");

```

```
//T24
```

```

Sim_system.link_ports("T24(A)", "lT24APort", "T24(B)", "iT24BPort");
Sim_system.link_ports("T24(A)", "rT24APort", "T24(C)", "iT24CPort");
Sim_system.link_ports("T24(B)", "lT24BPort", "T24(1)", "iT241Port");
Sim_system.link_ports("T24(B)", "rT24BPort", "T24(2)", "iT242Port");
Sim_system.link_ports("T24(C)", "lT24CPort", "T24(3)", "iT243Port");
Sim_system.link_ports("T24(C)", "rT24CPort", "T24(4)", "iT244Port");
Sim_system.link_ports("T24(3)", "vT243Port", "T34(2)", "vT342Port");
Sim_system.link_ports("T24(4)", "vT244Port", "T44(2)", "vT442Port");

```

```
//T31
```

```

Sim_system.link_ports("T31(A)", "lT31APort", "T31(B)", "iT31BPort");
Sim_system.link_ports("T31(A)", "rT31APort", "T31(C)", "iT31CPort");
Sim_system.link_ports("T31(B)", "lT31BPort", "T31(1)", "iT311Port");
Sim_system.link_ports("T31(B)", "rT31BPort", "T31(2)", "iT312Port");
Sim_system.link_ports("T31(C)", "lT31CPort", "T31(3)", "iT313Port");
Sim_system.link_ports("T31(C)", "rT31CPort", "T31(4)", "iT314Port");
Sim_system.link_ports("T31(2)", "hT312Port", "T32(1)", "hT321Port");
Sim_system.link_ports("T31(3)", "hT313Port", "T33(1)", "hT331Port");
Sim_system.link_ports("T31(4)", "hT314Port", "T34(1)", "hT341Port");
Sim_system.link_ports("T31(4)", "vT314Port", "T41(3)", "vT413Port");

```

```
//T32
```

```

Sim_system.link_ports("T32(A)", "lT32APort", "T32(B)", "iT32BPort");
Sim_system.link_ports("T32(A)", "rT32APort", "T32(C)", "iT32CPort");
Sim_system.link_ports("T32(B)", "lT32BPort", "T32(1)", "iT321Port");
Sim_system.link_ports("T32(B)", "rT32BPort", "T32(2)", "iT322Port");
Sim_system.link_ports("T32(C)", "lT32CPort", "T32(3)", "iT323Port");
Sim_system.link_ports("T32(C)", "rT32CPort", "T32(4)", "iT324Port");
Sim_system.link_ports("T32(3)", "hT323Port", "T33(2)", "hT332Port");
Sim_system.link_ports("T32(4)", "hT324Port", "T34(2)", "hT342Port");
Sim_system.link_ports("T32(4)", "vT324Port", "T42(3)", "vT423Port");

```

```
//T33
```

```

Sim_system.link_ports("T33(A)", "lT33APort", "T33(B)", "iT33BPort");
Sim_system.link_ports("T33(A)", "rT33APort", "T33(C)", "iT33CPort");
Sim_system.link_ports("T33(B)", "lT33BPort", "T33(1)", "iT331Port");
Sim_system.link_ports("T33(B)", "rT33BPort", "T33(2)", "iT332Port");
Sim_system.link_ports("T33(C)", "lT33CPort", "T33(3)", "iT333Port");
Sim_system.link_ports("T33(C)", "rT33CPort", "T33(4)", "iT334Port");
Sim_system.link_ports("T33(4)", "hT334Port", "T34(3)", "hT343Port");
Sim_system.link_ports("T33(4)", "h1T334Port", "T34(3)", "h1T343Port");
Sim_system.link_ports("T33(4)", "vT334Port", "T43(3)", "vT433Port");
Sim_system.link_ports("T33(4)", "v1T334Port", "T43(3)", "v1T433Port");

```

```
//T34
Sim_system.link_ports("T34(A)", "lT34APort", "T34(B)", "iT34BPort");
Sim_system.link_ports("T34(A)", "rT34APort", "T34(C)", "iT34CPort");
Sim_system.link_ports("T34(B)", "lT34BPort", "T34(1)", "iT341Port");
Sim_system.link_ports("T34(B)", "rT34BPort", "T34(2)", "iT342Port");
Sim_system.link_ports("T34(C)", "lT34CPort", "T34(3)", "iT343Port");
Sim_system.link_ports("T34(C)", "rT34CPort", "T34(4)", "iT344Port");
Sim_system.link_ports("T34(4)", "vT344Port", "T44(3)", "vT443Port");
```

```
//T41
Sim_system.link_ports("T41(A)", "lT41APort", "T41(B)", "iT41BPort");
Sim_system.link_ports("T41(A)", "rT41APort", "T41(C)", "iT41CPort");
Sim_system.link_ports("T41(B)", "lT41BPort", "T41(1)", "iT411Port");
Sim_system.link_ports("T41(B)", "rT41BPort", "T41(2)", "iT412Port");
Sim_system.link_ports("T41(C)", "lT41CPort", "T41(3)", "iT413Port");
Sim_system.link_ports("T41(C)", "rT41CPort", "T41(4)", "iT414Port");
Sim_system.link_ports("T41(2)", "hT412Port", "T42(1)", "hT421Port");
Sim_system.link_ports("T41(3)", "hT413Port", "T43(1)", "hT431Port");
Sim_system.link_ports("T41(4)", "hT414Port", "T44(1)", "hT441Port");
```

```
//T42
Sim_system.link_ports("T42(A)", "lT42APort", "T42(B)", "iT42BPort");
Sim_system.link_ports("T42(A)", "rT42APort", "T42(C)", "iT42CPort");
Sim_system.link_ports("T42(B)", "lT42BPort", "T42(1)", "iT421Port");
Sim_system.link_ports("T42(B)", "rT42BPort", "T42(2)", "iT422Port");
Sim_system.link_ports("T42(C)", "lT42CPort", "T42(3)", "iT423Port");
Sim_system.link_ports("T42(C)", "rT42CPort", "T42(4)", "iT424Port");
Sim_system.link_ports("T42(3)", "hT423Port", "T43(2)", "hT432Port");
Sim_system.link_ports("T42(4)", "hT424Port", "T44(2)", "hT442Port");
```

```
//T43
Sim_system.link_ports("T43(A)", "lT43APort", "T43(B)", "iT43BPort");
Sim_system.link_ports("T43(A)", "rT43APort", "T43(C)", "iT43CPort");
Sim_system.link_ports("T43(B)", "lT43BPort", "T43(1)", "iT431Port");
Sim_system.link_ports("T43(B)", "rT43BPort", "T43(2)", "iT432Port");
Sim_system.link_ports("T43(C)", "lT43CPort", "T43(3)", "iT433Port");
Sim_system.link_ports("T43(C)", "rT43CPort", "T43(4)", "iT434Port");
Sim_system.link_ports("T43(4)", "hT434Port", "T44(3)", "hT443Port");
```

```
//T44
Sim_system.link_ports("T44(A)", "lT44APort", "T44(B)", "iT44BPort");
Sim_system.link_ports("T44(A)", "rT44APort", "T44(C)", "iT44CPort");
Sim_system.link_ports("T44(B)", "lT44BPort", "T44(1)", "iT441Port");
Sim_system.link_ports("T44(B)", "rT44BPort", "T44(2)", "iT442Port");
Sim_system.link_ports("T44(C)", "lT44CPort", "T44(3)", "iT443Port");
Sim_system.link_ports("T44(C)", "rT44CPort", "T44(4)", "iT444Port");
Sim_system.run();
System.out.println("End time " + Sim_system.clock());
System.exit(0);
}
}
```

Appendix D: References

- [1] Selim G. Akl, "The Design and Analysis of Parallel Algorithms", *The Prentice-Hall Inc.*, 1989
- [2] L.Cannon, Ph.D. thesis, Montana State University, *Roseman, MN*, 1969
- [3] j. Choi, "A New Parallel Matrix Multiplication Algorithm on Distributed - Memory Concurrent Computers"
- [4] R.D Chamberlain and R.R Krchnavek, "Topologies and technologies for optically interconnected multicomputers using inverted graphs", *Proceedings of the First International Workshop on Massively Parallel Processing Using Optical Interconnections*, pp. Page(s): 255 - 265, 26-27 April 1994
- [5] D. Crawley, "An Analysis of MIMD Processor Interconnection Networks for Nanoelectronic Systems", *Survey Report*, March 1998
- [6] D. Das, M. De and B. P. Sinha, "A new network topology with multiple meshes", *IEEE Transactions on Computers*, Vol. 48, No. 5, pp. 536-551, May 1999.
- [7] M. Feldman, S. Esener, C. Guest, and S. Lee, "Comparison between Electrical and Free-Space Optical Interconnects Based on Power and Speed Considerations", *Applied Optics*, vol. 27, no. 9, pp. 1,742-1,751, May 1988
- [8] G. Fox, S. Otto, and A. Hey, "Matrix algorithms on a hypercube I: matrix multiplication," *Parallel Computing* 3(1987) pp17-31
- [9] H. Freeman and R. Shapira, "Determining the minimal-area enclosing rectangle for an arbitrary closed curve", *Communications of ACM*, 18:409-413, 1975.

- [10] J. W. Goodman, F. J. Leonberg, S. Y. Kung, and R. A. Athale, "Optical interconnections for VLSI systems", *Proceeding IEEE*, vol. 72, pp. 850-866, July 1984
- [11] R. van de Geijin and J. Watts "SUMMA: Scalable Universal Matrix Multiplication Algorithm" LAPACK Working Note 99, *Technical Report CS-95-286*, University of Tennessee, 1995
- [12] W. Hendrick, O Kibar, P. Marchand, C. Fan, D. V. Blerkom, F. McCormick, I. Cokgor, M. Hansen and S. Esener, "Modeling and optimization of the optical transpose interconnection system" in *Optoelectronic Technology Center, Program Review*, Cornell University, Sept. 1995.
- [13] K. Hwang, "Advanced computer Architecture: Parallelism, Scalability, Programmability", New York: *McGraw-Hill*, 1993.
- [14] J-W. Jang, M. Nigam, V.K. Prasanna, S. Sahni, "Constant time algorithms for computational geometry on the reconfigurable mesh", *IEEE Transactions on Parallel and Distributed Systems*, Vol: 8, Iss: 1, pp: 1 -12, Jan. 1997.
- [15] E. John, F. Hudson and L.K. John, "Hybrid tree: a scalable optoelectronic interconnection network for parallel computing", *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, vol. 7, pp. 466 -474, 6-9 Jan. 1998
- [16] F. Kiamilev, P. Marchand, A.V. Krishnamoorthy, S.C. Esener and S.H. Lee, "Performance comparison between optoelectronic and VLSI multistage interconnection networks", *Journal of Lightwave Technology*, vol. 9 Iss. 12, pp. 1674 -1692 Dec. 1991
- [17] A. Krishnamoorthy, P. Marchand, F. Kiamilev and S. Esener, "Grain-size considerations for optoelectronic multistage interconnection networks", *Applied Optics*, Vol. 31, No. 26, pp. 5480-5507, September 1992.
- [18] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*. San Mateo, CA : Morgan Kaufmann.

- [19] Louri and K. Hwang, "A bit-plane architecture for optical computing with two-dimensional symbolic substitution", *Conference Proceedings. 15th Annual International Symposium on Computer Architecture*, pp. 18 -27, June 1988
- [20] Louri, "Design of an optical content-addressable parallel processor with applications to fast searching and information retrieval", *Proc. Fifth International Parallel Processing Symposium*, pp. 234 -239, 30 April-2 May 1991
- [21] Louri and H. Sung, "A hypercube-based optical interconnection network: a solution to the scalability requirements for massively parallel computers", *Proc. of the First International Workshop on Massively Parallel Processing Using Optical Interconnections*, pp. 81 -93, 26-27 April 1994
- [22] A. Louri and H. Sung, "An optical multi-mesh hypercube: a scalable optical interconnection network for massively parallel computing", *Journal of Lightwave Technology*, Vol: 12, Iss: 4 , pp: 704 -716, April 1994
- [23] Louri and H. Sung, "3D Optical Interconnects for high-speed interchip and interboard communications", *Comput.*, vol. 27, pp. 27-37, Oct. 1994.
- [24] K. Mathur and S.L. Johnsson "Multiplication of Matrices of Arbitrary Shape on a Data Parallel Computer", *Parallel Computing* 20, 919-952
- [25] G. C. Marsden, P. J. Marchand, P. Harvey and S. C. Esener, "Optical transpose interconnection system architectures", *Optical Letters*, Vol. 18, No. 13, pp. 1083-1085, July 1993.
- [26] B. Mukherjee, "Optical Communication Networks", *McGraw-Hill*, ISBN 0-07-044435-8, 1997
- [27] A. Osterloh, "Sorting on the OTIS-Mesh", *Proceeding 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, pp. 269-274, 2000.
- [28] F. P. Preparata and M. I. Shamos, "Computational geometry an introduction", *Springer-Verlag*, 1985.

- [29] S. Rajasekaran and S. Sahni, "Randomized routing, selection and sorting on the OTIS-Mesh optoelectronic computer", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 9, pp. 833-840, 1998.
- [30] D. Sima, T. Fountain and P. Kacsuk, "Advanced Computer Architectures", *Addison Wesley Longman Limited*, 1997
- [31] B.P Sinha and S. Bandyopadhyay," OMULT: An Optical Interconnection System for Parallel Computing", communicated.
- [32] I. D. Scherson and S. Sen, "Parallel sorting in two-dimensional VLSI models of computation", *IEEE Transactions on Computers*, Vol. 38, No. 2, pp. 238-249, February 1989.
- [33] S. Sahni and C.-F. Wang, "BPC permutations on the OTIS-Mesh optoelectronic computer", *Proc. of the 4th International Conference on Massively Parallel Processing using Optical Interconnections (MPPOI '97)*, Montreal, Canada, pp. 130-135, 1997.
- [34] T. S. Wailes and D. G. Meyer, "Multiple Channel Architecture: A New Optical Interconnection Strategy for Massively Parallel Computers", *IEEE Journal of Lightwave Thechnology*, vol. 9, pp. 1702-1716, Dec 1991
- [35] C.-F. Wang and S. Sahni, "Image processing on the OTIS-Mesh optoelectronic computer", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 2, pp. 97-109, 2000.
- [36] C.-F. Wang and S. Sahni, "Matrix multiplication on the OTIS-Mesh optoelectronic computer", *IEEE Transactions on Computers*, Vol. 50, No. 7, pp. 635-646, 2001.
- [37] C-F Wang and S. Sahni, "Computational geometry on the OTIS-Mesh opto-electronic computer", *Proceedings International Conference on Parallel Processing*, pp: 501 –507,18- 21 Aug. 2002,
- [38] C.-F. Wang and S. Sahni, "Basic operations on the OTIS-Mesh optoelectronic computer", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 12, pp. 1226-1236, 1998.

[39] C-F. Wang and S. Sahni, "Matrix multiplication on the OTIS-Mesh optoelectronic computer", Proceedings, *The 6th International Conference on Parallel Interconnects*, pp. 131 - 138, 17-19 Oct. 1999

[40] F. Zane, P. Marchand, R. Paturi and S. Esener, "Scalable network architectures using the optical transpose interconnection system (OTIS)", *Journal of Parallel and Distributed Computing*, Vol. 60, No. 5, pp. 521-538, 2000.

[41] <http://www.cs.rit.edu/~nyj4905/project/first-part.doc>

[42] <http://www.dcs.ed.ac.uk/home/hase/Simjava/SimJava.htm>

[43] http://www.dcs.ed.ac.uk/home/hase/simjava/simjava-1.2/doc/simjava_guide/

VITA AUCTORIS

Name	Mohammad Rabiul Islam
Place of Birth	Narail, Bangladesh
Education	Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh 1987-1991 B.E (Mechanical) University of Windsor, Windsor, Ontario 2000-2001 B. Sc (Computer) University of Windsor, Windsor, Ontario 2002-2004 M. Sc (Computer)